# MCALF

***Release v0.1.1***

**Conor D. MacBride & David B. Jess**

**Jan 08, 2021**

# CONTENTS:

CHAPTER

ONE

# MCALF DOCUMENTATION

Welcome to MCALF's documentation!

MCALF is an open-source Python package for accurately constraining velocity information from spectral imaging observations using machine learning techniques.

These pages document how the package can be interacted with. Some examples are also provided. A `Documentation Index` and a `Module Index` are available.

## 1.1 User Documentation

*License*

MCALF is an open-source Python package for accurately constraining velocity information from spectral imaging observations using machine learning techniques.

This software package is intended to be used by solar physicists trying to extract line-of-sight (LOS) Doppler velocity information from spectral imaging observations (Stokes I measurements) of the Sun. A 'toolkit' is provided that can be used to define a spectral model optimised for a particular dataset.

This package is particularly suited for extracting velocity information from spectral imaging observations where the individual spectra can contain multiple spectral components. Such multiple components are typically present when active solar phenomenon occur within an isolated region of the solar disk. Spectra within such a region will often have a large emission component superimposed on top of the underlying absorption spectral profile from the quiescent solar atmosphere.

A sample model is provided for an IBIS Ca II 8542 Å spectral imaging sunspot dataset. This dataset typically contains spectra with multiple atmospheric components and this package supports the isolation of the individual components such that velocity information can be constrained for each component. Using this sample model, as well as the separate base (template) model it is built upon, a custom model can easily be built for a specific dataset.

The custom model can be designed to take into account the spectral shape of each particular spectrum in the dataset. By training a neural network classifier using a sample of spectra from the dataset labelled with their spectral shapes, the spectral shape of any spectrum in the dataset can be found. The fitting algorithm can then be adjusted for each spectrum based on the particular spectral shape the neural network assigned it.

This package is designed to run in parallel over large data cubes, as well as in serial. As each spectrum is processed in isolation, this package scales very well across many processor cores. Numerous functions are provided to plot the results in a clearly. The MCALF API also contains many useful functions which have the potential of being integrated into other Python packages.

### 1.1.1 Installation

For easier package management we recommend using Miniconda (or Anaconda) and creating a new conda environment to install MCALF inside. To install MCALF using Miniconda, run the following commands in your system's command prompt, or if you are using Windows, in the 'Anaconda Prompt':

```
$ conda config --add channels conda-forge
$ conda config --set channel_priority strict
$ conda install mcalf
```

MCALF is updated to the latest version by running:

```
$ conda update mcalf
```

Alternatively, you can install MCALF using `pip`:

```
$ pip install mcalf
```

### 1.1.2 Testing

First, install the package as usual, and then download the code associated with your installed MCALF version. Unzip the file and navigate to it in the terminal. Run the following command (in the same directory as `setup.py`) to test your installation,

```
$ python -m pytest --cov=mcalf
```

Make sure you are inside the virtual environment where it was installed.

### 1.1.3 Getting Started

The following examples provide the key details on how to use this package. For more details on how to use the particular classes and function, please consult the Code Reference. We plan to expand this section with more examples of this package being used.

#### example1: Basic usage of the package

**`FittingIBIS.ipynb`**

- View code
- `Download FittingIBIS.ipynb`

This file is an IPython Notebook containing examples of how to use the package to accomplish typical tasks.

#### FittingIBIS.pro

- Download FittingIBIS.pro

This file is similar to FittingIBIS.ipynb file, except it written is IDL. It is not recommended to use the IDL wrapper in production, just use it to explore the code if you are familiar with IDL and not Python. If you wish to use this package, please use the Python implementation. IDL is not fully supported in the current version of the code for reasons such as, the Python tuple datatype cannot be passed from IDL to Python, resulting in certain function calls not being possible.

#### config.yml

- Download config.yml

This is an example configuration file containing default parameters. This can be easier than setting the parameters in the code. The file follows the YAML format.

### Labelling Tutorial

This Jupyter notebook provides a simple, semi-automated, method to produce a ground truth data set that can be used to train a neural network for use as a spectral shape classifier in the MCALF package. The following code can be adapted depending on the number of classifications that you want.

*Download LabellingTutorial.ipynb*

Load the required packages

[ ]:
```
import mcalf.models
from mcalf.utils import normalise_spectrum
import numpy as np
from astropy.io import fits
import matplotlib.pyplot as plt
```

Load data files

[ ]:
```
wavelengths = np.loadtxt('wavelengths.csv', delimiter=',')  # Original wavelengths
prefilter_response_wvscl = np.loadtxt('prefilter_response_wvscl.csv', delimiter=',')
prefilter_response_main = np.loadtxt('prefilter_response_main.csv', delimiter=',')

with fits.open('spectral_data.fits') as hdul:  # Raw spectral data
    datacube = np.asarray(hdul[0].data, dtype=np.float64)
```

Initialise the model that will use the labelled data

[ ]:
```
model = mcalf.models.IBIS8542Model(original_wavelengths=wavelengths, prefilter_ref_
↪main=prefilter_response_main,
                                   prefilter_ref_wvscl=prefilter_response_wvscl)
```

Select the points to label

[ ]:
```
i_points, j_points = np.load('labelled_points.npy')
```

Select the spectra to label from the data file

[ ]:
```
raw_spectra = datacube[:, i_points, j_points].T
```

Normalise each spectrum to be in range [0, 1]

[ ]:
```
labelled_spectra = np.empty((len(raw_spectra), len(model.constant_wavelengths)))
for i in range(len(labelled_spectra)):
    labelled_spectra[i] = normalise_spectrum(raw_spectra[i], model=model)
```

### Script to semi-automate the classification process

- Type a number 0 - 4 for assign a classification to the plotted spectrum
- Type 5 to skip and move on to the next spectrum
- Type `back` to move to the previous spectrum
- Type `exit` to give up (keeping ones already done)

The labels are present in the `labels` variable (-1 represents an unclassified spectrum)

[ ]:
```
labels = np.full(len(labelled_spectra), -1, dtype=int)
i = 0
while i < len(labelled_spectra):

    # Show the spectrum to be classified along with description
    plt.figure(figsize=(15, 10))
    plt.plot(labelled_spectra[i])
    plt.show()
    print("i = {}".format(i))
    print("absorption --- both --- emission / skip")
    print("       0    1    2    3    4         5 ")

    # Ask for user's classification
    classification = input('Type [0-4]:')

    try:  # Must be an integer
        classification_int = int(classification)
    except ValueError:
        classification_int = -1  # Try current spectrum again

    if classification == 'back':
        i -= 1  # Go back to the previous spectrum
    elif classification == 'exit':
        break  # Exit the loop, saving labels that were given
    elif 0 <= classification_int <= 4:  # Valid classification
        labels[i] = int(classification)  # Assign the classification to the spectrum
        i += 1  # Move on to the next spectrum
    elif classification_int == 5:
        i += 1  # Skip and move on to the next spectrum
    else:  # Invalid integer classification
        i += 0  # Try current spectrum again
```

Plot bar chart of classification populations

[ ]:
```
unique, counts = np.unique(labels, return_counts=True)
plt.figure()
plt.bar(unique, counts)
plt.title('Number of spectra in each classification')
plt.xlabel('Classification')
```

(continues on next page)

(continued from previous page)

```
plt.ylabel('N_spectra')
plt.show()
```

Overplot the spectra of each classification

[ ]:
```
for classification in unique:
    plt.figure()
    for spectrum in labelled_spectra[labels == classification]:
        plt.plot(model.constant_wavelengths, spectrum)
    plt.title('Classification {}'.format(classification))
    plt.yticks([0, 1])
    plt.show()
```

Save the labelled spectra for use later

[ ]:
```
np.save('labelled_data.npy', labelled_spectra)
np.save('labels.npy', labels)
```

If you are interested in using this package in your research and would like advice on how to use this package, please contact Conor MacBride.

### 1.1.4 Contributing

*Code of Conduct*

If you find this package useful and have time to make it even better, you are very welcome to contribute to this package, regardless of how much prior experience you have. Types of ways you can contribute include, expanding the documentation with more use cases and examples, reporting bugs through the GitHub issue tracker, reviewing pull requests and the existing code, fixing bugs and implementing new features in the code.

You are encouraged to submit any bug reports and pull requests directly to the GitHub repository. If you have any questions regarding contributing to this package please contact Conor MacBride.

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

### 1.1.5 Citation

If you have used this package in work that leads to a publication, we would be very grateful if you could acknowledge your use of this package in the main text of the publication. Please cite the Zenodo DOI for the package version you used. Please also consider integrating your code and examples into the package.

### 1.1.6 License

MCALF is licensed under the terms of the BSD 2-Clause license.

# 1.2 Code Reference

## 1.2.1 MCALF

### mcalf Package

### MCALF: Multi-Component Atmospheric Line Fitting

MCALF is an open-source Python package for accurately constraining velocity information from spectral imaging observations using machine learning techniques.

## 1.2.2 MCALF models

This sub-package contains:

- Base and sample models that can be adapted and fitted to any spectral imaging dataset.
- Models optimised for particular data sets that can be used directly.
- Data structures for storing and exporting the fitted parameters, as well as simplifying the calculation of velocities (*mcalf.models.results*).

### mcalf.models Package

### Classes

| | |
|---|---|
| *FitResult*(fitted_parameters, fit_info) | Class that holds the result of a fit |
| *FitResults*(shape, n_parameters[, time]) | Class that holds multiple fit results in a way that can be easily processed |
| *IBIS8542Model*([stationary_line_core, ...]) | Class for working with IBIS 8542 Å calcium II spectral imaging observations |
| *ModelBase*() | Base class for spectral line model fitting. |

### FitResult

**class** mcalf.models.**FitResult**(*fitted_parameters*, *fit_info*)

Bases: object

Class that holds the result of a fit

**Parameters**

- **fitted_parameters** (*ndarray*) – The parameters fitted.
- **fit_info** (*dict*) – Additional information on the fit including at least 'classification', 'profile', 'success' and 'index'.

**parameters**

The parameters fitted.

**Type** ndarray

**classification**
Classification of the fitted spectrum.

**Type** int

**profile**
Profile of the fitted spectrum.

**Type** str

**success**
Whether the fit was completed successfully.

**Type** bool

**index**
Index ([<time>, <row>, <column>]) of the spectrum in the spectral array.

**Type** list

**__dict__**
Other attributes may be present depending on the *fit_info* used.

### Methods Summary

| | |
|---|---|
| *plot*(model, **kwargs) | Plot the data and fitted parameters |
| *velocity*(model[, vtype]) | Calculate the Doppler velocity of the fit using *model* parameters |

### Methods Documentation

**plot** (*model*, ***kwargs*)
Plot the data and fitted parameters

This calls the *plot* method on *model* but will plot for this FitResult object. See the model's *plot* method for more details.

**Parameters**

- **model** (*child class of ModelBase*) – The model object to plot with.
- ****kwargs** – See the *model.plot* method for more details.

**velocity** (*model*, *vtype='quiescent'*)
Calculate the Doppler velocity of the fit using *model* parameters

**Parameters**

- **model** (*child class of ModelBase*) – The model object to take parameters from.
- **vtype** (*{'quiescent', 'active'}, default='quiescent'*) – The velocity type to find.

**Returns** **velocity** – The calculated velocity.

**Return type** float

### FitResults

**class** mcalf.models.**FitResults**(*shape*, *n_parameters*, *time=None*)

Bases: object

Class that holds multiple fit results in a way that can be easily processed

**parameters**
Array of fitted parameters.

**Type** ndarray of shape (*row*, *column*, *parameter*)

**classifications**
Array of classifications.

**Type** ndarray of int of shape (*row*, *column*)

**profile**
Array of profiles.

**Type** ndarray of str of shape (*row*, *column*)

**success**
Array of success statuses.

**Type** ndarray of bool of shape (*row*, *column*)

**chi2**
Array of chi-squared values.

**Type** ndarray of shape (*row*, *column*)

**time**
Time index that the *FitResult* object refers to (if provided).

**Type** int, default = None

**n_parameters**
Number of parameters in the last dimension of *parameters*.

**Type** int

#### Methods Summary

| | |
|---|---|
| *append*(result) | Append a *FitResult* object to the *FitResults* object |
| *save*(filename[, model]) | Saves the FitResults object to a FITS file |
| *velocities*(model[, row, column, vtype]) | Calculate the Doppler velocities of the fit results using *model* parameters |

### Methods Documentation

**append**(*result*)
Append a *FitResult* object to the *FitResults* object

**Parameters** **result** (*FitResult object*) – *FitResult* object to append.

**save**(*filename*, *model=None*)
Saves the FitResults object to a FITS file

**Parameters**

- **filename** (*file path, file object or file-like object*) – FITS file to write to. If a file object, must be opened in a writeable mode.
- **model** (*child class of mcalf.models.base.ModelBase, optional, default = None*) – If provided, use this model to calculate and include both quiescent and active Doppler velocities.

### Notes

Saves a FITS file to the location specified by *filename*. All the parameters are stored in a separate, named, HDU.

**velocities**(*model*, *row=None*, *column=None*, *vtype='quiescent'*)
Calculate the Doppler velocities of the fit results using *model* parameters

**Parameters**

- **model** (*child class of ModelBase*) – The model object to take parameters from.
- **row** (*int, list, array_like, iterable, optional, default = None*) – The row indices to find velocities for. All if omitted.
- **column** (*int, list, array_like, iterable, optional, default = None*) – The column indices to find velocities for. All if omitted.
- **vtype** (*{'quiescent', 'active'}, default='quiescent'*) – The velocity type to find.

