eispac

Release 0.1.dev108+gdfa97b1

NRL EIS Team

Sep 23, 2023

CONTENTS

Welcome to the documentation for the EIS Python Analysis Code (eispac)! A PDF copy of the entire online documentation can be downloaded here

CHAPTER

ONE

INSTALLATION

1.1 Requirements

EISPAC depends on a number of Python packages that are commonly used in scientific and solar research. Normally, the installation process will automatically check and install missing dependencies, assuming your environment is configured appropriately. If it does not, you may wish to try installing the required packages individually first.

- python >= 3.8
- numpy >= 1.18
- scipy >= 1.4
- matplotlib >= 3.1
- h5py >= 2.9
- astropy >= 4.2.1
- sunpy >= 4.0
- ndcube >= 2.0
- pyqt >= 5.9
- parfive >= 1.5
- python-dateutil >= 2.8

1.2 Standard Installation

EISPAC is now available on PyPI. To install, just use the following command,

```
>>> python -m pip install eispac
```

To upgrade the package, please use:

```
>>> python -m pip install --upgrade eispac
```

pip should automatically install all package dependencies. If it does not, please see the list of required packages above. Note: if you are using conda to manage your Python packages, you may wish to install or update the dependencies manually first, before installing eispac using pip (this is by no means required, but it can help simplify updating packages).

1.3 Manual Installation

1. Download or clone "eispac" to a convenient location on your computer (it does not matter where).

```
>>> git clone https://github.com/USNavalResearchLaboratory/eispac.git
```

2. Open a terminal and navigate to the directory
3. To install:

```
>>> python -m pip install .
```

4. To upgrade:

```
>>> python -m pip install --upgrade .
```

The package should then be installed to the correct location for your current Python environment. You can now import the package using `import eispac`.

Now, you should be all set to do some science!

CHAPTER

TWO

USER'S GUIDE

Welcome to the user's guide for eispac! This guide will help you get started with eispac quickly and provides a high-level introduction to the core features and key functions. For more details about specific functions, please see the *API Reference* section.

Need help or want to contribute code? Please see the *User Support* or *Contributing to the Code* sections of the community guidelines.

2.1 Introduction

The EIS Python Analysis Code (EISPAC) is a software package designed for easy and accurate analysis of spectroscopic data from the EIS instrument on board the Hinode spacecraft.

2.1.1 What Can EISPAC Do?

- Search and download EIS observations
- Explore data contents
- Fit spectral line using multigaussian fit templates
- Compute measurements (line intensities, velocities, and widths)
- Generate coordinate-aware, Sunpy Maps of the fit measurements

2.1.2 Subcomponents of EISPAC

There are three main subcomponents of EISPAC,

1. **An archive of Level-1 data files**

   These files are saved in the HDF5 file format and come in pairs of header and data files. The archive is updated regularly and can be found at https://eis.nrl.navy.mil/

2. **A set of GUI and command line tools**

   Can be used to quickly search and and download EIS data, view available fit templates, and batch process multiple files at once using parallel processing.

3. **The Python package itself**

   Provides classes and functions that can read the Level-1 HDF5 files, perform all of the necessary calibration and pointing adjustments, and create user-friendly Python objects that can be manipulated as needed. Also included are functions for fitting the intensity profiles with multi-Gaussian functions using template files and a Python

port of the venerable `MPFIT` library[1]. Installing the Python package also automatically installs and registers the GUI and command line tools with your Python environment.

2.1.3 Simple Example

Reading and fitting EIS data requires minimal time and effort. Below is a basic, but complete, example:

```
import eispac

if __name__ == '__main__':
    # input data and template files
    data_filepath = './eis_20190404_131513.data.h5'
    template_filepath = './fe_12_195_119.2c.template.h5'

    # read fit template
    tmplt = eispac.read_template(template_filepath)

    # Read spectral window into an EISCube
    data_cube = eispac.read_cube(data_filepath, tmplt.central_wave)

    # Fit the data using parallel processing
    fit_res = eispac.fit_spectra(data_cube, tmplt, ncpu='max')
```

Citations

2.2 A Quick Guide to EISPAC

Once installed, EISPAC can be imported into any Python script or interactive session with a simple `import eispac` statement.

The three most important functions are *read_cube*, *read_template*, and *fit_spectra*. Details concerning their usage can be found below. Please also see *Advanced Example* program for a more complete demonstration.

2.2.1 read_cube

Example call signature,

```
read_cube('eis_20190404_131513.data.h5', 195.119)
```

read_cube typically requires two arguments:

1. **filename** (str or `Path`)

   Name or path of either the data or head HDF5 file for a single EIS observation

2. **window** (int or float, optional)

   Requested spectral window number or the value of any wavelength within the requested window. Default is '0'

[1] Markwardt, C. B. 2009, in Astronomical Society of the Pacific Conference Series, Vol. 411, Astronomical Data Analysis Software and Systems XVIII, ed. D. A. Bohlender, D. Durand, & P. Dowler, 251

The return value is an *EISCube* class instance which contains calibrated intensities, corrected wavelengths, and all of the associated metadata. `EISCube` objects are a subclass of `NDCube` from the Sunpy-affiliated package of the same name.

You can slice an `NDCube` object using either array indices or by using physical coordinates and the `ndcube.NDCube.crop` method. Please see the NDCube documentation for more details.

The *EISCube* subclass extends `NDCube` and provides additional features. First, an extra `.wavelength` attribute has been added which contains a 3D array with the corrected wavelength values at all locations within the cube. Slicing an `EISCube` will also appropriately slice the wavelength array. Secondly, there are a few extra methods for your convenience, such as `.sum_spectra()` which returns a new, 2D `NDCube` containing the sum along the wavelength axis.

A few more notes about `EISCube` objects,

- Data axes are in the same order as the array stored in the HDF5 file. Namely, the order is (slit_pixel, raster_step, wavelength), i.e. (solar-y, solar-x, wavelength).
- All of the EIS metadata and header information are stored as separate keys in the `.meta` dictionary attribute (e.g. the original FITS header is in `.meta['index']` while the updated pointing information is in `.meta['pointing']`).

2.2.2 read_template

Example call signature,

```
read_template('fe_12_195_119.2c.template.h5')
```

read_template is relatively simple. It takes a single argument giving the filename of a template file and returns an *EISFitTemplate* class instance containing the the initial fit parameters. Users may view the parameter values by using either the `print_parinfo()` method or manually inspecting the `.template` attribute. For convenience, there is also a `central_wave` attribute that contains the mean wavelength value within the template. This can be useful for loading the correct spectral window using *read_cube*.

2.2.3 fit_spectra

This is the main fitting routine. Example call signature,

```
fit_spectra(data_cube, fit_template)
```

The simplest way to use *fit_spectra()* is to give it two arguments:

1. **data_cube** (EISCube object or filepath)

   An EISCube class instance (or slice) containing one or more intensity profiles to be fit. Wavelength and error values will be extracted as needed from the EISCube.

2. **template** (EISFitTemplate object or filepath)

   An EISFitTemplate class instance containing both the general template information (in the `.template` attribute) and a `.parinfo` attribute with the fit parameter dictionary.