**Returns** **velocities** – The calculated velocities for the specified *row* and *column* positions.

**Return type** ndarray of shape (*row*, *column*)

## IBIS8542Model

**class** mcalf.models.**IBIS8542Model**(*stationary_line_core=None*, *absorption_guess=None*, *emission_guess=None*, *absorption_min_bound=None*, *emission_min_bound=None*, *absorption_max_bound=None*, *emission_max_bound=None*, *absorption_x_scale=None*, *emission_x_scale=None*, *neural_network=None*, *original_wavelengths=None*, *constant_wavelengths=None*, *delta_lambda=None*, *sigma=None*, *prefilter_response=None*, *prefilter_ref_main=None*, *prefilter_ref_wvscl=None*, *config=None*, *output=None*)
Bases: *mcalf.models.base.ModelBase*

Class for working with IBIS 8542 Å calcium II spectral imaging observations

**Parameters**

- **original_wavelengths** (*array_like*) – One-dimensional array of wavelengths that correspond to the uncorrected spectral data.
- **stationary_line_core** (*float, optional, default = 8542.104320687517*) – Wavelength of the stationary line core.
- **absorption_guess** (*array_like, length=4, optional, default = [-1000, stationary_line_core, 0.2, 0.1]*) – Initial guess to take when fitting the absorption Voigt profile.
- **emission_guess** (*array_like, length=4, optional, default = [1000, stationary_line_core, 0.2, 0.1]*) – Initial guess to take when fitting the emission Voigt profile.
- **absorption_min_bound** (*array_like, length=4, optional, default = [-np.inf, stationary_line_core-0.15, 1e-6, 1e-6]*) – Minimum bounds for all the absorption Voigt profile parameters in order of the function's arguments.
- **emission_min_bound** (*array_like, length=4, optional, default = [0, stationary_line_core-0.15, 1e-6, 1e-6]*) – Minimum bounds for all the emission Voigt profile parameters in order of the function's arguments.
- **absorption_max_bound** (*array_like, length=4, optional, default = [0, stationary_line_core+0.15, 1, 1]*) – Maximum bounds for all the absorption Voigt profile parameters in order of the function's arguments.
- **emission_max_bound** (*array_like, length=4, optional, default = [np.inf, stationary_line_core+0.15, 1, 1]*) – Maximum bounds for all the emission Voigt profile parameters in order of the function's arguments.
- **absorption_x_scale** (*array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]*) – Characteristic scale for all the absorption Voigt profile parameters in order of the function's arguments.
- **emission_x_scale** (*array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]*) – Characteristic scale for all the emission Voigt profile parameters in order of the function's arguments.
- **neural_network** (*sklearn.neural_network.MLPClassifier, optional, default = see description*) – The MLPClassifier object that will be used to classify the spectra. Its default value is *MLPClassifier(solver='lbfgs', alpha=1e-3, hidden_layer_sizes=(10, 4), random_state=1)*.
- **constant_wavelengths** (array_like, length same as *original_wavelengths*, optional, default = see description) – The desired set of wavelengths that the spectral data should be rescaled to represent. It is assumed that these have constant spacing, but that may not be a requirement if you specify your own array. The default value is an array from the minimum to the maximum wavelength of *original_wavelengths* in constant steps of *delta_lambda*, overshooting the upper bound if the maximum wavelength has not been reached.
- **delta_lambda** (*float, optional, default = 0.05*) – The step used between each value of *constant_wavelengths* when its default value has to be calculated.
- **sigma** (*list of array_like or bool, length=(2, n_wavelengths), optional, default = [type1, type2]*) – A list of different sigma that are used to weight particular wavelengths along the spectra when fitting. The fitting method will expect to be able to choose a sigma array from this list at a specific index. It's default value is *[generate_sigma(i, constant_wavelengths, stationary_line_core) for i in [1, 2]]*. See *utils.generate_sigma()* for more information. If bool, True will generate the default

sigma value regardless of the value specified in *config*, and False will set *sigma* to be all ones, effectively disabling it.

- **prefilter_response** (*array_like, length=n_wavelengths, optional, default = see note*) – Each constant wavelength scaled spectrum will be corrected by dividing it by this array. If *prefilter_response* is not give, and *prefilter_ref_main* and *prefilter_ref_wvscl* are not given, *prefilter_response* will have a default value of *None*.
- **prefilter_ref_main** (*array_like, optional, default = None*) – If *prefilter_response* is not specified, this will be used along with *prefilter_ref_wvscl* to generate the default value of *prefilter_response*.
- **prefilter_ref_wvscl** (*array_like, optional, default = None*) – If *prefilter_response* is not specified, this will be used along with *prefilter_ref_main* to generate the default value of *prefilter_response*.
- **config** (*str, optional, default = None*) – Filename of a *.yml* file (relative to current directory) containing the initialising parameters for this object. Parameters provided explicitly to the object upon initialisation will override any provided in this file. All (or some) parameters that this object accepts can be specified in this file, except *neural_network* and *config*. Each line of the file should specify a different parameter and be formatted like *emission_guess: '[-inf, wl-0.15, 1e-6, 1e-6]'* or *original_wavelengths: 'original.fits'* for example. When specifying a string, use 'inf' to represent *np.inf* and 'wl' to represent *stationary_line_core* as shown. If the string matches a file, *utils.load_parameter()* is used to load the contents of the file.
- **output** (*str, optional, default = None*) – If the program wants to output data, it will place it relative to the location specified by this parameter. Some methods will only save data to a file if this parameter is not *None*. Such cases will be documented where relevant.

**original_wavelengths**
One-dimensional array of wavelengths that correspond to the uncorrected spectral data.

**Type** array_like

**stationary_line_core**
Wavelength of the stationary line core.

**Type** float, optional, default = 8542.099145376844

**absorption_guess**
Initial guess to take when fitting the absorption Voigt profile.

**Type** array_like, length=4, optional, default = [-1000, stationary_line_core, 0.2, 0.1]

**emission_guess**
Initial guess to take when fitting the emission Voigt profile.

**Type** array_like, length=4, optional, default = [1000, stationary_line_core, 0.2, 0.1]

**absorption_min_bound**
Minimum bounds for all the absorption Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [-np.inf, stationary_line_core-0.15, 1e-6, 1e-6]

**emission_min_bound**
Minimum bounds for all the emission Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [0, -np.inf, 1e-6, 1e-6]

**absorption_max_bound**
Maximum bounds for all the absorption Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [0, stationary_line_core+0.15, 1, 1]

**emission_max_bound**
Maximum bounds for all the emission Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [np.inf, np.inf, 1, 1]

**absorption_x_scale**
Characteristic scale for all the absorption Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]

**emission_x_scale**
Characteristic scale for all the emission Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]

**neural_network**
The MLPClassifier object (or similar) that will be used to classify the spectra. Defaults to a *GridSearchCV* with *MLPClassifier(solver='lbfgs', hidden_layer_sizes=(40,), max_iter=1000)* for best *alpha* selected from *[1e-5, 2e-5, 3e-5, 4e-5, 5e-5, 6e-5, 7e-5, 8e-5, 9e-5]*.

**Type** sklearn.neural_network.MLPClassifier, optional, default = see description

**constant_wavelengths**
The desired set of wavelengths that the spectral data should be rescaled to represent. It is assumed that these have constant spacing, but that may not be a requirement if you specify your own array. The default value is an array from the minimum to the maximum wavelength of *original_wavelengths* in constant steps of *delta_lambda*, overshooting the upper bound if the maximum wavelength has not been reached.

**Type** array_like, length same as *original_wavelengths*, optional, default = see description

**sigma**
A list of different sigma that are used to weight particular wavelengths along the spectra when fitting. The fitting method will expect to be able to choose a sigma array from this list at a specific index. It's default value is *[generate_sigma(i, constant_wavelengths, stationary_line_core) for i in [1, 2]]*. See *utils.generate_sigma()* for more information.

**Type** list of array_like, length=(2, n_wavelengths), optional, default = [type1, type2]

**prefilter_response**
Each constant wavelength scaled spectrum will be corrected by dividing it by this array. If *prefilter_response* is not give, and *prefilter_ref_main* and *prefilter_ref_wvscl* are not given, *prefilter_response* will have a default value of *None*.

**Type** array_like, length=n_wavelengths, optional, default = see note

**output**
If the program wants to output data, it will place it relative to the location specified by this parameter. Some methods will only save data to a file if this parameter is not *None*. Such cases will be documented where relevant.

**Type** str, optional, default = None

**quiescent_wavelength**
The index within the fitted parameters of the absorption Voigt line core wavelength.

**Type** int, default = 1

**active_wavelength**
The index within the fitted parameters of the emission Voigt line core wavelength.

**Type** int, default = 5

### Methods Summary

| | |
|---|---|
| *classify_spectra*([time, row, column, ...]) | Classify the specified spectra |
| *fit*([time, row, column, spectrum, profile, ...]) | Fits the model to specified spectra |
| *plot*([fit, time, row, column, spectrum, ...]) | Plots the data and fitted parameters |
| *plot_separate*(*args, **kwargs) | Plot the fitted profiles separately |
| *plot_subtraction*(*args, **kwargs) | Plot the spectrum with the emission fit subtracted from it |

### Methods Documentation

**classify_spectra**(*time=None*, *row=None*, *column=None*, *spectra=None*, *only_normalise=False*)
Classify the specified spectra

Will also normalise each spectrum such that its intensity will range from zero to one.

**Parameters**

- **time** (*int or iterable, optional, default=None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default=None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default=None*) – The column index. See comment for *time* parameter.
- **spectra** (*ndarray, optional, default=None*) – The explicit spectra to classify. If *only_normalise* is False, this must be 1D.
- **only_normalise** (*bool, optional, default = False*) – Whether the single spectrum given in *spectra* should not be interpolated and corrected.

**Returns** **classifications** – Array of classifications with the same time, row and column indices as *spectra*.

**Return type** ndarray

**See also:**

**train()** Train the neural network

**test()** Test the accuracy of the neural network

**get_spectra()** Get processed spectra from the objects *array* attribute

**fit**(*time=None*, *row=None*, *column=None*, *spectrum=None*, *profile=None*, *sigma=None*, *classifications=None*, *background=None*, *n_pools=None*)
Fits the model to specified spectra

Fits the model to an array of spectra using multiprocessing if requested.

**Parameters**

- **time** (*int or iterable, optional, default=None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default=None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default=None*) – The column index. See comment for *time* parameter.
- **spectrum** (*ndarray, ndim=1, optional, default=None*) – The explicit spectrum to fit the model to.
- **profile** (*str, optional, default = None*) – The profile to fit. (Will infer profile from *classifications* if omitted.)
- **sigma** (*int or array_like, optional, default = None*) – Explicit sigma index or profile. See *_get_sigma* for details.
- **classifications** (*int, optional, default = None*) – Classifications to determine the fitted profile to use (if profile not explicitly given). Will use neural network to classify them if not.
- **background** (*float, optional, default = None*) – If provided, this value will be subtracted from the explicit spectrum provided in *spectrum*. Will not be applied to spectra found from the indices, use the *load_background* method instead.
- **n_pools** (*int, optional, default = None*) – The number of processing pools to calculate the fitting over. This allocates the fitting of different spectra to *n_pools* separate worker processes. When processing a large number of spectra this will make the fitting process take less time overall. It also distributes such that each worker process has the same ratio of classifications to process. This should balance out the workload between workers. If few spectra are being fitted, performance may decrease due to the overhead associated with splitting the evaluation over separate processes. If *n_pools* is not an integer greater than zero, it will fit the spectrum with a for loop.

**Returns** **result** – Outcome of the fits returned as a list of FitResult objects

**Return type** list of FitResult, length=n_spectra

**plot** (*fit=None*, *time=None*, *row=None*, *column=None*, *spectrum=None*, *classification=None*, *background=None*, *sigma=None*, *stationary_line_core=None*, *output=False*, ***kwargs*)
Plots the data and fitted parameters

**Parameters**

- **fit** (FitResult *or list or array_like, optional, default = None*) – The fitted parameters to plot with the data. Can extract the necessary plot metadata from the fit object. Otherwise, *fit* should be the parameters to be fitted to either a Voigt or double Voigt profile depending on the number of parameters fitted.
- **time** (*int or iterable, optional, default = None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default = None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default = None*) – The column index. See comment for *time* parameter.
- **spectrum** (ndarray of length *original_wavelengths*, ndim=1, optional, default = None) – The explicit spectrum to plot along with a fit (if specified).

- **classification** (*int, optional, default = None*) – Used to determine which sigma profile to use. See *_get_sigma* for more details.
- **background** (float or array_like of length *constant_wavelengths*, optional, default = see note) – Background to added to the fitted profiles. If a *spectrum* is given, this will default to zero, otherwise the value loaded by *load_background* will be used.
- **sigma** (*int or array_like, optional, default = None*) – Explicit sigma index or profile. See *_get_sigma* for details.
- **stationary_line_core** (float, optional, default = *self.stationary_line_core*) – The stationary line core wavelength to mark on the plot.
- **output** (*bool or str, optional, default = False*) – Whether to save the plot to a file. If true, a file of format *plot_<time>_<row>_<column>.eps* will be created in the current directory. If a string, that will be used as the filename. (Can change filetype like this.) If false, no file will be created.
- ****kwargs** – Parameters used by matplotlib and *separate* (see *plot_separate*) and *subtraction* (see *plot_subtraction*).
  - *figsize* passed to *matplotlib.pyplot.figure*
  - *legend_position* passed to *matplotlib.pyplot.legend*
  - *dpi* passed to *matplotlib.pyplot.figure* and *matplotlib.pyplot.savefig*
  - *fontfamily* passed to *matplotlib.pyplot.rc('font', family=`fontfamily`)* if given

**See also:**

***plot_separate()*** Plot the fit parameters separately

***plot_subtraction()*** Plot the spectrum with the emission fit subtracted from it

***FitResult.plot()*** Plotting method on the fit result

**plot_separate**(**args*, ***kwargs*)

Plot the fitted profiles separately

If multiple profiles exist, fit them separately. See *plot* for more details.

**See also:**

***plot()*** General plotting method

***plot_subtraction()*** Plot the spectrum with the emission fit subtracted from it

***FitResult.plot()*** Plotting method on the fit result

**plot_subtraction**(**args*, ***kwargs*)

Plot the spectrum with the emission fit subtracted from it

If multiple profiles exist, subtract the fitted emission from the raw data. See *plot* for more details.

**See also:**

***plot()*** General plotting method

***plot_separate()*** Plot the fit parameters separately

***FitResult.plot()*** Plotting method on the fit result

### ModelBase

**class** mcalf.models.**ModelBase**

Bases: object

Base class for spectral line model fitting.

Warning: This class should not be used directly. Use derived classes instead.

**array**
Array holding spectra.

**Type** ndarray, dimensions are ['time', 'row', 'column', 'spectra']

**background**
Array holding spectral backgrounds.

**Type** ndarray, dimensions are ['time', 'row', 'column']

#### Methods Summary

| | |
|---|---|
| *fit_spectrum*(spectrum, **kwargs) | Fits the specified spectrum array |
| *get_spectra*([time, row, column, spectrum, ...]) | Gets corrected spectra from the spectral array |
| *load_array*(array[, names]) | Load an array of spectra. |
| *load_background*(array[, names]) | Load an array of spectral backgrounds. |
| *test*(X, y) | Test the accuracy of the trained neural network. |
| *train*(X, y) | Fit the neural network model to spectra matrix X and spectra labels y. |

#### Methods Documentation

**fit_spectrum**(*spectrum*, ***kwargs*)
Fits the specified spectrum array

Passes the spectrum argument to the fit method. For easily iterating over a list of spectra.

**Parameters**

- **spectrum** (*ndarray of ndim=1*) – The explicit spectrum.
- ****kwargs** (*dictionary, optional*) – Extra keyword arguments to pass to fit.

**Returns** **result** – Result of the fit.

**Return type** *FitResult*

**See also:**

**fit()** General fitting method

**get_spectra**(*time=None*, *row=None*, *column=None*, *spectrum=None*, *correct=True*, *background=False*)
Gets corrected spectra from the spectral array

Takes either a set of indices or an explicit spectrum and optionally applied corrections and background removal.

**Parameters**

- **time** (*int or iterable, optional, default=None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default=None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default=None*) – The column index. See comment for *time* parameter.
- **spectrum** (*ndarray of ndim=1, optional, default=None*) – The explicit spectrum.
- **correct** (*bool, optional, default=True*) – Whether to reinterpolate the spectrum and apply the prefilter correction (if exists).
- **background** (*bool, optional, default=False*) – Whether to include the background in the outputted spectra. Only removes the background if the relevant background array has been loaded. Does not remove background is processing an explicit spectrum.

**load_array**(*array*, *names=None*)

Load an array of spectra.

Load *array* with dimension names *names* into the *array* parameter of the model object.

**Parameters**

- **array** (*ndarray of ndims > 1*) – An array containing at least two spectra.
- **names** (list of str, length = *array.ndims*) – List of dimension names for *array*. Valid dimension names are 'time', 'row', 'column' and 'wavelength'. 'wavelength' is a required dimension.

**See also:**

***load_background()*** Load an array of spectral backgrounds

**load_background**(*array*, *names=None*)

Load an array of spectral backgrounds.

Load *array* with dimension names *names* into *background* parameter of the model object.

**Parameters**

- **array** (*ndarray of ndim>0*) – An array containing at least two backgrounds.
- **names** (list of str, length = *array.ndims*) – List of dimension names for *array*. Valid dimension names are 'time', 'row' and 'column'.

**See also:**

***load_array()*** Load and array of spectra

**test**(*X*, *y*)

Test the accuracy of the trained neural network.

**Prints a table of results showing:**

1) the percentage of predictions that equal the target labels;
2) **the average classification deviation and standard deviation from the ground truth classification** for each labelled classification;

3) the average classification deviation and standard deviation overall.

If the model object has an output parameter, it will create a CSV file (*self.output*/neural_network/test.csv) listing the predictions and ground truth data.

**Parameters**

- **X** (*ndarray or sparse matrix of shape (n_spectra, n_wavelengths)*) – The input spectra.
- **y** (*ndarray of shape (n_spectra,) or (n_spectra, n_outputs)*) – The target class labels.

**See also:**

***train()*** Train the neural network

**train**(*X*, *y*)

Fit the neural network model to spectra matrix X and spectra labels y.

Calls the *fit* method on the *neural_network* parameter of the model object.

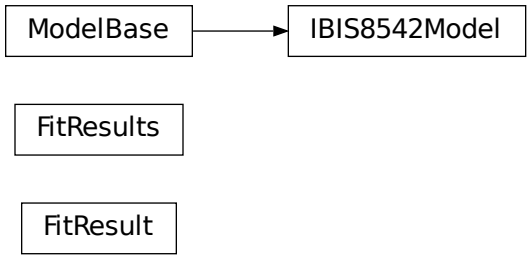
**Parameters**

- **X** (*ndarray or sparse matrix of shape (n_spectra, n_wavelengths)*) – The input spectra.
- **y** (*ndarray of shape (n_spectra,) or (n_spectra, n_outputs)*) – The target class labels.

**See also:**

***test()*** Test how well the neural network has been trained

### Class Inheritance Diagram



## mcalf.models.base Module

### Classes

| | |
|---|---|
| *ModelBase*() | Base class for spectral line model fitting. |

### ModelBase

**class** mcalf.models.base.**ModelBase**

Bases: object

Base class for spectral line model fitting.

Warning: This class should not be used directly. Use derived classes instead.

**array**

Array holding spectra.

**Type** ndarray, dimensions are ['time', 'row', 'column', 'spectra']

**background**

Array holding spectral backgrounds.

**Type** ndarray, dimensions are ['time', 'row', 'column']

#### Methods Summary

| | |
|---|---|
| *fit_spectrum*(spectrum, **kwargs) | Fits the specified spectrum array |
| *get_spectra*([time, row, column, spectrum, ...]) | Gets corrected spectra from the spectral array |
| *load_array*(array[, names]) | Load an array of spectra. |
| *load_background*(array[, names]) | Load an array of spectral backgrounds. |
| *test*(X, y) | Test the accuracy of the trained neural network. |
| *train*(X, y) | Fit the neural network model to spectra matrix X and spectra labels y. |

#### Methods Documentation

**fit_spectrum**(*spectrum*, ***kwargs*)

Fits the specified spectrum array

Passes the spectrum argument to the fit method. For easily iterating over a list of spectra.

**Parameters**

- **spectrum** (*ndarray of ndim=1*) – The explicit spectrum.
- ****kwargs** (*dictionary, optional*) – Extra keyword arguments to pass to fit.

**Returns** **result** – Result of the fit.

**Return type** *FitResult*

**See also:**

**fit()** General fitting method

**get_spectra**(*time=None*, *row=None*, *column=None*, *spectrum=None*, *correct=True*, *background=False*)

Gets corrected spectra from the spectral array

Takes either a set of indices or an explicit spectrum and optionally applied corrections and background removal.

**Parameters**

- **time** (*int or iterable, optional, default=None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default=None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default=None*) – The column index. See comment for *time* parameter.
- **spectrum** (*ndarray of ndim=1, optional, default=None*) – The explicit spectrum.
- **correct** (*bool, optional, default=True*) – Whether to reinterpolate the spectrum and apply the prefilter correction (if exists).
- **background** (*bool, optional, default=False*) – Whether to include the background in the outputted spectra. Only removes the background if the relevant background array has been loaded. Does not remove background is processing an explicit spectrum.

**load_array**(*array*, *names=None*)

Load an array of spectra.

Load *array* with dimension names *names* into the *array* parameter of the model object.

**Parameters**

- **array** (*ndarray of ndims > 1*) – An array containing at least two spectra.
- **names** (list of str, length = *array.ndims*) – List of dimension names for *array*. Valid dimension names are 'time', 'row', 'column' and 'wavelength'. 'wavelength' is a required dimension.

**See also:**

***load_background()*** Load an array of spectral backgrounds

**load_background**(*array*, *names=None*)

Load an array of spectral backgrounds.

Load *array* with dimension names *names* into *background* parameter of the model object.

**Parameters**

- **array** (*ndarray of ndim>0*) – An array containing at least two backgrounds.
- **names** (list of str, length = *array.ndims*) – List of dimension names for *array*. Valid dimension names are 'time', 'row' and 'column'.

**See also:**

***load_array()*** Load and array of spectra

**test** (*X*, *y*)

Test the accuracy of the trained neural network.

**Prints a table of results showing:**

1) the percentage of predictions that equal the target labels;
2) **the average classification deviation and standard deviation from the ground truth classification** for each labelled classification;
3) the average classification deviation and standard deviation overall.

If the model object has an output parameter, it will create a CSV file (*self.output*/neural_network/test.csv) listing the predictions and ground truth data.

**Parameters**

- **X** (*ndarray or sparse matrix of shape (n_spectra, n_wavelengths)*) – The input spectra.
- **y** (*ndarray of shape (n_spectra,) or (n_spectra, n_outputs)*) – The target class labels.

**See also:**

***train()*** Train the neural network

**train** (*X*, *y*)

Fit the neural network model to spectra matrix X and spectra labels y.

Calls the *fit* method on the *neural_network* parameter of the model object.

**Parameters**

- **X** (*ndarray or sparse matrix of shape (n_spectra, n_wavelengths)*) – The input spectra.
- **y** (*ndarray of shape (n_spectra,) or (n_spectra, n_outputs)*) – The target class labels.

**See also:**

***test()*** Test how well the neural network has been trained

### Class Inheritance Diagram

### mcalf.models.ibis Module

### Classes

| | |
|---|---|
| *IBIS8542Model*([stationary_line_core, ...]) | Class for working with IBIS 8542 Å calcium II spectral imaging observations |

### IBIS8542Model

**class** mcalf.models.ibis.**IBIS8542Model**(*stationary_line_core=None*, *absorption_guess=None*, *emission_guess=None*, *absorption_min_bound=None*, *emission_min_bound=None*, *absorption_max_bound=None*, *emission_max_bound=None*, *absorption_x_scale=None*, *emission_x_scale=None*, *neural_network=None*, *original_wavelengths=None*, *constant_wavelengths=None*, *delta_lambda=None*, *sigma=None*, *prefilter_response=None*, *prefilter_ref_main=None*, *prefilter_ref_wvscl=None*, *config=None*, *output=None*)

Bases: *mcalf.models.base.ModelBase*

Class for working with IBIS 8542 Å calcium II spectral imaging observations

**Parameters**

- **original_wavelengths** (*array_like*) – One-dimensional array of wavelengths that correspond to the uncorrected spectral data.
- **stationary_line_core** (*float, optional, default = 8542.104320687517*) – Wavelength of the stationary line core.
- **absorption_guess** (*array_like, length=4, optional, default = [-1000, stationary_line_core, 0.2, 0.1]*) – Initial guess to take when fitting the absorption Voigt profile.
- **emission_guess** (*array_like, length=4, optional, default = [1000, stationary_line_core, 0.2, 0.1]*) – Initial guess to take when fitting the emission Voigt profile.
- **absorption_min_bound** (*array_like, length=4, optional, default = [-np.inf, stationary_line_core-0.15, 1e-6, 1e-6]*) – Minimum bounds for all the absorption Voigt profile parameters in order of the function's arguments.
- **emission_min_bound** (*array_like, length=4, optional, default = [0, stationary_line_core-0.15, 1e-6, 1e-6]*) – Minimum bounds for all the emission Voigt profile parameters in order of the function's arguments.
- **absorption_max_bound** (*array_like, length=4, optional, default = [0, stationary_line_core+0.15, 1, 1]*) – Maximum bounds for all the absorption Voigt profile parameters in order of the function's arguments.
- **emission_max_bound** (*array_like, length=4, optional, default = [np.inf, stationary_line_core+0.15, 1, 1]*) – Maximum bounds for all the emission Voigt profile parameters in order of the function's arguments.

- **absorption_x_scale** (*array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]*) – Characteristic scale for all the absorption Voigt profile parameters in order of the function's arguments.
- **emission_x_scale** (*array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]*) – Characteristic scale for all the emission Voigt profile parameters in order of the function's arguments.
- **neural_network** (*sklearn.neural_network.MLPClassifier, optional, default = see description*) – The MLPClassifier object that will be used to classify the spectra. Its default value is *MLPClassifier(solver='lbfgs', alpha=1e-3, hidden_layer_sizes=(10, 4), random_state=1)*.
- **constant_wavelengths** (array_like, length same as *original_wavelengths*, optional, default = see description) – The desired set of wavelengths that the spectral data should be rescaled to represent. It is assumed that these have constant spacing, but that may not be a requirement if you specify your own array. The default value is an array from the minimum to the maximum wavelength of *original_wavelengths* in constant steps of *delta_lambda*, overshooting the upper bound if the maximum wavelength has not been reached.
- **delta_lambda** (*float, optional, default = 0.05*) – The step used between each value of *constant_wavelengths* when its default value has to be calculated.
- **sigma** (*list of array_like or bool, length=(2, n_wavelengths), optional, default = [type1, type2]*) – A list of different sigma that are used to weight particular wavelengths along the spectra when fitting. The fitting method will expect to be able to choose a sigma array from this list at a specific index. It's default value is *[generate_sigma(i, constant_wavelengths, stationary_line_core) for i in [1, 2]]*. See *utils.generate_sigma()* for more information. If bool, True will generate the default sigma value regardless of the value specified in *config*, and False will set *sigma* to be all ones, effectively disabling it.
- **prefilter_response** (*array_like, length=n_wavelengths, optional, default = see note*) – Each constant wavelength scaled spectrum will be corrected by dividing it by this array. If *prefilter_response* is not give, and *prefilter_ref_main* and *prefilter_ref_wvscl* are not given, *prefilter_response* will have a default value of *None*.
- **prefilter_ref_main** (*array_like, optional, default = None*) – If *prefilter_response* is not specified, this will be used along with *prefilter_ref_wvscl* to generate the default value of *prefilter_response*.
- **prefilter_ref_wvscl** (*array_like, optional, default = None*) – If *prefilter_response* is not specified, this will be used along with *prefilter_ref_main* to generate the default value of *prefilter_response*.
- **config** (*str, optional, default = None*) – Filename of a *.yml* file (relative to current directory) containing the initialising parameters for this object. Parameters provided explicitly to the object upon initialisation will override any provided in this file. All (or some) parameters that this object accepts can be specified in this file, except *neural_network* and *config*. Each line of the file should specify a different parameter and be formatted like *emission_guess: '[-inf, wl-0.15, 1e-6, 1e-6]'* or *original_wavelengths: 'original.fits'* for example. When specifying a string, use 'inf' to represent *np.inf* and 'wl' to represent *stationary_line_core* as shown. If the string matches a file, *utils.load_parameter()* is used to load the contents of the file.
- **output** (*str, optional, default = None*) – If the program wants to output data, it will place it relative to the location specified by this parameter. Some methods

will only save data to a file if this parameter is not *None*. Such cases will be documented where relevant.