Alternatively, you may provide the function with individual data arrays like this,

```
fit_spectra(inten_arr, template_dict, parinfo=parinfo_dict,
            wave=wavelength_arr, errs=error_arr)
```

The function will loop over the data according to its dimensionality. 3D data is assumed to be a full EIS raster (or a sub region), 2D data is assumed to be a single EIS slit, and 1D data is assumed to be a single profile.

fit_spectra returns a *EISFitResult* class instance containing the fit parameter values. As well as assorted metadata, there are two important class methods that may be of use `.get_params()` and `.get_fit_profile()`.

The method `.get_params()` extracts parameters values by either component number, name, or pixel coordinate (or any combination of the three). The arguments are:

- **component** (int or None, optional)

  Integer number (or list of ints) of the functional component(s). If set to None, will return the total combined fit profile. Default is None.

- **param_name** (str, optional)

  String name of the requested parameter. If set to None, will not filter based on paramater name. Default is None.

- **coords** (list or tuple, optional)

  (Y, X) coordinates of the requested datapoint. If set to None, will instead return the parameters at all locations. Default is None

- **num_wavelengths** (int, optional)

  Number of wavelength values to compute the fit intensity at. These values will be equally spaced and span the entire fit window. If set to None, will use the observed wavelength values. Default is None.

- **casefold** (bool, optional)

  If set to True, will ignore case when extracting parameters by name. Default is False.

Examples,

```
c0_params = fit_res.get_fit_profile(component=0)
widths = fit_res.get_fit_profile(param_name='width')
```

The `.get_fit_profile()` method may be used to generate the either the combined fit intensity profile or the profile of a single component function. The method takes up to three arguments:

- **component** (int or None, optional)

  Integer number (or list of ints) of the functional component(s). If set to None, will return the total combined fit profile. Default is None.

- **coords** (list or tuple, optional)

  (Y, X) coordinates of the requested datapoint. If set to None, will instead return the parameters at all locations. Default is None

- **num_wavelengths** (int, optional)

  Number of wavelength values to compute the fit intensity at. These values will be equally spaced and span the entire fit window. If set to None, will use the observed wavelength values. Default is None.

`get_fit_profile()` returns two arrays, `fit_wave` & `fit_inten`, which contain the wavelengths and corresponding fit intensity values.

Examples,

```
fit_x, fit_y = fit_res.get_fit_profile(coords=(5,5))
c0_fit_x, c0_fit_y = fit_res.get_fit_profile(component=0, num_wavelengths=100)
```

2.3 EIS Data Overview

The EUV Imaging Spectrometer — EIS — was designed to study the solar atmosphere and answer fundamental questions on the heating of the solar corona, the origin of the solar wind, and the release of energy in solar flares[1]. EIS observes two wavelength bands in the extreme ultraviolet, 171–-212Å and 245–-291Å, with a spectral resolution of about 22mÅ and a plate scale of 1 per pixel. However, it is not practical to observe the entirety of these bands at all times; therefore EIS observations are typically collected in discrete spectral "windows" containing a smaller wavelength range centered on select lines of interest. These windows can vary depending on the particular study and a single EIS observation can contain up to 25 windows.

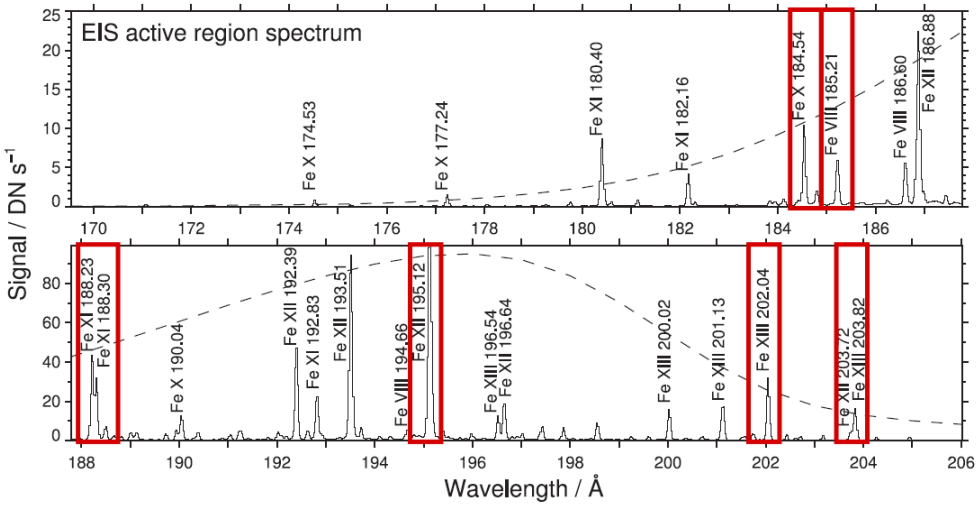


Fig. 2.1: Example EIS spectrum of an active region on 2006 November 4 observed in the 171–-212Å EUV band. Red boxes demonstrate the bounds of some common spectral windows used by observation planners. The dashed line shows the effective area of EIS, which determines the sensitivity of the instrument to specific wavelengths. Adapted from Figure 2 of Young et al., 2007[2]

EIS has two basic modes of operation, "scan" and "sit-and-stare", which comprise ~89% and 11% of all observations respectively. In scan mode, solar images are made by stepping the slit over a region of the Sun and taking an exposure at each position. Sit-and-stare observations, on the other hand, involve taking one or more long exposures of a single location. The EIS slit is always oriented along the Solar-Y (north-south) axis. As a consequence, scan positions are always stepped along the Solar-X (east-west) axis, starting at the edge of the observed region nearest the solar west limb and proceeding towards the east limb.

A detailed description of EIS is given in the instrument paper[3]. At the beginning of the Hinode mission, the strategy was to release unprocessed level-0 FITS files and software routines written in IDL for processing these files into a format that could be used for data analysis. Additionally, all of the routines for computing ancillary information, such as the offsets of the detectors or the magnitude of the instrumental broadening, were all written in IDL. Unfortunately, IDL is an expensive, proprietary language, little used outside of solar physics. Python, in contrast, is a free, open source language that has grown dramatically in popularity since the launch of Hinode, making it an obvious choice for future software development.

[1] EIS is part of the Hinode mission and was sponsored by the Japan Aerospace Exploration Agency (JAXA), the United Kingdom Space Agency (UKSA), and National Aeronautics and Space Administration (NASA) with contributions from ESA and Norway. Hinode was launched on September 22, 2006 at 21:36 UTC from the Uchinoura Space Center in Japan and continues to operate.

[2] Young, P. R., Del Zanna, G., Mason, H. E., et al. 2007, *Publ. Astron. Soc. Japan*, **59**, S857

[3] Culhane, J. L., Harra, L. K., James, A. M., et al. 2007, *Sol. Phys.*, **243**, 19

2.3.1 EIS Level-1 HDF5 Files

To accelerate the transition to Python we have created a new level-1 product that contains both the processed level-1 data and all ancillary information needed for data analysis. The alternative approach, to port all of the existing IDL software to Python, would be time consuming and create confusion about which routines are being actively supported during the transition. Distributing level-1 files removes this problem, but does make the user dependent on the team for reformatting all of the files as bugs are discovered. Since the mission has been going on for some time now, the number of bugs is likely to be small.