**original_wavelengths**
One-dimensional array of wavelengths that correspond to the uncorrected spectral data.

**Type** array_like

**stationary_line_core**
Wavelength of the stationary line core.

**Type** float, optional, default = 8542.099145376844

**absorption_guess**
Initial guess to take when fitting the absorption Voigt profile.

**Type** array_like, length=4, optional, default = [-1000, stationary_line_core, 0.2, 0.1]

**emission_guess**
Initial guess to take when fitting the emission Voigt profile.

**Type** array_like, length=4, optional, default = [1000, stationary_line_core, 0.2, 0.1]

**absorption_min_bound**
Minimum bounds for all the absorption Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [-np.inf, stationary_line_core-0.15, 1e-6, 1e-6]

**emission_min_bound**
Minimum bounds for all the emission Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [0, -np.inf, 1e-6, 1e-6]

**absorption_max_bound**
Maximum bounds for all the absorption Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [0, stationary_line_core+0.15, 1, 1]

**emission_max_bound**
Maximum bounds for all the emission Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [np.inf, np.inf, 1, 1]

**absorption_x_scale**
Characteristic scale for all the absorption Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]

**emission_x_scale**
Characteristic scale for all the emission Voigt profile parameters in order of the function's arguments.

**Type** array_like, length=4, optional, default = [1500, 0.2, 0.3, 0.5]

**neural_network**
The MLPClassifier object (or similar) that will be used to classify the spectra. Defaults to a *GridSearchCV* with *MLPClassifier(solver='lbfgs', hidden_layer_sizes=(40,), max_iter=1000)* for best *alpha* selected from *[1e-5, 2e-5, 3e-5, 4e-5, 5e-5, 6e-5, 7e-5, 8e-5, 9e-5]*.

**Type** sklearn.neural_network.MLPClassifier, optional, default = see description

**constant_wavelengths**
The desired set of wavelengths that the spectral data should be rescaled to represent. It is assumed that these have constant spacing, but that may not be a requirement if you specify your own array. The default value is an array from the minimum to the maximum wavelength of *original_wavelengths* in constant steps of *delta_lambda*, overshooting the upper bound if the maximum wavelength has not been reached.

**Type** array_like, length same as *original_wavelengths*, optional, default = see description

**sigma**
A list of different sigma that are used to weight particular wavelengths along the spectra when fitting. The fitting method will expect to be able to choose a sigma array from this list at a specific index. It's default value is *[generate_sigma(i, constant_wavelengths, stationary_line_core) for i in [1, 2]]*. See *utils.generate_sigma()* for more information.

**Type** list of array_like, length=(2, n_wavelengths), optional, default = [type1, type2]

**prefilter_response**
Each constant wavelength scaled spectrum will be corrected by dividing it by this array. If *prefilter_response* is not give, and *prefilter_ref_main* and *prefilter_ref_wvscl* are not given, *prefilter_response* will have a default value of *None*.

**Type** array_like, length=n_wavelengths, optional, default = see note

**output**
If the program wants to output data, it will place it relative to the location specified by this parameter. Some methods will only save data to a file if this parameter is not *None*. Such cases will be documented where relevant.

**Type** str, optional, default = None

**quiescent_wavelength**
The index within the fitted parameters of the absorption Voigt line core wavelength.

**Type** int, default = 1

**active_wavelength**
The index within the fitted parameters of the emission Voigt line core wavelength.

**Type** int, default = 5

### Methods Summary

| | |
|---|---|
| *classify_spectra*([time, row, column, ...]) | Classify the specified spectra |
| *fit*([time, row, column, spectrum, profile, ...]) | Fits the model to specified spectra |
| *plot*([fit, time, row, column, spectrum, ...]) | Plots the data and fitted parameters |
| *plot_separate*(*args, **kwargs) | Plot the fitted profiles separately |
| *plot_subtraction*(*args, **kwargs) | Plot the spectrum with the emission fit subtracted from it |

### Methods Documentation

**classify_spectra**(*time=None*, *row=None*, *column=None*, *spectra=None*, *only_normalise=False*)
Classify the specified spectra

Will also normalise each spectrum such that its intensity will range from zero to one.

**Parameters**

- **time** (*int or iterable, optional, default=None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default=None*) – The row index. See comment for *time* parameter.

- **column** (*int or iterable, optional, default=None*) – The column index. See comment for *time* parameter.
- **spectra** (*ndarray, optional, default=None*) – The explicit spectra to classify. If *only_normalise* is False, this must be 1D.
- **only_normalise** (*bool, optional, default = False*) – Whether the single spectrum given in *spectra* should not be interpolated and corrected.

**Returns classifications** – Array of classifications with the same time, row and column indices as *spectra*.

**Return type** ndarray

**See also:**

**train()** Train the neural network

**test()** Test the accuracy of the neural network

**get_spectra()** Get processed spectra from the objects *array* attribute

**fit**(*time=None*, *row=None*, *column=None*, *spectrum=None*, *profile=None*, *sigma=None*, *classifications=None*, *background=None*, *n_pools=None*)
Fits the model to specified spectra

Fits the model to an array of spectra using multiprocessing if requested.

**Parameters**

- **time** (*int or iterable, optional, default=None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default=None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default=None*) – The column index. See comment for *time* parameter.
- **spectrum** (*ndarray, ndim=1, optional, default=None*) – The explicit spectrum to fit the model to.
- **profile** (*str, optional, default = None*) – The profile to fit. (Will infer profile from *classifications* if omitted.)
- **sigma** (*int or array_like, optional, default = None*) – Explicit sigma index or profile. See *_get_sigma* for details.
- **classifications** (*int, optional, default = None*) – Classifications to determine the fitted profile to use (if profile not explicitly given). Will use neural network to classify them if not.
- **background** (*float, optional, default = None*) – If provided, this value will be subtracted from the explicit spectrum provided in *spectrum*. Will not be applied to spectra found from the indices, use the *load_background* method instead.
- **n_pools** (*int, optional, default = None*) – The number of processing pools to calculate the fitting over. This allocates the fitting of different spectra to *n_pools* separate worker processes. When processing a large number of spectra this will make the fitting process take less time overall. It also distributes such that each worker process has the same ratio of classifications to process. This should balance out the workload between

workers. If few spectra are being fitted, performance may decrease due to the overhead associated with splitting the evaluation over separate processes. If *n_pools* is not an integer greater than zero, it will fit the spectrum with a for loop.

**Returns** **result** – Outcome of the fits returned as a list of FitResult objects

**Return type** list of FitResult, length=n_spectra

**plot**(*fit=None*, *time=None*, *row=None*, *column=None*, *spectrum=None*, *classification=None*, *background=None*, *sigma=None*, *stationary_line_core=None*, *output=False*, ***kwargs*)

Plots the data and fitted parameters

**Parameters**

- **fit** (*FitResult or list or array_like, optional, default = None*) – The fitted parameters to plot with the data. Can extract the necessary plot metadata from the fit object. Otherwise, *fit* should be the parameters to be fitted to either a Voigt or double Voigt profile depending on the number of parameters fitted.
- **time** (*int or iterable, optional, default = None*) – The time index. The index can be either a single integer index or an iterable. E.g. a list, a NumPy array, a Python range, etc. can be used.
- **row** (*int or iterable, optional, default = None*) – The row index. See comment for *time* parameter.
- **column** (*int or iterable, optional, default = None*) – The column index. See comment for *time* parameter.
- **spectrum** (ndarray of length *original_wavelengths*, ndim=1, optional, default = None) – The explicit spectrum to plot along with a fit (if specified).
- **classification** (*int, optional, default = None*) – Used to determine which sigma profile to use. See *_get_sigma* for more details.
- **background** (float or array_like of length *constant_wavelengths*, optional, default = see note) – Background to added to the fitted profiles. If a *spectrum* is given, this will default to zero, otherwise the value loaded by *load_background* will be used.
- **sigma** (*int or array_like, optional, default = None*) – Explicit sigma index or profile. See *_get_sigma* for details.
- **stationary_line_core** (float, optional, default = *self.stationary_line_core*) – The stationary line core wavelength to mark on the plot.
- **output** (*bool or str, optional, default = False*) – Whether to save the plot to a file. If true, a file of format *plot_<time>_<row>_<column>.eps* will be created in the current directory. If a string, that will be used as the filename. (Can change filetype like this.) If false, no file will be created.
- ****kwargs** – Parameters used by matplotlib and *separate* (see *plot_separate*) and *subtraction* (see *plot_subtraction*).
  - *figsize* passed to *matplotlib.pyplot.figure*
  - *legend_position* passed to *matplotlib.pyplot.legend*
  - *dpi* passed to *matplotlib.pyplot.figure* and *matplotlib.pyplot.savefig*
  - *fontfamily* passed to *matplotlib.pyplot.rc('font', family=`fontfamily`)* if given

**See also:**

***plot_separate()*** Plot the fit parameters separately

*plot_subtraction()* Plot the spectrum with the emission fit subtracted from it

**FitResult.plot()** Plotting method on the fit result

**plot_separate**(**args*, ***kwargs*)
Plot the fitted profiles separately

If multiple profiles exist, fit them separately. See *plot* for more details.

**See also:**

*plot()* General plotting method

*plot_subtraction()* Plot the spectrum with the emission fit subtracted from it

**FitResult.plot()** Plotting method on the fit result

**plot_subtraction**(**args*, ***kwargs*)
Plot the spectrum with the emission fit subtracted from it

If multiple profiles exist, subtract the fitted emission from the raw data. See *plot* for more details.

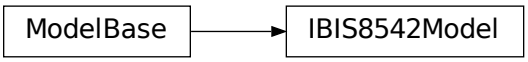
**See also:**

*plot()* General plotting method

*plot_separate()* Plot the fit parameters separately

**FitResult.plot()** Plotting method on the fit result

### Class Inheritance Diagram




## mcalf.models.results Module

### Classes

| | |
|---|---|
| *FitResult*(fitted_parameters, fit_info) | Class that holds the result of a fit |
| *FitResults*(shape, n_parameters[, time]) | Class that holds multiple fit results in a way that can be easily processed |

### FitResult

**class** mcalf.models.results.**FitResult**(*fitted_parameters*, *fit_info*)

Bases: object

Class that holds the result of a fit

**Parameters**

- **fitted_parameters** (*ndarray*) – The parameters fitted.
- **fit_info** (*dict*) – Additional information on the fit including at least 'classification', 'profile', 'success' and 'index'.

**parameters**
The parameters fitted.

**Type** ndarray

**classification**
Classification of the fitted spectrum.

**Type** int

**profile**
Profile of the fitted spectrum.

**Type** str

**success**
Whether the fit was completed successfully.

**Type** bool

**index**
Index ([<time>, <row>, <column>]) of the spectrum in the spectral array.

**Type** list

**__dict__**
Other attributes may be present depending on the *fit_info* used.

#### Methods Summary

| | |
|---|---|
| *plot*(model, **kwargs) | Plot the data and fitted parameters |
| *velocity*(model[, vtype]) | Calculate the Doppler velocity of the fit using *model* parameters |

#### Methods Documentation

**plot**(*model*, ***kwargs*)
Plot the data and fitted parameters

This calls the *plot* method on *model* but will plot for this FitResult object. See the model's *plot* method for more details.

**Parameters**

- **model** (*child class of ModelBase*) – The model object to plot with.

- ****kwargs** – See the *model.plot* method for more details.

**velocity**(*model*, *vtype='quiescent'*)
Calculate the Doppler velocity of the fit using *model* parameters

**Parameters**

- **model** (*child class of ModelBase*) – The model object to take parameters from.
- **vtype** (*{'quiescent', 'active'}, default='quiescent'*) – The velocity type to find.

**Returns** **velocity** – The calculated velocity.

**Return type** float

### FitResults

**class** mcalf.models.results.**FitResults**(*shape*, *n_parameters*, *time=None*)
Bases: object

Class that holds multiple fit results in a way that can be easily processed

**parameters**
Array of fitted parameters.

**Type** ndarray of shape (*row*, *column*, *parameter*)

**classifications**
Array of classifications.

**Type** ndarray of int of shape (*row*, *column*)

**profile**
Array of profiles.

**Type** ndarray of str of shape (*row*, *column*)

**success**
Array of success statuses.

**Type** ndarray of bool of shape (*row*, *column*)

**chi2**
Array of chi-squared values.

**Type** ndarray of shape (*row*, *column*)

**time**
Time index that the *FitResult* object refers to (if provided).

**Type** int, default = None

**n_parameters**
Number of parameters in the last dimension of *parameters*.

**Type** int

### Methods Summary

| | |
|---|---|
| *append*(result) | Append a *FitResult* object to the *FitResults* object |
| *save*(filename[, model]) | Saves the FitResults object to a FITS file |
| *velocities*(model[, row, column, vtype]) | Calculate the Doppler velocities of the fit results using *model* parameters |

### Methods Documentation

**append** (*result*)
Append a *FitResult* object to the *FitResults* object

**Parameters** **result** (*FitResult object*) – *FitResult* object to append.

**save** (*filename*, *model=None*)
Saves the FitResults object to a FITS file

**Parameters**

- **filename** (*file path, file object or file-like object*) – FITS file to write to. If a file object, must be opened in a writeable mode.
- **model** (*child class of mcalf.models.base.ModelBase, optional, default = None*) – If provided, use this model to calculate and include both quiescent and active Doppler velocities.