There are several other design decisions that merit some explanation

- The data and header information are stored in separate files. Since the data is large and unlikely to change, the time-consuming download of these files should only need to be done once. The header file is very small and can be updated easily.
- HDF5 is used to store the data. This is a very widely used, high-performance file format that is well supported by both IDL and Python. The most attractive feature for this application is that data is stored in a self-documenting, directory-like tree structure instead of binary table extensions.
- The data is processed from raw "data numbers" to "photon events" or "counts". The default behavior of `eis_prep` is to convert to calibrated units. With the HDF5 files, conversion to absolute units is done using a calibration curve in the header file, and several different calibration curves can be considered.

The processed level-1 HDF5 files can be downloaded either directly from the NRL Hinode/EIS website at https://eis.nrl.navy.mil/ or using the tools included in EISPAC. The chapter on *Level-1 HDF5 File Processing* describes how the files were processed.

Note: The HDF5 data archive currently only contains the science observations from the 1 & 2 slit positions. Engineering, calibration, and 40 & 266 slot data are therefore *not* available as HDF5 files.

Footnotes and Citations

2.4 Command Line Scripts

The command line scripts should be automatically installed and registered with your OS as part of installing EISPAC. These scripts are designed to help users quickly search, download, and fit Gaussian functions to the data, all without needing to write a full Python program. To use a script, simply enter its name in the command line from any directory in which you have read and write privileges.

Tip: Some scripts will default to saving files to your current working directory, therefore we recommend running the scripts from the directory in which you intend to do most of your analysis.

There are currently five command line scripts available,

- `eis_catalog` - GUI tool from searching the as-run EIS data catalog and downloading the HDF5 files your computer. Can also generate a text list of files to download.
- `eis_browse_templates` - GUI tool for browsing the fit templates corresponding to each spectral window in a given EIS observation and copying the template files from EISPAC to your current working directory (fit templates are explained more in the *Fitting the Data* chapter).
- `eis_download_files` - Command line tool for downloading a the level-1 HDF5 files associated with one or more level-0 EIS fits files. Can also download an entire list of files using the text output of `eis_catalog`. Example usage,

```
>>> eis_downdload_files eis_l0_20190404_131513.fits
```

- `eis_fit_files` - Command line tool for fitting all of the HDF5 files in a given directory with each fit template found in another directory. Example usage,

```
>>> eis_fit_files ./eis_study/ ./eis_study/templates/
```

- `eis_plot_fit` - Command line tool for a quick-look plot of fit line intensity, velocity, and width. Example usage,

```
>>> eis_plot_fit eis_20190404_131513.fe_12_195_119.2c-0.fit.h5'
```

Tip: `eis_fit_files` and `eis_plot_fit` can also be run without any filename arguments. In such cases, the script will attempt to execute its function using all relevant files in your current working directory.

2.4.1 eis_catalog

The `eis_catalog` GUI tool can be used to search the EIS as-run catalog and download HDF5 files to your local file system. To launch it, simply type `eis_catalog` in the command line of the environment where you installed EISPAC. If you have a working IDL installation of SolarSoft, the tool will attempt to locate a copy of the `eis_cat.sqlite` database on your computer. Otherwise, the program will try to download the database the first time you start the GUI. Please be patient, downloading the full catalog can take a minute of two.

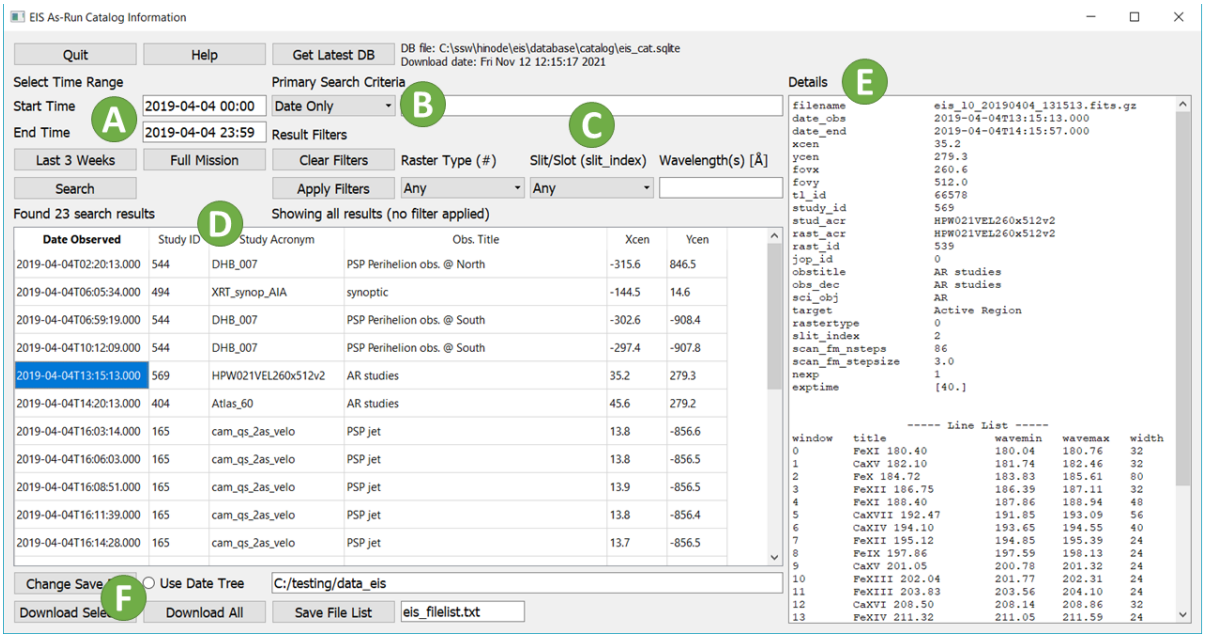


Fig. 2.2: The main window of the `eis_catalog` GUI tool. Please see the text for details about each of the labeled sections.

Above is an image of the main `eis_catalog` window. Some notes on the various sections of the interface.

A. **Time Range**: ISO format (YYYY-MM-DD HH:MM) is recommended but not required. If no end time is given, the program will search the 24 hour time period following the start time.

B. **Primary Search Criteria**: Select an option from the dropdown list and enter your search value in the adjacent box. Current options include "Date only", "Study ID", "Study Acronym", "HOP ID", "Target", "Sci. Obj.", and "Obs. Title". Click the "help" button in the GUI to get and short description of each option. Please be advised, "Date only" searches over long time periods can take a while to process.

C. **Result Filters**: Filter the search results by raster type, slit/slot position, and observed wavelengths (in units of Å). Multiple wavelengths can be given as a comma separated list. Note: applying or removing a filter does not re-run the search, however it may take a moment to update the list if there are a lot of results.

D. **Search Results**: List of all EIS observations matching your search criteria and filters. Clicking on any row in the list will display additional information about that observation in the "Details" panel.

E. **Details**: General use panel for displaying help information, observation details, and status updates.

F. **Download Controls**: Select output directory, download selected file, download all files in the list (use with care!), or make a text list of the filenames. Progress bars for each file download will be printed to your console, NOT the details pane (this will be redirected better in a future update). If the "Use Date Tree" box is checked, files will be downloaded into subdirectories organized by month and day.