#### Notes

Saves a FITS file to the location specified by *filename*. All the parameters are stored in a separate, named, HDU.

**velocities** (*model*, *row=None*, *column=None*, *vtype='quiescent'*)
Calculate the Doppler velocities of the fit results using *model* parameters

**Parameters**

- **model** (*child class of ModelBase*) – The model object to take parameters from.
- **row** (*int, list, array_like, iterable, optional, default = None*) – The row indices to find velocities for. All if omitted.
- **column** (*int, list, array_like, iterable, optional, default = None*) – The column indices to find velocities for. All if omitted.
- **vtype** (*{'quiescent', 'active'}, default='quiescent'*) – The velocity type to find.

**Returns** **velocities** – The calculated velocities for the specified *row* and *column* positions.

**Return type** ndarray of shape (*row*, *column*)

### 1.2.3 MCALF profiles

This sub-package contains:

- Functions that can be used to model the spectra.
- Voigt profile with a variety of wrappers for different applications (*mcalf.profiles.voigt*).
- Gaussian profiles and skew normal distributions (*mcalf.profiles.gaussian*).

#### mcalf.profiles Package

#### Functions

| | |
|---|---|
| *double_voigt*(x, a1, b1, s1, g1, a2, b2, s2, ...) | Double Voigt function with background |
| *double_voigt_approx*(x, a1, b1, s1, g1, a2, ...) | Double Voigt function (efficient approximation) with background |
| *double_voigt_approx_nobg*(x, a1, b1, s1, g1, ...) | Double Voigt function (efficient approximation) with no background |
| *double_voigt_nobg*(x, a1, b1, s1, g1, a2, b2, ...) | Double Voigt function with no background |
| *single_gaussian*(x, a, b, c, d) | Gaussian function |
| *skew_normal*(x, a_a, alpha, xi, omega, d) | |
| *skew_normal_with_gaussian*(x, a_a, alpha, xi, ...) | |
| *voigt*(x, a, b, s, g, d[, clib]) | Voigt function with background |
| *voigt_approx*(x, a, b, s, g, d) | Voigt function (efficient approximation) with background |
| *voigt_approx_nobg*(x, a, b, s, g) | Voigt function (efficient approximation) with no background (Base approx. |
| *voigt_nobg*(x, a, b, s, g[, clib]) | Voigt function with no background (Base Voigt function) |

#### double_voigt

mcalf.profiles.**double_voigt**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*, *d*, *clib=True*)

Double Voigt function with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.
- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.

- **d** (*float*) – Background.
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

## double_voigt_approx

mcalf.profiles.**double_voigt_approx**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*, *d*)

Double Voigt function (efficient approximation) with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of the Lorentzian of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.
- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.
- **d** (*float*) – Background.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx_nobg()*** Base approximated Voigt function with no background

***voigt_approx()*** Approximated Voigt function with background added

***double_voigt_approx_nobg()*** Two approximated Voigt functions added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### double_voigt_approx_nobg

mcalf.profiles.**double_voigt_approx_nobg**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*)
Double Voigt function (efficient approximation) with no background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of the Lorentzian of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.
- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx_nobg()*** Base approximated Voigt function with no background

***voigt_approx()*** Approximated Voigt function with background added

***double_voigt_approx()*** Two approximated Voigt functions and a background added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### double_voigt_nobg

mcalf.profiles.**double_voigt_nobg**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*, *clib=True*)
Double Voigt function with no background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.

- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt()*** Two Voigt function and a background added together

### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

## single_gaussian

mcalf.profiles.**single_gaussian**(*x*, *a*, *b*, *c*, *d*)

Gaussian function

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Gaussian function at.
- **a** (*float*) – Amplitude.
- **b** (*float*) – Central line core.
- **c** (*float*) – Sigma of Gaussian.
- **d** (*float*) – Background to add.

## skew_normal

mcalf.profiles.**skew_normal**(*x*, *a_a*, *alpha*, *xi*, *omega*, *d*)

### skew_normal_with_gaussian

mcalf.profiles.**skew_normal_with_gaussian**(*x*, *a_a*, *alpha*, *xi*, *omega*, *a_e*, *b*, *c*, *d*)

### voigt

mcalf.profiles.**voigt**(*x*, *a*, *b*, *s*, *g*, *d*, *clib=True*)

Voigt function with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).
- **d** (*float*) – Background.
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_nobg()*** Base Voigt function with no background

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

#### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

### voigt_approx

mcalf.profiles.**voigt_approx**(*x*, *a*, *b*, *s*, *g*, *d*)

Voigt function (efficient approximation) with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude of the Lorentzian.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).

- **g** (*float*) – Gamma (for Lorentzian).
- **d** (*float*) – Background.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx_nobg()*** Base approximated Voigt function with no background

***double_voigt_approx_nobg()*** Two approximated Voigt functions added together

***double_voigt_approx()*** Two approximated Voigt functions and a background added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### voigt_approx_nobg

mcalf.profiles.**voigt_approx_nobg**(*x*, *a*, *b*, *s*, *g*)

Voigt function (efficient approximation) with no background (Base approx. Voigt function)

This is the base for all other approximated Voigt functions. Not implemented in any models yet as initial tests exhibited slow convergence.

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude of the Lorentzian.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx()*** Approximated Voigt function with background added

***double_voigt_approx_nobg()*** Two approximated Voigt functions added together

***double_voigt_approx()*** Two approximated Voigt functions and a background added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### Notes

This algorithm is taken from A. B. McLean et al.[1].

### References

## voigt_nobg

mcalf.profiles.**voigt_nobg**(*x*, *a*, *b*, *s*, *g*, *clib=True*)

Voigt function with no background (Base Voigt function)

This is the base of all the other Voigt functions.

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

[1] A. B. McLean, C. E. J. Mitchell and D. M. Swanston, "Implementation of an efficient analytical approximation to the Voigt function for photoemission lineshape analysis," Journal of Electron Spectroscopy and Related Phenomena, vol. 69, pp. 125-132, 1994. https://doi.org/10.1016/0368-2048(94)02189-7

### mcalf.profiles.voigt Module

### Functions

| | |
|---|---|
| *voigt_approx_nobg*(x, a, b, s, g) | Voigt function (efficient approximation) with no background (Base approx. |
| *voigt_approx*(x, a, b, s, g, d) | Voigt function (efficient approximation) with background |
| *double_voigt_approx_nobg*(x, a1, b1, s1, g1, ...) | Double Voigt function (efficient approximation) with no background |
| *double_voigt_approx*(x, a1, b1, s1, g1, a2, ...) | Double Voigt function (efficient approximation) with background |
| *voigt_nobg*(x, a, b, s, g[, clib]) | Voigt function with no background (Base Voigt function) |
| *voigt*(x, a, b, s, g, d[, clib]) | Voigt function with background |
| *double_voigt_nobg*(x, a1, b1, s1, g1, a2, b2, ...) | Double Voigt function with no background |
| *double_voigt*(x, a1, b1, s1, g1, a2, b2, s2, ...) | Double Voigt function with background |

### voigt_approx_nobg

mcalf.profiles.voigt.**voigt_approx_nobg**(*x*, *a*, *b*, *s*, *g*)

Voigt function (efficient approximation) with no background (Base approx. Voigt function)

This is the base for all other approximated Voigt functions. Not implemented in any models yet as initial tests exhibited slow convergence.

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude of the Lorentzian.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx()*** Approximated Voigt function with background added

***double_voigt_approx_nobg()*** Two approximated Voigt functions added together

***double_voigt_approx()*** Two approximated Voigt functions and a background added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### Notes

This algorithm is taken from A. B. McLean et al.[1].

### References

## voigt_approx

mcalf.profiles.voigt.**voigt_approx**(*x*, *a*, *b*, *s*, *g*, *d*)

Voigt function (efficient approximation) with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude of the Lorentzian.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).
- **d** (*float*) – Background.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx_nobg()*** Base approximated Voigt function with no background

***double_voigt_approx_nobg()*** Two approximated Voigt functions added together

***double_voigt_approx()*** Two approximated Voigt functions and a background added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

## double_voigt_approx_nobg

mcalf.profiles.voigt.**double_voigt_approx_nobg**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*)

Double Voigt function (efficient approximation) with no background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of the Lorentzian of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.

[1] A. B. McLean, C. E. J. Mitchell and D. M. Swanston, "Implementation of an efficient analytical approximation to the Voigt function for photoemission lineshape analysis," Journal of Electron Spectroscopy and Related Phenomena, vol. 69, pp. 125-132, 1994. https://doi.org/10.1016/0368-2048(94)02189-7

- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx_nobg()*** Base approximated Voigt function with no background

***voigt_approx()*** Approximated Voigt function with background added

***double_voigt_approx()*** Two approximated Voigt functions and a background added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### double_voigt_approx

mcalf.profiles.voigt.**double_voigt_approx**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*, *d*)

Double Voigt function (efficient approximation) with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of the Lorentzian of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.
- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.
- **d** (*float*) – Background.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_approx_nobg()*** Base approximated Voigt function with no background

***voigt_approx()*** Approximated Voigt function with background added

***double_voigt_approx_nobg()*** Two approximated Voigt functions added together

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

### voigt_nobg

mcalf.profiles.voigt.**voigt_nobg**(*x*, *a*, *b*, *s*, *g*, *clib=True*)

Voigt function with no background (Base Voigt function)

This is the base of all the other Voigt functions.

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

#### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

### voigt

mcalf.profiles.voigt.**voigt**(*x*, *a*, *b*, *s*, *g*, *d*, *clib=True*)

Voigt function with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a** (*float*) – Amplitude.
- **b** (*float*) – Central line core.
- **s** (*float*) – Sigma (for Gaussian).
- **g** (*float*) – Gamma (for Lorentzian).
- **d** (*float*) – Background.
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_nobg()*** Base Voigt function with no background

***double_voigt_nobg()*** Two Voigt functions added together

***double_voigt()*** Two Voigt function and a background added together

#### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

### double_voigt_nobg

mcalf.profiles.voigt.**double_voigt_nobg**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*, *clib=True*)

Double Voigt function with no background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.
- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.

- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt()*** Two Voigt function and a background added together

### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

## double_voigt

mcalf.profiles.voigt.**double_voigt**(*x*, *a1*, *b1*, *s1*, *g1*, *a2*, *b2*, *s2*, *g2*, *d*, *clib=True*)

Double Voigt function with background

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Voigt function at.
- **a1** (*float*) – Amplitude of 1st Voigt function.
- **b1** (*float*) – Central line core of 1st Voigt function.
- **s1** (*float*) – Sigma (for Gaussian) of 1st Voigt function.
- **g1** (*float*) – Gamma (for Lorentzian) of 1st Voigt function.
- **a2** (*float*) – Amplitude of 2st Voigt function.
- **b2** (*float*) – Central line core of 2st Voigt function.
- **s2** (*float*) – Sigma (for Gaussian) of 2st Voigt function.
- **g2** (*float*) – Gamma (for Lorentzian) of 2st Voigt function.
- **d** (*float*) – Background.
- **clib** (*bool, optional, default = True*) – Whether to use the complied C library or a slower Python version. If using the C library, the accuracy of the integration is reduced to give the code a significant speed boost. Python version can be used when speed is not a priority. Python version will remove deviations that are sometimes present around the wings due to the reduced accuracy.

**Returns** **result** – The value of the Voigt function here.

**Return type** ndarray of shape *x.shape*

**See also:**

***voigt_nobg()*** Base Voigt function with no background

***voigt()*** Voigt function with background added

***double_voigt_nobg()*** Two Voigt functions added together

### Notes

More information on the Voigt function can be found here: https://en.wikipedia.org/wiki/Voigt_profile

## mcalf.profiles.gaussian Module

## Functions

| | |
|---|---|
| *single_gaussian*(x, a, b, c, d) | Gaussian function |
| *skew_normal*(x, a_a, alpha, xi, omega, d) | |
| *skew_normal_with_gaussian*(x, a_a, alpha, xi, ...) | |

### single_gaussian

mcalf.profiles.gaussian.**single_gaussian**(*x*, *a*, *b*, *c*, *d*)

Gaussian function

**Parameters**

- **x** (*ndarray*) – Wavelengths to evaluate Gaussian function at.
- **a** (*float*) – Amplitude.
- **b** (*float*) – Central line core.
- **c** (*float*) – Sigma of Gaussian.
- **d** (*float*) – Background to add.

### skew_normal

mcalf.profiles.gaussian.**skew_normal**(*x*, *a_a*, *alpha*, *xi*, *omega*, *d*)

### skew_normal_with_gaussian

mcalf.profiles.gaussian.**skew_normal_with_gaussian**(*x*, *a_a*, *alpha*, *xi*, *omega*, *a_e*, *b*, *c*, *d*)

### 1.2.4 MCALF visualisation

This sub-package contains:

- Functions to plot the input spectrum and the fitted model (*mcalf.visualisation.spec*).
- Functions to plot the spatial distribution and their general profile (*mcalf.visualisation.classifications*).
- Functions to plot the velocities calculated for a spectral imaging scan (*mcalf.visualisation.velocity*).

#### mcalf.visualisation Package

#### Functions

| | |
|---|---|
| *plot_averaged_class_map*(class_map[, ...]) | Plot an image of the time averaged classifications |
| *plot_class_map*(class_map[, overall_classes, ...]) | Plot an image of the classifications at a particular time along with bar charts of the classifications |
| *plot_classifications*(class_map, spectra, labels) | Plot the spectra separated into their classifications along with an example classified map. |
| *plot_ibis8542*(wavelengths, spectrum[, fit, ...]) | Plot an IBIS8542Model fit |
| *plot_map*(velmap[, mask, umbra_mask, ...]) | Plot the velocity map |
| *plot_spectrum*(wavelengths, spectrum[, ...]) | Plot a spectrum with the wavelength grid shown. |

#### plot_averaged_class_map

mcalf.visualisation.**plot_averaged_class_map**(*class_map*, *classes=None*, *continuous=False*, *xticks=0, 15, 2*, *yticks=0, 15, 2*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *figsize=None*, *dpi=600*, *fontfamily=None*)

Plot an image of the time averaged classifications

**Parameters**

- **class_map** (*ndarray, ndim=3*) – Three-dimensional array of classifications, with the times given in the first dimension.
- **classes** (*ndarray, optional, default = ndarray of [0, 1, 2, 3, 4]*) – Array of all the possible classifications in *class_map*.
- **continuous** (*bool, optional, default = False*) – Whether to plot the with a continuous color scale or round to the nearest classification.
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as. If omitted, the plot will not be saved.

- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.

### plot_class_map

mcalf.visualisation.**plot_class_map**(*class_map*, *overall_classes=None*, *classes=None*, *time_index=None*, *cadence=None*, *xticks=0, 15, 2*, *yticks=0, 15, 2*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *file_prefix='classmap_plot_'*, *file_ext='png'*, *figsize=6.0, 3.6*, *dpi=600*, *fontfamily=None*, *cache=False*)

Plot an image of the classifications at a particular time along with bar charts of the classifications

**Parameters**

- **class_map** (*ndarray, ndim=2 or 3*) – Two-dimensional array of classifications. If three dimensions are given, the first dimension is assumed to represent the time.
- **overall_classes** (*ndarray or bool, optional*) – The percentage of spectra that belong to each classification in the overall dataset. If omitted, these will be calculated used all of the classifications given is *class_map*. If true is given, these will also be calculated in the same way and returned without any plotting done. (This returned array can then be used to speed up later calls of this function.)
- **classes** (*ndarray, optional, default = ndarray of [0, 1, 2, 3, 4]*) – Array of all the possible classifications in *class_map*.
- **time_index** (*int, optional, default = 0*) – The index of the time dimension of *class_map*, required if class_map is 3D. Also used for plotting the time.
- **cadence** (*float, units = seconds, optional, default = None*) – If given, the time index will be multiplied by this value and converted into a time in minutes on the plot. Otherwise, the *time_index* will be plotted without units.
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **output** (*str or bool, optional, default = None*) – If present, the filename to save the plot as. If omitted, the plot will not be saved. If true, the filename will be generated using the *time_index* along with the *file_prefix* and *file_ext*.
- **file_prefix** (str, optional, default = '**classmap_plot_**') – The prefix to use in the filename when *output* is true.
- **file_ext** (*str, optional, default = 'png'*) – The file extension (without the dot) to use when *output* is true.

- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **cache** (*bool, optional, default = False*) – If true, the plot will not be regenerated if the output filename already exists.

**Returns** **overall_classes** – If *overall_classes* is initially true, their calculated values will be returned.

**Return type** ndarray

### plot_classifications

mcalf.visualisation.**plot_classifications**(*class_map*, *spectra*, *labels*, *extent=0, 200, 0, 200*, *xticks=0, 15, 3*, *yticks=0, 15, 3*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *figsize=None*, *dpi=600*, *fontfamily=None*)

Plot the spectra separated into their classifications along with an example classified map.

Must be 5 classifications.

**Parameters**

- **class_map** (*ndarray, ndim=2*) – Two-dimensional array of classifications.
- **spectra** (*ndarray, ndim=2*) – Two-dimensional array with dimensions [spectra, wavelengths].
- **labels** (ndarray, ndim=1, length of *spectra*) – List of classifications for each spectrum in *spectra*.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **vmin** (float, optional, default = -max(|`velmap`|)) – Minimum velocity to plot. If not given, will be -vmax, for vmax not None.
- **vmax** (float, optional, default = max(|`velmap`|)) – Maximum velocity to plot. If not given, will be -vmin, for vmin not None.
- **extent** (*4-tuple, optional, default = (0, 200, 0, 200)*) – Region the *velmap* is cropped to.
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.

- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.

### plot_ibis8542

mcalf.visualisation.**plot_ibis8542**(*wavelengths*, *spectrum*, *fit=None*, *background=0*, *sigma=None*, *sigma_scale=70*, *stationary_line_core=None*, *subtraction=False*, *separate=False*, *output=None*, *figsize=None*, *legend_position='best'*, *dpi=None*, *fontfamily=None*, *reduced_legend=False*, *show_intensity=True*, *hook=None*)

Plot an IBIS8542Model fit

It is recommended to use the plot method on either an IBIS8542Model or a FitResult from an IBIS8542Model instead.

**Parameters**

- **wavelengths** (*ndarray*) – The x-axis values.
- **spectrum** (*ndarray, length=n_wavelengths*) – The y-axis values.
- **fit** (*array_like, optional, default = None*) – The fitted parameters.
- **background** (*float or ndarray of length n_wavelengths, optional, default = 0*) – The background to add to the fitted profiles.
- **sigma** (*ndarray, length=n_wavelengths, optional, default = None*) – The sigma profile used when fitting the parameters to *spectrum*. If given, will be plotted as shaded regions.
- **sigma_scale** (*float, optional, default = 70*) – A factor to multiply the error bars to change their prominence.
- **stationary_line_core** (*float, optional, default = None*) – If given, will show a dashed line at this wavelength.
- **subtraction** (*bool, optional, default = False*) – Whether to plot the *spectrum* minus emission fit (if exists) instead.
- **separate** (*bool, optional, default = False*) – Whether to plot the fitted profiles separately (if multiple components exist).
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **figsize** (*2-tuple, optional*) – Size of the figure.
- **legend_position** (*str or int or pair of floats, optional, default = 'best'*) – Position of the legend. See *matplotlib.pyplot.legend* documentation from possible values.
- **dpi** (*int*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **reduced_legend** (*bool, optional, default = False*) – Whether to add to the legend the labels that would be displayed on an absorption only plot. Useful for saving space when plotting both a single component fit and a multi-component fit alongside each other.

- **show_intensity** (*bool, optional, default = True*) – Whether to show the intensity axis tick labels and axis label.
- **hook** (*callable, optional, default = None*) – If provided this function must accept the current `plt' as a single argument such that it can operate upon it and make changes to the plot.

**See also:**

**models.IBIS8542.plot()** General plotting method

**models.IBIS8542.plot_separate()** Plot the fit parameters separately

**models.IBIS8542.plot_subtraction()** Plot the spectrum with the emission fit subtracted from it

**models.FitResult.plot()** Plotting method on the fit result

### plot_map

mcalf.visualisation.**plot_map**(*velmap*, *mask=None*, *umbra_mask=None*, *figsize=None*, *vmin=None*, *vmax=None*, *extent=None*, *xticks=0, 15, 2*, *yticks=0, 15, 2*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *dpi=600*, *fontfamily=None*, *units='km/s'*, *linewidths=None*)

Plot the velocity map

Plots a velocity map for publication.

**Parameters**

- **velmap** (*ndarray, ndim=2*) – Two-dimensional array of velocities.
- **mask** (ndarray of bool, ndim=2, shape *velmap*, optional, default = None) – Mask showing the region where velocities were found for. True is outside the velocity region and False is where valid velocities should be found. Specifying a mask allows for errors in the velocity calculation to be black and points outside the region to be gray. If omitted, all invalid points will be gray.
- **umbra_mask** (ndarray of bool, ndim=2, shape *velmap*, optional, default = None) – A mask of the umbra, True outside, False inside. If given, a contour will outline the umbra, or other feature the mask represents.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **figsize** (*2-tuple*) – Size of the figure.
- **dpi** (*int*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **vmin** (float, optional, default = -max(|`velmap`|)) – Minimum velocity to plot. If not given, will be -vmax, for vmax not None.
- **vmax** (float, optional, default = max(|`velmap`|)) – Maximum velocity to plot. If not given, will be -vmin, for vmin not None.
- **extent** (*4-tuple, optional, default = (0, n_rows, 0, n_columns)*) – Data range of *velmap*. TODO: Remove (assume one-to-one relationship)

- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **units** (*str, optional, default = 'km/s'*) – The units of *velmap* data. Printed on colorbar.
- **linewidths** (*float or sequence of floats, optional, default = None*) – The width of the contours plotted for *umbra_mask*.

### plot_spectrum

mcalf.visualisation.**plot_spectrum**(*wavelengths*, *spectrum*, *output=None*, *normalised=True*, *smooth=True*, *figsize=7, 3*, *dpi=600*, *fontfamily=None*)

Plot a spectrum with the wavelength grid shown.

Intended for plotting the raw data.

**Parameters**

- **wavelengths** (*ndarray*) – The x-axis values.
- **spectrum** (*ndarray, length=n_wavelengths*) – The y-axis values.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **normalised** (*bool, optional, default = True*) – Whether to normalise the spectrum using the last three spectral points.
- **smooth** (*bool, optional, default = True*) – Whether to smooth the *spectrum* with a spline.
- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.

### mcalf.visualisation.spec Module

### Functions

| | |
|---|---|
| *plot_ibis8542*(wavelengths, spectrum[, fit, ...]) | Plot an IBIS8542Model fit |
| *plot_spectrum*(wavelengths, spectrum[, ...]) | Plot a spectrum with the wavelength grid shown. |

### plot_ibis8542

mcalf.visualisation.spec.**plot_ibis8542**(*wavelengths*, *spectrum*, *fit=None*, *background=0*, *sigma=None*, *sigma_scale=70*, *stationary_line_core=None*, *subtraction=False*, *separate=False*, *output=None*, *figsize=None*, *legend_position='best'*, *dpi=None*, *fontfamily=None*, *reduced_legend=False*, *show_intensity=True*, *hook=None*)

Plot an IBIS8542Model fit

It is recommended to use the plot method on either an IBIS8542Model or a FitResult from an IBIS8542Model instead.

**Parameters**

- **wavelengths** (*ndarray*) – The x-axis values.
- **spectrum** (*ndarray, length=n_wavelengths*) – The y-axis values.
- **fit** (*array_like, optional, default = None*) – The fitted parameters.
- **background** (*float or ndarray of length n_wavelengths, optional, default = 0*) – The background to add to the fitted profiles.
- **sigma** (*ndarray, length=n_wavelengths, optional, default = None*) – The sigma profile used when fitting the parameters to *spectrum*. If given, will be plotted as shaded regions.
- **sigma_scale** (*float, optional, default = 70*) – A factor to multiply the error bars to change their prominence.
- **stationary_line_core** (*float, optional, default = None*) – If given, will show a dashed line at this wavelength.
- **subtraction** (*bool, optional, default = False*) – Whether to plot the *spectrum* minus emission fit (if exists) instead.
- **separate** (*bool, optional, default = False*) – Whether to plot the fitted profiles separately (if multiple components exist).
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **figsize** (*2-tuple, optional*) – Size of the figure.
- **legend_position** (*str or int or pair of floats, optional, default = 'best'*) – Position of the legend. See *matplotlib.pyplot.legend* documentation from possible values.
- **dpi** (*int*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **reduced_legend** (*bool, optional, default = False*) – Whether to add to the legend the labels that would be displayed on an absorption only plot. Useful for saving space when plotting both a single component fit and a multi-component fit alongside each other.
- **show_intensity** (*bool, optional, default = True*) – Whether to show the intensity axis tick labels and axis label.

- **hook** (*callable, optional, default = None*) – If provided this function must accept the current `plt' as a single argument such that it can operate upon it and make changes to the plot.

**See also:**

**models.IBIS8542.plot()** General plotting method

**models.IBIS8542.plot_separate()** Plot the fit parameters separately

**models.IBIS8542.plot_subtraction()** Plot the spectrum with the emission fit subtracted from it

**models.FitResult.plot()** Plotting method on the fit result

### plot_spectrum

mcalf.visualisation.spec.**plot_spectrum**(*wavelengths*, *spectrum*, *output=None*, *normalised=True*, *smooth=True*, *figsize=7, 3*, *dpi=600*, *fontfamily=None*)

Plot a spectrum with the wavelength grid shown.

Intended for plotting the raw data.

**Parameters**

- **wavelengths** (*ndarray*) – The x-axis values.
- **spectrum** (*ndarray, length=n_wavelengths*) – The y-axis values.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **normalised** (*bool, optional, default = True*) – Whether to normalise the spectrum using the last three spectral points.
- **smooth** (*bool, optional, default = True*) – Whether to smooth the *spectrum* with a spline.
- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.

## mcalf.visualisation.classifications Module

### Functions

| | |
|---|---|
| *plot_classifications*(class_map, spectra, labels) | Plot the spectra separated into their classifications along with an example classified map. |
| *plot_class_map*(class_map[, overall_classes, ...]) | Plot an image of the classifications at a particular time along with bar charts of the classifications |
| *plot_averaged_class_map*(class_map[, ...]) | Plot an image of the time averaged classifications |

### plot_classifications

mcalf.visualisation.classifications.**plot_classifications**(*class_map*, *spectra*, *labels*, *extent=0, 200, 0, 200*, *xticks=0, 15, 3*, *yticks=0, 15, 3*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *figsize=None*, *dpi=600*, *fontfamily=None*)

Plot the spectra separated into their classifications along with an example classified map.

Must be 5 classifications.

**Parameters**

- **class_map** (*ndarray, ndim=2*) – Two-dimensional array of classifications.
- **spectra** (*ndarray, ndim=2*) – Two-dimensional array with dimensions [spectra, wavelengths].
- **labels** (ndarray, ndim=1, length of *spectra*) – List of classifications for each spectrum in *spectra*.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **vmin** (float, optional, default = -max(|`velmap`|)) – Minimum velocity to plot. If not given, will be -vmax, for vmax not None.
- **vmax** (float, optional, default = max(|`velmap`|)) – Maximum velocity to plot. If not given, will be -vmin, for vmin not None.
- **extent** (*4-tuple, optional, default = (0, 200, 0, 200)*) – Region the *velmap* is cropped to.
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.