2.4.2 eis_browse_templates

The `eis_browse_templates` GUI tool can be used to view and copy fit templates included with EISPAC. Once the GUI is open, use the "Select Header" button to load an HDF5 header file and see a list of templates available for all spectral windows in the associated data file. Clicking on a template name in the list will display an example plot of the template relative to representative solar spectra (NOT the data in the actual observation). You can then use the "Copy template" button to make a copy of the template file in the output directory. Fit templates are explained more in the *Fitting the Data* chapter.

2.5 Downloading EIS Data

2.5.1 eis_catalog GUI or Web Browser

The easiest way to search for and download the processed HDF5 files is to use the `eis_catalog` GUI tool included with EISPAC. This tool provides an easy interface to the official EIS as-run database and can search based on a wide range of technical criteria (see the *eis_catalog* section for details). Alternatively, you can browse and download files directly from the online archive (https://eis.nrl.navy.mil/) or use the *`download_hdf5_data()`* function (assuming you know the exact name of the file you want).

2.5.2 Using SunPy's Fido Interface

As of November of 2022, there is now a handy client for Sunpy's `Fido` data interface! `Fido` is a unified tool that can query a variety of solar and heliophysics data repositories and return a standardized format of results for downloading. A general guide for using ~sumpy.net.Fido can be found in the `Finding and Downloading Data section of the SunPy User's Guide.

Here is a simple example for initializing and using the EISPAC Fido interface. **Please note:** At this time, only time-based searches of EIS data are available through `Fido`.

```
>>> from sunpy.net import Fido, attrs as a
>>> import eispac.net
>>> from eispac.net.attrs import FileType
>>> results = Fido.search(a.Time('2020-11-09 00:00:00','2020-11-09 01:00:00'),
```

(continues on next page)

(continued from previous page)

```
...                       a.Instrument('EIS'),
...                       a.Physobs.intensity,
...                       a.Source('Hinode'),
...                       a.Provider('NRL'),
...                       a.Level('1'))
>>> results
<sunpy.net.fido_factory.UnifiedResponse object at ...>
Results from 1 Provider:

3 Results from the EISClient:
Source: https://eis.nrl.navy.mil/

       Start Time               End Time        ... Level   FileType
----------------------- ----------------------- ... ----- -----------
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1   HDF5 data
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1 HDF5 header
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1        FITS


>>> results = Fido.search(a.Time('2020-11-09 00:00:00','2020-11-09 01:00:00'),
...                       a.Instrument('EIS'),
...                       a.Physobs.intensity,
...                       a.Source('Hinode'),
...                       a.Provider('NRL'),
...                       a.Level('1'),
...                       FileType('HDF5 header'))
>>> results
<sunpy.net.fido_factory.UnifiedResponse object at ...>
Results from 1 Provider:

1 Results from the EISClient:
Source: https://eis.nrl.navy.mil/

       Start Time               End Time        ... Level   FileType
----------------------- ----------------------- ... ----- -----------
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1 HDF5 header
```

Attention: Some laboratory and institution networks (including VPNs) are known to cause issues with using certain `Fido` data clients. In such cases, your system administrator may be able to provide a solution.

2.6 Exploring EIS data

Once installed, EISPAC can be imported into any Python script or interactive session with a simple `import eispac` statement.

2.6.1 Reading Data into Python

Assuming you have already downloaded some data, the following code snippet below illustrates how to how to read the level-1 data from the spectral window containing the 195.12Å line (window 7, in our example file). At the end of the chapter we will show how to examine a data header file to determine what wavelengths are available in a given observation.

```
>>> import eispac
>>> data_filename = 'eis_20190404_131513.data.h5'
>>> data_cube = eispac.read_cube(data_filename, 195.119)
```

The *read_cube()* function will read and apply all of the calibration and pointing corrections necessary for scientific analysis. The function has three key arguments (see the full documentation for additional keywords for specifying your own calibration curve or a count offset):

- **filename** (str or pathlib path) - Name or path of either the data or head HDF5 file for a single EIS observation
- **window** (int or float, optional) - Requested spectral window number (if <= 24) or the value of any wavelength within the requested window (in units of [Angstrom]). Default is "0"
- **apply_radcal** (bool, optional) - If set to True, will apply the pre-flight radiometric calibration curve found in the HDF5 header file and set units to $erg/(cm^2 ssr)$. If set to False, will simply return the data in units of photon counts. Default is True.

2.6.2 EISCube Objects

The return value of *read_cube* is an *EISCube* class instance which contains calibrated intensities (or photon counts), corrected wavelengths, and all of the associated metadata. *EISCube* objects are a subclass of `NDCube` (from the Sunpy-affiliated package of the same name) and, as such, have built-in slicing and coordinate conversion capabilities as virtue of having an associated World Coordinate System (WCS) object.

The *EISCube* subclass extends `ndcube` by including a few additional features. First, an extra `.wavelength` attribute has been added which contains a 3D array with the corrected wavelength values at all locations within the cube. This correction accounts for two effects (1) a systematic spectral shift caused by a tilt in the orientation of the the EIS slit relative to the CCD and (2) a variable spectral shift caused by the orbit of the spacecraft. Secondly, four methods are included to quickly perform common EIS image manipulations,

- The *apply_radcal* and *remove_radcal* methods can be used to convert the data and uncertainty values to and from intensity and photon count units using either the pre-flight radiometric calibration curve provided in the HDF5 header file (default) or a user-inputted array. If *apply_radcal* is given a custom calibration curve, the number of data points in the input array must match the number of data points along the wavelength axis.
- The *sum_spectra* method sums the data along the wavelength axis and returns a new, 2D `NDCube` with just the data (no uncertainty or wavelength information). It requires no arguments.
- The *smooth_cube* method applies a boxcar moving average to the data along one or more spatial axes. It requires a single argument, "width", that must be either a singular value or list of ints, floats, or `astropy.units.Quantity` instances specifying the number of pixels or angular distance to smooth over. If given a single value, only the y-axis will be smoothed. Floats and angular distances will be converted to the nearest whole pixel value.

If a width value is even, width + 1 will be used instead. *smooth_cube* also accepts any number of optional keyword arguments that will be passed to the astropy.convolution.convolve function, which does the actual smoothing operation.

The calibrated intensity and uncertainty values are stored in numpy arrays in the .data and .uncertainty attributes. The order of the axes are (slit position, raster step, dispersion) which correspond to the physical axes of (Solar-Y, Solar-X, Wavelength). You can inspect the dimensions of an NDCube object like so,

```
>>> data_cube.dimensions
[512, 87, 24] pix
```

As you can see, our example data has dimensions of (512, 87, 24). That is, 512 pixels along the slit (in the Solar-Y direction), 87 raster steps (along the Solar-X axis), and 24 pixels along the dispersion (wavelength) axis.

2.6.3 Slicing an EISCube

One of the most powerful features of NDCube-based objects is the ability to slice using either array indices or world coordinates. Slicing by array indices works just like slicing a normal numpy array. Slicing by coordinates is done using the ndcube.NDCube.crop method and a collection of two or more points constructed using high level coordinate objects from astropy.coordinates. Note: for spatial coordinates, you must specify the observer frame, which can be constructed using the EISCube.wcs object.

For example,

```
>>> import astropy.units as u
>>> from astropy.coordinates import SkyCoord, SpectralCoord
>>> from astropy.wcs.utils import wcs_to_celestial_frame
>>> eis_frame = wcs_to_celestial_frame(data_cube.wcs)
>>> lower_left = [SpectralCoord(195.0, unit=u.angstrom),
...               SkyCoord(Tx=48, Ty=225, unit=u.arcsec, frame=eis_frame)]
>>> upper_right = [SpectralCoord(195.3, unit=u.AA),
...                SkyCoord(Tx=165, Ty=378, unit=u.arcsec, frame=eis_frame)]
>>> data_cutout = data_cube.crop(lower_left, upper_right)
>>> data_cutout.dimensions
[154, 48, 14] pix
```

Slicing an *EISCube* also automatically slices all of the associated subarrays (data, uncertainty, wcs, and wavelength). Please see the ndcube documentation for more information about slicing and manipulating NDCube objects.

Attention: Slicing the wavelength axis with .crop() uses the wavelength values in the WCS object, NOT the corrected values stored in .wavelength. Please use with care. If you don't want or need to slice along the wavelength axis, simply give the function a value of None instead of a SpectralCoord object.

2.6.4 Exploring Metadata

All metadata and information from the HDF5 header file are packed into a single dictionary stored in the `.meta` attribute of the *EISCube*. The structure of the `.meta` dictionary mirrors the internal structure of the HDF5 file, with a few extra keys added for convenience. You can explore the contents with the usual Python commands,

```
>>> data_cube.meta.keys()
dict_keys(['filename_data', 'filename_head', 'wininfo', 'iwin', 'iwin_str',
           'index', 'pointing', 'wave', 'radcal', 'slit_width',
           'slit_width_units', 'ccd_offset', 'wave_corr', 'wave_corr_t',
           'wave_corr_tilt', 'date_obs', 'date_obs_format', 'duration',
           'duration_units', 'mod_index', 'aspect', 'aspect_ratio', 'notes'])
>>> data_cube.meta['pointing']['x_scale']
2.9952
>>> data_cube.meta['radcal']
array([8.06751  , 8.060929 , 8.054517 , 8.048271 , 8.042198 , 8.036295 ,
       8.030562 , 8.024157 , 8.017491 , 8.010971 , 8.0046015, 7.998385 ,
       7.9923196, 7.9864078, 7.980654 , 7.975055 , 7.969617 , 7.9643393,
       7.959224 , 7.9542727, 7.949487 , 7.9448686, 7.9404206, 7.9361415],
      dtype=float32)
```

Assorted metadata

For completeness and transparency, we have bundled all of the commonly used instrumental and processing metadata provided in IDL. EISPAC attempts to correct for most instrumental effects, so you will not need to reference the `ccd_offset` or `wave_corr_tilt`, for example, but that information is there if you want it. As a reminder, the `slit_width` array gives the empirically-determined instrumental broadening along the EIS slit, and not the physical width of the slit itself.

Here `x_scale` is the number of arcsec between step positions in the raster. Most EIS rasters take more than 1 arcsec per step, which degrades the spatial resolution but increases the cadence. The variable `radcal` is the pre-flight calibration curve for this data window. It includes all of the factors for converting counts directly to $erg/(cm^2 ssr)$.

Of particular note, the `.meta['index']` dictionary contains the original EIS Level-0 FIT header keywords. Be aware, the pointing information in the Level-0 header is uncorrected. The `.meta['mod_index']` (modified index) dictionary contains a reduced set of header keywords *including* all pointing corrections. Additionally, the `mod_index` values are updated by EISPAC whenever the *EISCube* is sliced while the original index is not updated.

2.6.5 Spectral Windows

We usually don't care about the numbering of the data windows. It's more natural to want to read the data corresponding to a particular wavelength. The *read_wininfo* function can be used help identify the spectral contents of each data window. The function takes an input header file and returns a `numpy.recarray` containing the window numbers, min and max wavelengths and primary spectral line for each data window. Note: for your convenience, a copy of the `wininfo` array is also stored in the `EISCube.meta` dictionary.

```
>>> import eispac
>>> header_filename = 'eis_20190404_131513.head.h5'
>>> wininfo = eispac.read_wininfo(header_filename)
>>> wininfo.dtype.names
('iwin', 'line_id', 'wvl_min', 'wvl_max', 'nl', 'xs')
>>> wininfo[0:4]
```

(continues on next page)

(continued from previous page)

```
rec.array([(0, 'Fe XI 180.400', 180.03426, 180.72559, 32, 661),
           (1, 'Ca XV 182.100', 181.75139, 182.44266, 32, 738),
           (2, 'Fe X 184.720', 183.82512, 185.5865 , 80, 831),
           (3, 'Fe XII 186.750', 186.3891 , 187.0802 , 32, 946)],
           ... ... ...
           (23, 'Mg VII 280.390', 279.7766 , 280.9996 , 56, 3720),
           (24, 'Fe XV 284.160', 283.89   , 284.40134, 24, 3905)],
          dtype=[('iwin', '<i4'), ('line_id', '<U64'), ('wvl_min', '<f4'),
                 ('wvl_max', '<f4'), ('nl', '<i4'), ('xs', '<i4')])
```

We can then use a `numpy.where` call on the `wininfo` array to map wavelength to window number. Users familiar with IDL may be interested to note that numpy record arrays can be accessed like an IDL array of structures (e.g. instead of `wininfo['wvl_min']` below, you could also use `wininfo.wvl_min`).

```
>>> import numpy as np
>>> wvl = 195.119
>>> p = (wininfo['wvl_max'] - wvl)*(wvl - wininfo['wvl_min'])
>>> iwin = np.where(p >= 0)[0]
>>> iwin
array([7], dtype=int64)
```

If the result is an empty array, the wavelength is not in the data.

2.6.6 Plotting

We can make a quick image of the EIS data by making use of the `.plot()` method provided in all `NDCube` objects (note, it usually helps to sum along the dispersion direction first).

```
>>> data_cube.sum_spectra().plot(aspect=data_cube.meta['aspect'])
```

The `.plot()` method can also be used to display the spectrum from a single pixel, as shown below. For illustration, we also convert the data back in units of photon counts (this is the same as dividing the calibrated data by the `.meta['radcal']` array).

```
>>> ix = 48
>>> iy = 326
>>> spec = data_cube[iy,ix,:].remove_radcal()
>>> spec_plot = spec.plot()
>>> spec_plot.set_title(f'ix = {ix}, iy = {iy}, units = counts')
```

To perform more advanced plotting, such as logarithmically scaling the intensities, you will need to extract the data from the *EISCube* and create the figure yourself using any of the various Python plotting libraries. For example,

```
import numpy as np
import matplotlib.pyplot as plt
import eispac

data_filename = 'eis_20190404_131513.data.h5'
data_cube = eispac.read_cube(data_filename, 195.119)
raster_sum = np.sum(data_cube.data, axis=2) # or data_cube.sum_spectra().data
scaled_img = np.log10(raster_sum)
```

(continues on next page)

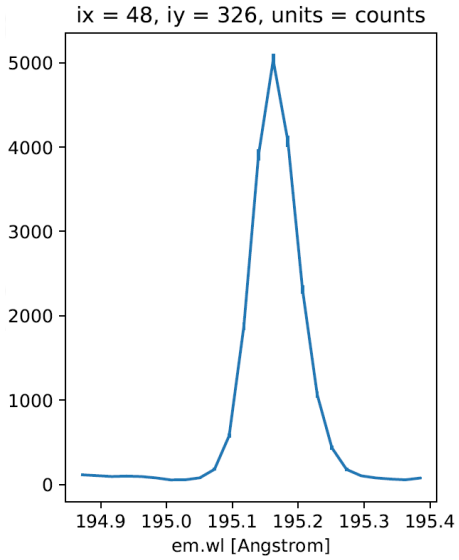


Fig. 2.3: An example Fe XII 195.119 Å line profile from the raster.