### plot_class_map

mcalf.visualisation.classifications.**plot_class_map**(*class_map*, *overall_classes=None*, *classes=None*, *time_index=None*, *cadence=None*, *xticks=0, 15, 2*, *yticks=0, 15, 2*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *file_prefix='classmap_plot_'*, *file_ext='png'*, *figsize=6.0, 3.6*, *dpi=600*, *fontfamily=None*, *cache=False*)

Plot an image of the classifications at a particular time along with bar charts of the classifications

**Parameters**

- **class_map** (*ndarray, ndim=2 or 3*) – Two-dimensional array of classifications. If three dimensions are given, the first dimension is assumed to represent the time.
- **overall_classes** (*ndarray or bool, optional*) – The percentage of spectra that belong to each classification in the overall dataset. If omitted, these will be calculated used all of the classifications given is *class_map*. If true is given, these will also be calculated in the same way and returned without any plotting done. (This returned array can then be used to speed up later calls of this function.)
- **classes** (*ndarray, optional, default = ndarray of [0, 1, 2, 3, 4]*) – Array of all the possible classifications in *class_map*.
- **time_index** (*int, optional, default = 0*) – The index of the time dimension of *class_map*, required if class_map is 3D. Also used for plotting the time.
- **cadence** (*float, units = seconds, optional, default = None*) – If given, the time index will be multiplied by this value and converted into a time in minutes on the plot. Otherwise, the *time_index* will be plotted without units.
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **output** (*str or bool, optional, default = None*) – If present, the filename to save the plot as. If omitted, the plot will not be saved. If true, the filename will be generated using the *time_index* along with the *file_prefix* and *file_ext*.
- **file_prefix** (str, optional, default = '**classmap_plot_**') – The prefix to use in the filename when *output* is true.
- **file_ext** (*str, optional, default = 'png'*) – The file extension (without the dot) to use when *output* is true.
- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.

- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **cache** (*bool, optional, default = False*) – If true, the plot will not be regenerated if the output filename already exists.

**Returns** **overall_classes** – If *overall_classes* is initially true, their calculated values will be returned.

**Return type** ndarray

### plot_averaged_class_map

mcalf.visualisation.classifications.**plot_averaged_class_map**(*class_map*, *classes=None*, *continuous=False*, *xticks=0, 15, 2*, *yticks=0, 15, 2*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *figsize=None*, *dpi=600*, *fontfamily=None*)

Plot an image of the time averaged classifications

**Parameters**

- **class_map** (*ndarray, ndim=3*) – Three-dimensional array of classifications, with the times given in the first dimension.
- **classes** (*ndarray, optional, default = ndarray of [0, 1, 2, 3, 4]*) – Array of all the possible classifications in *class_map*.
- **continuous** (*bool, optional, default = False*) – Whether to plot the with a continuous color scale or round to the nearest classification.
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.
- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as. If omitted, the plot will not be saved.
- **figsize** (*2-tuple, optional, default = None*) – Size of the figure.
- **dpi** (*int, optional, default = 600*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.

### mcalf.visualisation.velocity Module

### Functions

| | |
|---|---|
| *plot_map*(velmap[, mask, umbra_mask, ...]) | Plot the velocity map |

### plot_map

mcalf.visualisation.velocity.**plot_map**(*velmap*, *mask=None*, *umbra_mask=None*, *figsize=None*, *vmin=None*, *vmax=None*, *extent=None*, *xticks=0, 15, 2*, *yticks=0, 15, 2*, *xscale=0.070325*, *yscale=0.070325*, *output=None*, *dpi=600*, *fontfamily=None*, *units='km/s'*, *linewidths=None*)

Plot the velocity map

Plots a velocity map for publication.

**Parameters**

- **velmap** (*ndarray, ndim=2*) – Two-dimensional array of velocities.
- **mask** (ndarray of bool, ndim=2, shape *velmap*, optional, default = None) – Mask showing the region where velocities were found for. True is outside the velocity region and False is where valid velocities should be found. Specifying a mask allows for errors in the velocity calculation to be black and points outside the region to be gray. If omitted, all invalid points will be gray.
- **umbra_mask** (ndarray of bool, ndim=2, shape *velmap*, optional, default = None) – A mask of the umbra, True outside, False inside. If given, a contour will outline the umbra, or other feature the mask represents.
- **output** (*str, optional, default = None*) – If present, the filename to save the plot as.
- **figsize** (*2-tuple*) – Size of the figure.
- **dpi** (*int*) – The number of dots per inch. For controlling the quality of the outputted figure.
- **fontfamily** (*str, optional, default = None*) – If provided, this family string will be added to the 'font' rc params group.
- **vmin** (float, optional, default = -max(|`velmap`|)) – Minimum velocity to plot. If not given, will be -vmax, for vmax not None.
- **vmax** (float, optional, default = max(|`velmap`|)) – Maximum velocity to plot. If not given, will be -vmin, for vmin not None.
- **extent** (*4-tuple, optional, default = (0, n_rows, 0, n_columns)*) – Data range of *velmap*. TODO: Remove (assume one-to-one relationship)
- **xticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the x-axis ticks in Mm.
- **yticks** (*3-tuple, optional, default = (0, 15, 2)*) – The start, stop and step for the y-axis ticks in Mm.

- **xscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between x-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **yscale** (*float, optional = 0.725 * 0.097*) – Scaling factor between y-axis data coordinate steps and 1 Mm. Mm = data / xscale.
- **units** (*str, optional, default = 'km/s'*) – The units of *velmap* data. Printed on colorbar.
- **linewidths** (*float or sequence of floats, optional, default = None*) – The width of the contours plotted for *umbra_mask*.

### 1.2.5 MCALF utils

This sub-package contains:

- Functions for processing spectra (*mcalf.utils.spec*).
- Functions for smoothing n-dimensional arrays (*mcalf.utils.smooth*).
- Functions for masking the input data to limit the region computed (*mcalf.utils.mask*).
- Miscellaneous utility functions (*mcalf.utils.misc*).

#### mcalf.utils Package

#### Functions

| | |
|---|---|
| *gaussian_kern_3d*([width, sigma]) | 3D Gaussian kernel |
| *generate_sigma*(sigma_type, wavelengths, ...) | Generate the default sigma profiles |
| *genmask*(width, height[, radius, ...]) | Generate a circular mask of specified size |
| *load_parameter*(parameter[, wl]) | Load parameters from file, optionally evaluating variables from strings |
| *make_iter*(*args) | Returns each inputted argument, wrapping in a list if not already iterable |
| *moving_average*(array, width) | Boxcar moving average |
| *normalise_spectrum*(spectrum[, ...]) | Normalise an individual spectrum to have intensities in range [0, 1] |
| *radial_distances*(n_cols, n_rows) | Generates a 2D array of specified shape of radial distances from the centre |
| *reinterpolate_spectrum*(spectrum, ...) | Reinterpolate the spectrum |
| *smooth_cube*(cube, mask, **kwargs) | Apply Gaussian smoothing to velocities |

#### gaussian_kern_3d

mcalf.utils.**gaussian_kern_3d**(*width=5*, *sigma=1, 1, 1*)

3D Gaussian kernel

Create a Gaussian kernel of shape *width`*`width`*`width*.

**Parameters**

- **width** (*int, optional, default = 5*) – Length of all three dimensions of the Gaussian kernel.

- **sigma** (*array_like, tuple, optional, default = (1, 1, 1)*) – Sigma values for the time, horizontal and vertical dimensions.

**Returns** **kernel** – The generated kernel.

**Return type** ndarray, shape (*width*, *width*, *width*)

### Examples

```
>>> gaussian_kern_3d(width=3, sigma=(2, 1, 1.5))
array([[[0.42860385, 0.53526143, 0.42860385],
        [0.48567179, 0.60653066, 0.48567179],
        [0.42860385, 0.53526143, 0.42860385]],
       [[0.70664828, 0.8824969 , 0.70664828],
        [0.8007374 , 1.        , 0.8007374 ],
        [0.70664828, 0.8824969 , 0.70664828]],
       [[0.42860385, 0.53526143, 0.42860385],
        [0.48567179, 0.60653066, 0.48567179],
        [0.42860385, 0.53526143, 0.42860385]]])
```

## generate_sigma

mcalf.utils.**generate_sigma**(*sigma_type*, *wavelengths*, *line_core*, *a=- 0.95*, *c=0.04*, *d=1*, *centre_rad=7*, *a_peak=0.4*)

Generate the default sigma profiles

**Parameters**

- **sigma_type** (*int*) – Type of profile to generate. Should be either *1* or *2*.
- **wavelengths** (*array_like*) – Wavelengths to use for sigma profile.
- **line_core** (*float*) – Line core to use as centre of Gaussian sigma profile.
- **a** (*float, optional, default = -0.95*) – Amplitude of Gaussian sigma profile.
- **c** (*float, optional, default = 0.04*) – Sigma of Gaussian sigma profile.
- **d** (*float, optional, default = 1*) – Background of Gaussian sigma profile.
- **centre_rad** (*int, optional, default = 7*) – Width of central flattened region.
- **a_peak** (*float, optional, default = 0.4*) – Amplitude of central 7 pixel section, if *sigma_type* is 2.

**Returns** **sigma** – The generated sigma profile

**Return type** ndarray, length = n_wavelengths

### genmask

mcalf.utils.**genmask**(*width*, *height*, *radius=inf*, *right_shift=0*, *up_shift=0*)

Generate a circular mask of specified size

**Parameters**

- **width** (*int*) – Width of mask.
- **height** (*int*) – Height of mask.
- **radius** (*int, optional, default = inf*) – Radius of mask.
- **right_shift** (*int, optional, default = 0*) – Indices to shift forward through row.
- **up_shift** (*int, optional, default = 0*) – Indices to shift forward through columns.

**Returns** **array** – The generated mask.

**Return type** ndarray of shape (height, width)

### load_parameter

mcalf.utils.**load_parameter**(*parameter*, *wl=None*)

Load parameters from file, optionally evaluating variables from strings

Loads the parameter from string or file.

**Parameters**

- **parameter** (*str*) – Parameter to load, either string of Python list/number or filename string. Supported filename extensions are '.fits', '.fit', '.fts', '.csv', '.txt', '.npy', '.npz', and '.sav'. If the file does not exist, it will assume the string is a Python expression.
- **wl** (*float, optional, default = None*) – Central line core wavelength to replace 'wl' in strings. Will only replace occurrences in the *parameter* variable itself or in files with extension ".csv" or ".txt". When using *wl*, also use 'inf' and 'nan' as required.

**Returns** **value** – Value of parameter in easily computable format (not string)

**Return type** ndarray or list of floats

#### Examples

```
>>> load_parameter("wl + 4.2", wl=7.1)
11.3
```

```
>>> load_parameter("[wl + 4.2, 5.2 - inf, 5 > 3]", wl=7.1)
[11.3, -inf, 1.0]
```

Filenames are given as follows:

```
>>> x = load_parameter("datafile.csv", wl=12.4)
```

```
>>> x = load_parameter("datafile.fits")
```

If the file does not exist, the function will assume that the string is a Python expression, possibly leading to an error:

```
>>> load_parameter("nonexistant.csv")
TypeError: 'NoneType' object is not subscriptable
```

### make_iter

mcalf.utils.**make_iter**(**args*)

Returns each inputted argument, wrapping in a list if not already iterable

**Parameters** ***args** – Arguments to make iterable.

**Returns** **args* converted to iterables.

**Return type** iterables

#### Examples

```
>>> make_iter(1)
[[1]]
```

```
>>> make_iter(1, 2, 3)
[[1], [2], [3]]
```

```
>>> make_iter(1, [2], 3)
[[1], [2], [3]]
```

It is intended that a list of arguments be passed to the function for conversion:

```
>>> make_iter(*[1, [2], 3])
[[1], [2], [3]]
```

Remember that strings are already iterable!

```
>>> make_iter(*[[1, 2, 3], (4, 5, 6), "a"])
[[1, 2, 3], (4, 5, 6), 'a']
```

### moving_average

mcalf.utils.**moving_average**(*array*, *width*)

Boxcar moving average

Calculate the moving average of an array with a boxcar of defined width. An odd width is recommended.

**Parameters**

- **array** (*ndarray, ndim=1*) – Array to find the moving average of.
- **width** (*int*) – Width of the boxcar. Odd integer recommended. Less than or equal to length of *array*.

**Returns** **averaged** – Averaged array.

**Return type** ndarray of shape *array*

### Notes

The moving average is calculated at each point of the *array* by finding the (unweighted) mean of the subarrays of length given by *width*. These subarrays are centred at the point in the *array* that the current average is currently being calculated for. If an odd *width* is chosen, the sub array will include the current point plus an equal number of points on either side. However, if an even *width* is chosen, the sub array will bias including the extra point to the left of the current index. If the subarray spans past the boundaries, the values beyond the boundary is ignored and the mean is calculated by dividing by the number of points that are inside the boundaries.

### Examples

```
>>> x = np.array([1, 2, 3, 4, 5])
>>> moving_average(x, 3)
array([1.5, 2. , 3. , 4. , 4.5])
>>> moving_average(x, 2)
array([1. , 1.5, 2.5, 3.5, 4.5])
```

## normalise_spectrum

mcalf.utils.**normalise_spectrum**(*spectrum*, *original_wavelengths=None*, *constant_wavelengths=None*, *prefilter_response=None*, *model=None*)

Normalise an individual spectrum to have intensities in range [0, 1]

Not recommended for normalising many spectra in a loop.