(continued from previous page)

```
plt.figure()
plt.imshow(scaled_img, origin='lower', extent=data_cube.meta['extent_arcsec'], cmap='gray
↪')
plt.title(data_cube.meta['date_obs'][-1])
plt.xlabel('Solar-X [arcsec]')
plt.ylabel('Solar-Y [arcsec]')
plt.show()
```

Tip: Setting both "aspect" (y_scale/x_scale) and "extent" (data range as [left, right, bottom, top]) in `plt.imshow()` can sometimes give unexpected results. You may need to experiment with the combination of keywords needed to get the plot you expect.

2.7 Fitting the Data

Why use MPFIT?

We have chosen to use MPFIT due to its flexibility and long (20+ year) legacy in solar and heliophysics. While newer fitting methods provide powerful tools for data exploration, they are still young and incur significant additional computation time (and are also more complicated to parallelize).

Fitting of the spectra involves selecting a spectral line of interest (e.g. 195.119Å) from one of the spectral windows of in the data, choosing a function (or combination of functions) to fit, and determining an initial guess for each parameter. The next ingredient for a fit is the selection of an optimization method. By default, EISPAC uses a Python implementation of the well-known IDL method **MPFIT**, which solves the non-linear least squares problem using the Levenberg- Marquardt algorithm. The Python module, `mpfit.py`, can be found on GitHub[1] and is included in EISPAC.

[1] Sergey Koposov's Python port of MPFIT can be found at https://github.com/segasai/astrolibpy/

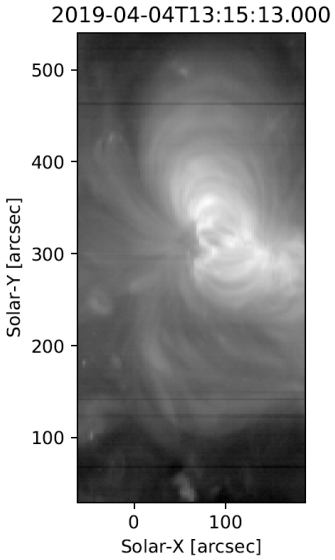


Fig. 2.4: An example image formed by summing the data for the Fe XII spectral window in the dispersion direction. In a subsequent chapter we'll discuss fitting the spectra.

Future versions of the code will include full support for other fitting packages such as the newer `astropy.modeling` framework.

2.7.1 Template Files

In order to make fitting quick and easy, we've created a set of fit templates for different spectral lines. Each template consists of one or more component Gaussian functions and a constant background term. Template files are named according to the following pattern: `{primary spectral line}.{number of Gaussians}c.template.h5`. 119 templates are provided with EISPAC, covering all of the commonly observed spectral line combinations.

The easiest way to browse and copy fit template files is to use the `eis_browse_templates` GUI tool included with EISPAC. This tool can be used to display a list of all templates available for each spectral window in an EIS observation and copy the template files to your work directory (see the *eis_browse_templates* section for a few tips). A similar result can be obtained with the *`match_templates`* function, which takes either an *`EISCube`* or the path to a header file and outputs a list (or list of lists) of all fit templates available for a given spectral window (or data file).

```
>>> import eispac
>>> data_filename = 'eis_20190404_131513.data.h5'
>>> data_cube = eispac.read_cube(data_filename, 195.119)
>>> template_list = eispac.match_templates(data_cube)
>>> for file in template_list:
>>>     print(file.name)
fe_12_195_119.1c.template.h5
fe_12_195_119.2c.template.h5
fe_12_195_179.2c.template.h5
```

In the example above, notice how both one and two component fits are available for the same line. Furthermore, as with many multi-component templates, the third template file is actually a duplicate of the second template, just with the name of the secondary spectral line in the filename. This was designed to make it easier to see exactly which spectral lines can be fit. However, it also means you cannot simply loop over all template files available for a given spectral window, as you would be unnecessarily multiplying the amount of fitting performed (and resultant output files).

We recommend copying the template files you wish to use to your project directory so you have a more self-contained record of your fitting process. Assuming you are running Python from your project directory, the following code snippet illustrates one method you could use.

```
>>> import os
>>> import shutil
>>> current_dir = os.getcwd()
>>> shutil.copy(template_list[1], current_dir)
```

Attention: Take care to only *copy* template files. Moving a template from your EISPAC installation will lead to incomplete results in future calls to `match_templates()`.

2.7.2 Loading Fit Templates

An `h5dump`[2] on one of the template files shows that it contains a `/template` group for the initial guess on the fit parameters and a `/parinfo` group containing constraints on the parameters for use by *mpfit*.

```
h5dump -n fe_12_195_119.2c.template.h5
HDF5 "fe_12_195_119.2c.template.h5" {
FILE_CONTENTS {
 group      /
 group      /parinfo
 dataset    /parinfo/fixed
 dataset    /parinfo/limited
 dataset    /parinfo/limits
 dataset    /parinfo/tied
 dataset    /parinfo/value
 group      /template
 dataset    /template/component
 dataset    /template/data_e
 dataset    /template/data_x
 dataset    /template/data_y
 dataset    /template/fit
 dataset    /template/fit_back
 dataset    /template/fit_gauss
 dataset    /template/line_ids
 dataset    /template/n_gauss
 dataset    /template/n_poly
 dataset    /template/order
 dataset    /template/wmax
 dataset    /template/wmin
 }
```

The *read_template* function can be used to read a template file and examine the contents.

```
>>> import eispac
>>> tmplt_filename = 'fe_12_195_119.2c.template.h5'
>>> tmplt = eispac.read_template(tmplt_filename)
```

[2] `h5dump` is a command line tool used to inspect the contents of an HDF5 file. It is included the Anaconda Python distribution platform, but can also be installed on its own.

Use the command `print(TEMPLATE)` to view the initial parameter values and constraints in a nice format.

```
>>> print(tmplt)
--- FIT TEMPLATE PARAMETER CONSTRAINTS ---
 *         Value      Fixed        Limited            Limits                Tied
p[0]     57514.6647     0          1    0       0.0000       0.0000
p[1]       195.1179     0          1    1     195.0778     195.1581
p[2]         0.0289     0          1    1       0.0191       0.0510
p[3]      8013.4013     0          1    0       0.0000       0.0000
p[4]       195.1779     0          1    1     195.1378     195.2181          p[1]+0.06
p[5]         0.0289     0          1    1       0.0191       0.0510          p[2]
p[6]       664.3349     0          0    0       0.0000       0.0000
```

funcinfo dictionary

The `EISFitTemplate` object returned by `eispac.read_template()` also generates a `.funcinfo` list. This list will help with the implementation of other fitting methods in the future, but is currently not used by the code and can, thereby, be safely ignored.

The structure of `parinfo` is specific to MPFIT and should be familiar to anyone who has used the original IDL version; please see the section on *Constraining Parameter Values* for more details. The templates provided with EISPAC consist of one or more Gaussian functions (with parameters in the order of peak, centroid, & width) followed by one or more background polynomial terms (usually just a single, constant value). The values `.template['n_gauss']` and `.template['n_poly']` indicate, respectively, the number of Gaussian functions and background polynomial terms in a given template.

Note: The multigaussian function is composed of generalized Gaussian functions of the form $f(x) = Aexp(-(x - b)^2/2c^2)$, where A is the amplitude (peak value), b is the position of the center of the peak (centroid), and c is the standard deviation (width). This is consistent with the fit parameters used for EIS data in the IDL SolarSoftWare (SSW) analysis suite.

2.7.3 Fitting Spectra

Once you've read in a template file, you can use the central wavelength to find the desired spectral window in the data using *read_cube*.

```
>>> data_filename = 'eis_20190404_131513.data.h5'
>>> data_cube = eispac.read_cube(data_filename, tmplt.central_wave)
```

As mentioned in the previous chapter, *read_cube* automatically applies all of the pointing and wavelength corrections, bad data masking, and error estimations needed for scientific analysis. By default, the code also converts the data from photon counts to intensity units of erg cm^{-2} s^{-1} sr^{-1} using the appropriate pre-flight calibration curve. This conversion can be disabled by setting the keyword `apply_radcal=False`, should you prefer to run your fits in count space.

Summary of fitting process

Here's what's happening under the hood, `fit_spectra()` calls the helper function `scale_guess()` to scale the initial parameter values to the data, then `mpfit` is called to actually run the Levenberg-Marquardt fitting on a custom function that computes the deviates between the input spectrum and a multigaussian fit. If `ncpu` is set to a value >

1, then each raster step position will be processed separately and the full set of results will be combined into a single output.

On to the fitting! Now that you have a template and the data elements, you can perform a fit of the entire data cube by calling the top-level fitting routine, *fit_spectra*. The easiest way to use *fit_spectra* is to just give it both an *EISCube* and *EISFitTemplate* object (or filepaths to the data and template HDF5 files). You may slice your *EISCube* however you wish before fitting and the code will loop over the data appropriately (this includes fitting a single spectra or slit observation). Additionally, *fit_spectra* takes advantage of the `multiprocessing` package in the Python standard library to automatically parallelize the fitting process and minimize the run time. You may control the number of processing cores used for the fitting with `ncpu` keyword, or set it equal to "max" or `None` to use the maximum number of cores available. Please see the full doc string for *fit_spectra* for additional options and parameters.

Attention: Due to the specifics of how the multiprocessing library works, any statements that call `fit_spectra()` using ncpu > 1 MUST be wrapped in a `"if __name__ == __main__:"` statement in the top-level script or program. If such a "name guard" statement is not detected, `fit_spectra()` will fall back to using a single process. Unfortunately, this means you can not directly use parallel fitting from an interactive Python shell, you must first write a program that you save and run.

Here is a minimal example program that just loads and fits the data.

```
import eispac

if __name__ == '__main__':
    # input data and template files
    data_filepath = './eis_20190404_131513.data.h5'
    template_filepath = './fe_12_195_119.2c.template.h5'

    # read fit template
    tmplt = eispac.read_template(template_filepath)

    # Read spectral window into an EISCube
    data_cube = eispac.read_cube(data_filepath, tmplt.central_wave)

    # Fit the data, then save it to disk and test loading it back in
    fit_res = eispac.fit_spectra(data_cube, tmplt, ncpu='max')
    save_filepaths = eispac.save_fit(fit_res, save_dir='cwd')
    FITS_file = eispac.export_fits(fit_res, save_dir='cwd')
    load_fit = eispac.read_fit(save_filepaths[0])
```

Note: The command line script `eis_fit_files` can be used to quickly fit a directory of files using one or more templates in another directory.

2.7.4 EISFitResult Objects

fit_spectra outputs an *EISFitResult* object, which may be saved to an HDF5 file and read back in later using the *save_fit* and *read_fit* functions (as shown in the example above). The output fit parameters are stored in a dictionary of arrays.

```
>>> for key in fit_res.fit.keys():
...     print(f"{key:<15} {fit_res.fit[key].dtype} {fit_res.fit[key].shape}")

line_ids        <U14 (2,)
main_component  int16 ()
n_gauss         int16 ()
n_poly          int16 ()
wave_range      float64 (2,)
status          float64 (128, 32)
chi2            float64 (128, 32)
mask            int32 (128, 32, 24)
wavelength      float64 (128, 32, 24)
int             float64 (128, 32, 2)
err_int         float64 (128, 32, 2)
vel             float64 (128, 32, 2)
err_vel         float64 (128, 32, 2)
params          float64 (128, 32, 7)
perror          float64 (128, 32, 7)
component       int32 (7,)
param_names     <U32 (7,)
param_units     <U32 (7,)
```

We can extract an array of the fit parameters or intensity profile using the *get_params* and *get_fit_profile* methods. Both methods take optional keywords for selecting the component number and/or an individual pixel (using array coordinates). *get_fit_profile* also has a `num_wavelengths` keyword that allows us to interpolate the fit profile at a higher wavelength resolution than observed by EIS. The use of these methods are demonstrated in the longer example program below, which also shows one method for finding the indices and coordinates of maximum intensity.

```
import numpy as np
import matplotlib.pyplot as plt
import astropy.units as u
from astropy.coordinates import SkyCoord
from astropy.wcs.utils import wcs_to_celestial_frame
import eispac

if __name__ == '__main__':
    # Read in the fit template and EIS observation
    data_filepath = './eis_20190404_131513.data.h5'
    template_filepath = './fe_12_195_119.2c.template.h5'
    tmplt = eispac.read_template(template_filepath)
    data_cube = eispac.read_cube(data_filepath, tmplt.central_wave)

    # Select a cutout of the raster
    eis_frame = wcs_to_celestial_frame(data_cube.wcs)
    lower_left = [None, SkyCoord(Tx=-25, Ty=225, unit=u.arcsec, frame=eis_frame)]
    upper_right = [None, SkyCoord(Tx=175, Ty=425, unit=u.arcsec, frame=eis_frame)]
    raster_cutout = data_cube.crop(lower_left, upper_right)
```

(continues on next page)