**Parameters**

- **spectrum** (*ndarray of ndim=1*) – Spectrum to reinterpolate and normalise.
- **original_wavelengths** (ndarray of ndim=1, length=length of *spectrum*, optional) – Wavelengths of *spectrum*.
- **constant_wavelengths** (*ndarray of ndim=1, optional*) – Wavelengths to cast *spectrum* into.
- **prefilter_response** (ndarray of ndim=1, length=length of *constant_wavelengths*, optional) – Prefilter response to divide spectrum by.
- **model** (ModelBase, *optional*) – Model to extract the above parameters from.

**Returns** **spectrum** – The normalised spectrum.

**Return type** ndarray of ndim-1, length=length of *constant_wavelengths*

## radial_distances

mcalf.utils.**radial_distances**(*n_cols*, *n_rows*)

Generates a 2D array of specified shape of radial distances from the centre

**Parameters**

- **n_cols** (*int*) – Number of columns.
- **n_rows** (*int*) – Number of rows.

**Returns** **array** – Array of radial distances.

**Return type** ndarray of shape (n_rows, n_cols)

**See also:**

*genmask()* Generates a circular mask

### reinterpolate_spectrum

mcalf.utils.**reinterpolate_spectrum**(*spectrum*, *original_wavelengths*, *constant_wavelengths*)

Reinterpolate the spectrum

Reinterpolates the spectrum such that intensities at *original_wavelengths* are transformed into intensities at *constant_wavelengths*. Uses *scipy.interpolate.InterpolatedUnivariateSpline* to interpolate.

**Parameters**

- **spectrum** (*ndarray of ndim=1*) – Spectrum to reinterpolate.
- **original_wavelengths** (ndarray of ndim=1, length=length of *spectrum*) – Wavelengths of *spectrum*.
- **constant_wavelengths** (*ndarray of ndim=1*) – Wavelengths to cast *spectrum* into.

**Returns** **spectrum** – Reinterpolated spectrum.

**Return type** ndarray, length=length of *constant_wavelengths*

### smooth_cube

mcalf.utils.**smooth_cube**(*cube*, *mask*, ***kwargs*)

Apply Gaussian smoothing to velocities

Smooth the cube of velocities with a Gaussian kernel, applying weights at boundaries.

**Parameters**

- **cube** (*ndarray, ndim=3*) – Cube of velocities with dimensions [time, row, column].
- **mask** (*ndarray, ndim=2*) – The mask to apply to the [row, column] at every time. Points that are 0 or false will be removed.
- **kwargs** (*optional*) – Keyword arguments to pass to *gaussian_kern_3d*.

**Returns** **cube_** – The smoothed cube.

**Return type** ndarray, shape *cube*

## mcalf.utils.spec Module

### Functions

| | |
|---|---|
| *reinterpolate_spectrum*(spectrum, ...) | Reinterpolate the spectrum |
| *normalise_spectrum*(spectrum[, ...]) | Normalise an individual spectrum to have intensities in range [0, 1] |
| *generate_sigma*(sigma_type, wavelengths, ...) | Generate the default sigma profiles |

### reinterpolate_spectrum

mcalf.utils.spec.**reinterpolate_spectrum**(*spectrum*, *original_wavelengths*, *constant_wavelengths*)

Reinterpolate the spectrum

Reinterpolates the spectrum such that intensities at *original_wavelengths* are transformed into intensities at *constant_wavelengths*. Uses *scipy.interpolate.InterpolatedUnivariateSpline* to interpolate.

**Parameters**

- **spectrum** (*ndarray of ndim=1*) – Spectrum to reinterpolate.
- **original_wavelengths** (ndarray of ndim=1, length=length of *spectrum*) – Wavelengths of *spectrum*.
- **constant_wavelengths** (*ndarray of ndim=1*) – Wavelengths to cast *spectrum* into.

**Returns** **spectrum** – Reinterpolated spectrum.

**Return type** ndarray, length=length of *constant_wavelengths*

### normalise_spectrum

mcalf.utils.spec.**normalise_spectrum**(*spectrum*, *original_wavelengths=None*, *constant_wavelengths=None*, *prefilter_response=None*, *model=None*)

Normalise an individual spectrum to have intensities in range [0, 1]

Not recommended for normalising many spectra in a loop.

**Parameters**

- **spectrum** (*ndarray of ndim=1*) – Spectrum to reinterpolate and normalise.
- **original_wavelengths** (ndarray of ndim=1, length=length of *spectrum*, optional) – Wavelengths of *spectrum*.
- **constant_wavelengths** (*ndarray of ndim=1, optional*) – Wavelengths to cast *spectrum* into.
- **prefilter_response** (ndarray of ndim=1, length=length of *constant_wavelengths*, optional) – Prefilter response to divide spectrum by.
- **model** (ModelBase, *optional*) – Model to extract the above parameters from.

**Returns** **spectrum** – The normalised spectrum.

**Return type** ndarray of ndim-1, length=length of *constant_wavelengths*

### generate_sigma

mcalf.utils.spec.**generate_sigma**(*sigma_type*, *wavelengths*, *line_core*, *a=- 0.95*, *c=0.04*, *d=1*, *centre_rad=7*, *a_peak=0.4*)

Generate the default sigma profiles

**Parameters**

- **sigma_type** (*int*) – Type of profile to generate. Should be either *1* or *2*.
- **wavelengths** (*array_like*) – Wavelengths to use for sigma profile.
- **line_core** (*float*) – Line core to use as centre of Gaussian sigma profile.
- **a** (*float, optional, default = -0.95*) – Amplitude of Gaussian sigma profile.
- **c** (*float, optional, default = 0.04*) – Sigma of Gaussian sigma profile.
- **d** (*float, optional, default = 1*) – Background of Gaussian sigma profile.
- **centre_rad** (*int, optional, default = 7*) – Width of central flattened region.
- **a_peak** (*float, optional, default = 0.4*) – Amplitude of central 7 pixel section, if *sigma_type* is 2.

**Returns** **sigma** – The generated sigma profile

**Return type** ndarray, length = n_wavelengths

## mcalf.utils.smooth Module

### Functions

| | |
|---|---|
| *moving_average*(array, width) | Boxcar moving average |
| *gaussian_kern_3d*([width, sigma]) | 3D Gaussian kernel |
| *smooth_cube*(cube, mask, **kwargs) | Apply Gaussian smoothing to velocities |

### moving_average

mcalf.utils.smooth.**moving_average**(*array*, *width*)

Boxcar moving average

Calculate the moving average of an array with a boxcar of defined width. An odd width is recommended.

**Parameters**

- **array** (*ndarray, ndim=1*) – Array to find the moving average of.
- **width** (*int*) – Width of the boxcar. Odd integer recommended. Less than or equal to length of *array*.

**Returns** **averaged** – Averaged array.

**Return type** ndarray of shape *array*

### Notes

The moving average is calculated at each point of the *array* by finding the (unweighted) mean of the subarrays of length given by *width*. These subarrays are centred at the point in the *array* that the current average is currently being calculated for. If an odd *width* is chosen, the sub array will include the current point plus an equal number of points on either side. However, if an even *width* is chosen, the sub array will bias including the extra point to the left of the current index. If the subarray spans past the boundaries, the values beyond the boundary is ignored and the mean is calculated by dividing by the number of points that are inside the boundaries.

### Examples

```
>>> x = np.array([1, 2, 3, 4, 5])
>>> moving_average(x, 3)
array([1.5, 2. , 3. , 4. , 4.5])
>>> moving_average(x, 2)
array([1. , 1.5, 2.5, 3.5, 4.5])
```

## gaussian_kern_3d

mcalf.utils.smooth.**gaussian_kern_3d**(*width=5*, *sigma=1, 1, 1*)
3D Gaussian kernel

Create a Gaussian kernel of shape *width`*`width`*`width*.

**Parameters**

- **width** (*int, optional, default = 5*) – Length of all three dimensions of the Gaussian kernel.
- **sigma** (*array_like, tuple, optional, default = (1, 1, 1)*) – Sigma values for the time, horizontal and vertical dimensions.

**Returns** **kernel** – The generated kernel.

**Return type** ndarray, shape (*width*, *width*, *width*)

### Examples

```
>>> gaussian_kern_3d(width=3, sigma=(2, 1, 1.5))
array([[[0.42860385, 0.53526143, 0.42860385],
        [0.48567179, 0.60653066, 0.48567179],
        [0.42860385, 0.53526143, 0.42860385]],
       [[0.70664828, 0.8824969 , 0.70664828],
        [0.8007374 , 1.        , 0.8007374 ],
        [0.70664828, 0.8824969 , 0.70664828]],
       [[0.42860385, 0.53526143, 0.42860385],
        [0.48567179, 0.60653066, 0.48567179],
        [0.42860385, 0.53526143, 0.42860385]]])
```

### smooth_cube

mcalf.utils.smooth.**smooth_cube**(*cube*, *mask*, ***kwargs*)

Apply Gaussian smoothing to velocities

Smooth the cube of velocities with a Gaussian kernel, applying weights at boundaries.

**Parameters**

- **cube** (*ndarray, ndim=3*) – Cube of velocities with dimensions [time, row, column].
- **mask** (*ndarray, ndim=2*) – The mask to apply to the [row, column] at every time. Points that are 0 or false will be removed.
- **kwargs** (*optional*) – Keyword arguments to pass to *gaussian_kern_3d*.

**Returns** **cube_** – The smoothed cube.

**Return type** ndarray, shape *cube*

## mcalf.utils.mask Module

### Functions

| | |
|---|---|
| *genmask*(width, height[, radius, ...]) | Generate a circular mask of specified size |
| *radial_distances*(n_cols, n_rows) | Generates a 2D array of specified shape of radial distances from the centre |

### genmask

mcalf.utils.mask.**genmask**(*width*, *height*, *radius=inf*, *right_shift=0*, *up_shift=0*)

Generate a circular mask of specified size

**Parameters**

- **width** (*int*) – Width of mask.
- **height** (*int*) – Height of mask.
- **radius** (*int, optional, default = inf*) – Radius of mask.
- **right_shift** (*int, optional, default = 0*) – Indices to shift forward through row.
- **up_shift** (*int, optional, default = 0*) – Indices to shift forward through columns.

**Returns** **array** – The generated mask.

**Return type** ndarray of shape (height, width)

### radial_distances

mcalf.utils.mask.**radial_distances**(*n_cols*, *n_rows*)

Generates a 2D array of specified shape of radial distances from the centre

**Parameters**

- **n_cols** (*int*) – Number of columns.
- **n_rows** (*int*) – Number of rows.

**Returns** **array** – Array of radial distances.

**Return type** ndarray of shape (n_rows, n_cols)

**See also:**

***genmask()*** Generates a circular mask

## mcalf.utils.misc Module

## Functions

| | |
|---|---|
| *make_iter*(*args) | Returns each inputted argument, wrapping in a list if not already iterable |
| *load_parameter*(parameter[, wl]) | Load parameters from file, optionally evaluating variables from strings |

### make_iter

mcalf.utils.misc.**make_iter**(**args*)

Returns each inputted argument, wrapping in a list if not already iterable

**Parameters** ***args** – Arguments to make iterable.

**Returns** **args* converted to iterables.

**Return type** iterables

#### Examples

```
>>> make_iter(1)
[[1]]
```

```
>>> make_iter(1, 2, 3)
[[1], [2], [3]]
```

```
>>> make_iter(1, [2], 3)
[[1], [2], [3]]
```

It is intended that a list of arguments be passed to the function for conversion:

```
>>> make_iter(*[1, [2], 3])
[[1], [2], [3]]
```

Remember that strings are already iterable!

```
>>> make_iter(*[[1, 2, 3], (4, 5, 6), "a"])
[[1, 2, 3], (4, 5, 6), 'a']
```

### load_parameter

mcalf.utils.misc.**load_parameter**(*parameter*, *wl=None*)

Load parameters from file, optionally evaluating variables from strings

Loads the parameter from string or file.

**Parameters**

- **parameter** (*str*) – Parameter to load, either string of Python list/number or filename string. Supported filename extensions are '.fits', '.fit', '.fts', '.csv', '.txt', '.npy', '.npz', and '.sav'. If the file does not exist, it will assume the string is a Python expression.
- **wl** (*float, optional, default = None*) – Central line core wavelength to replace 'wl' in strings. Will only replace occurrences in the *parameter* variable itself or in files with extension ".csv" or ".txt". When using *wl*, also use 'inf' and 'nan' as required.

**Returns** **value** – Value of parameter in easily computable format (not string)

**Return type** ndarray or list of floats

#### Examples

```
>>> load_parameter("wl + 4.2", wl=7.1)
11.3
```

```
>>> load_parameter("[wl + 4.2, 5.2 - inf, 5 > 3]", wl=7.1)
[11.3, -inf, 1.0]
```

Filenames are given as follows:

```
>>> x = load_parameter("datafile.csv", wl=12.4)
```

```
>>> x = load_parameter("datafile.fits")
```

If the file does not exist, the function will assume that the string is a Python expression, possibly leading to an error:

```
>>> load_parameter("nonexistant.csv")
TypeError: 'NoneType' object is not subscriptable
```

## 1.3 Contributor Covenant Code of Conduct

### 1.3.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

### 1.3.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 1.3.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

### 1.3.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

### 1.3.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at mcalf@macbride.me. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

### 1.3.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

#### 1. Correction

**Community Impact**: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

**Consequence**: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

#### 2. Warning

**Community Impact**: A violation through a single incident or series of actions.

**Consequence**: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

#### 3. Temporary Ban

**Community Impact**: A serious violation of community standards, including sustained inappropriate behavior.

**Consequence**: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

#### 4. Permanent Ban

**Community Impact**: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

**Consequence**: A permanent ban from any sort of public interaction within the community.

### 1.3.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

## 1.4 MCALF Licence

# PYTHON MODULE INDEX

## m

# INDEX

## Symbols



## A



## B



## C



## D



## E

## F



## G



## I



## L



## M

## N



## O



## P



## Q



## R



## S

## T



## V