(continued from previous page)

```
    # Fit the data and save it to disk
    fit_res = eispac.fit_spectra(raster_cutout, tmplt, ncpu='max')
    save_filepaths = eispac.save_fit(fit_res, save_dir='cwd')

    # Find indices and world coordinates of max intensity
    sum_data_inten = raster_cutout.sum_spectra().data
    iy, ix = np.unravel_index(sum_data_inten.argmax(), sum_data_inten.shape)
    ex_world_coords = raster_cutout.wcs.array_index_to_world(iy, ix, 0)[1]
    y_arcsec, x_arcsec = ex_world_coords.Ty.value, ex_world_coords.Tx.value

    # Extract data profile and interpolate fit at higher spectral resolution
    data_x = raster_cutout.wavelength[iy, ix, :]
    data_y = raster_cutout.data[iy, ix, :]
    data_err = raster_cutout.uncertainty.array[iy, ix, :]
    fit_x, fit_y = fit_res.get_fit_profile(coords=[iy,ix], num_wavelengths=100)
    c0_x, c0_y = fit_res.get_fit_profile(0, coords=[iy,ix], num_wavelengths=100)
    c1_x, c1_y = fit_res.get_fit_profile(1, coords=[iy,ix], num_wavelengths=100)
    c2_x, c2_y = fit_res.get_fit_profile(2, coords=[iy,ix], num_wavelengths=100)

    # Make a multi-panel figure with the cutout and example profile
    fig = plt.figure(figsize=[10,5])
    plot_grid = fig.add_gridspec(nrows=1, ncols=2, wspace=0.3)

    data_subplt = fig.add_subplot(plot_grid[0,0])
    data_subplt.imshow(sum_data_inten, origin='lower', extent=cutout_extent)
    data_subplt.scatter(x_arcsec, y_arcsec, color='r', marker='x')
    data_subplt.set_title('Data Cutout\n'+raster_cutout.meta['mod_index']['date_obs'])
    data_subplt.set_xlabel('Solar-X [arcsec]')
    data_subplt.set_ylabel('Solar-Y [arcsec]')

    profile_subplt = fig.add_subplot(plot_grid[0,1])
    profile_subplt.errorbar(data_x, data_y, yerr=data_err, ls='', marker='o', color='k')
    profile_subplt.plot(fit_x, fit_y, color='b', label='Combined profile')
    profile_subplt.plot(c0_x, c0_y, color='r', label=fit_res.fit['line_ids'][0])
    profile_subplt.plot(c1_x, c1_y, color='r', ls='--', label=fit_res.fit['line_ids'][1])
    profile_subplt.plot(c2_x, c2_y, color='g', label='Background')
    profile_subplt.set_title(f'Cutout indices: iy = {iy}, ix = {ix}')
    profile_subplt.set_xlabel('Wavelength [$\AA$]')
    profile_subplt.set_ylabel('Intensity ['+raster_cutout.unit.to_string()+']')
    profile_subplt.legend(loc='upper left', frameon=False)
    plt.show()
```

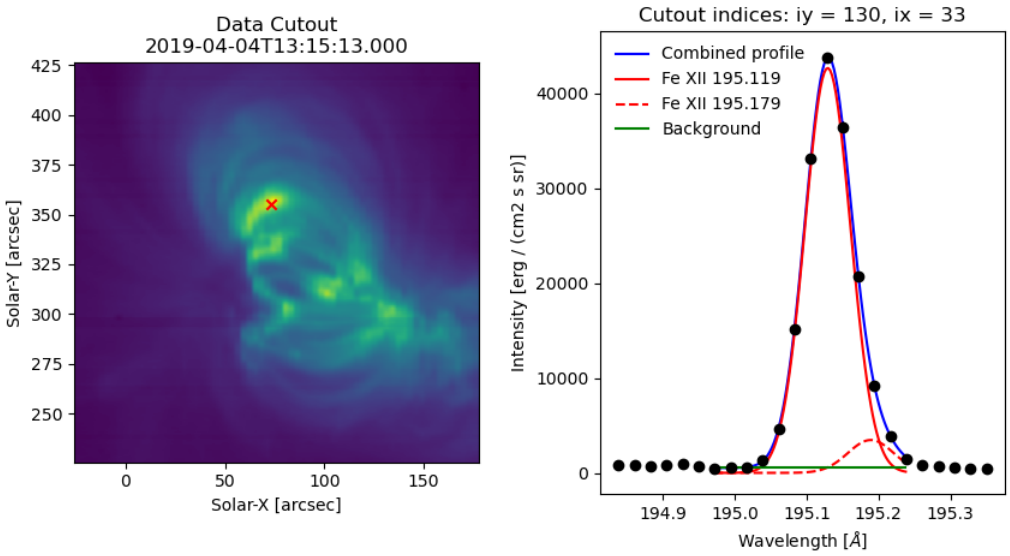


Fig. 2.5: Example data cutout (left) and fit profile (right) for the spectral window containing the Fe XII 195.119 Å line. The red X shows the location of the maximum summed intensity.

2.7.5 EISMaps for Sunpy

The fit line intensities, velocities, and widths can be loaded into an *EISMap*, which is a subclass of `sunpy.map.Map`. This allow us to leverage the full power of Sunpy to do all sorts of cool science like comparing spacecraft locations, co-aligning images, reprojecting maps, and performing field extrapolations (see the Map sections of the SunPy User's Guide and Example Gallery for some demonstrations). You can get an *EISMap* by either using the *get_map* method or saving the measurements to FITS files using *export_fits* and then loading them in with either `eispac.EISMap(FILENAME)` or even `sunpy.map.Map(FILENAME)` (assuming EISPAC is also imported in your program).

For now, we will just show you some examples of the quick-look plots.

```
>>> # Fit intensity (in a nice sunpy Map)
>>> inten_map = fit_res.get_map(component=0, measurement='intensity')
>>> inten_map.peek()
```

```
>>> # Fit velocity map
>>> # Note: You can also use positional arguments and abbreviations
>>> vel_map = fit_res.get_map(0, 'vel')
>>> vel_map.peek()
```

Attention: While we have corrected the velocity maps for orbital effects, there are still some unknown uncertainties. This is largely the case for *ALL* EIS velocity maps, not just those computed by EISPAC. Please use with care.

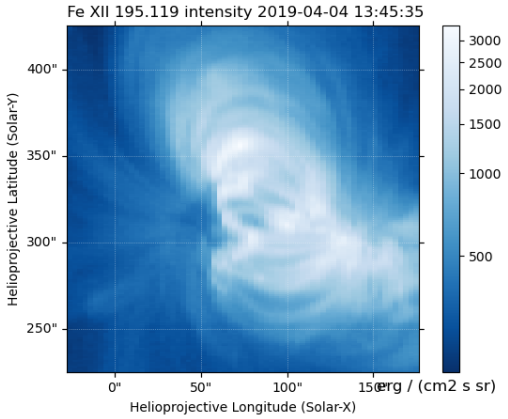


Fig. 2.6: Fit line intensity in a nice SunPy Map.

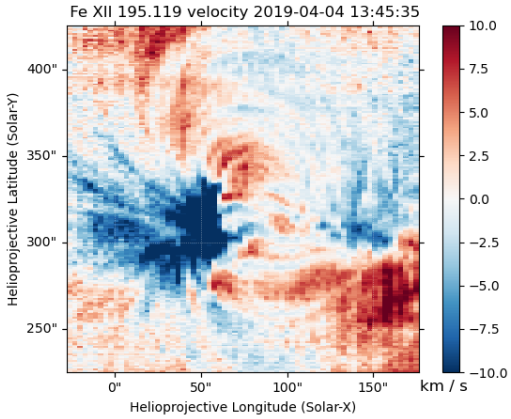


Fig. 2.7: Fit line velocity map.

2.8 Level-1 HDF5 File Processing

This chapter describes in more detail how the EIS level-1 HDF5 files were processed and saved. These HDF5 files can be downloaded from https://eis.nrl.navy.mil/ or by using the search & download functionality of EISPAC (see the section on *eis_catalog GUI or Web Browser*).

2.8.1 Prepping the Data in IDL

The level-0 fits files were prepped using the IDL routine `eis_prep` available via SolarSoft[1] with the following options:

```
default = 1
save = 1
quiet = 1
retain = 1
photons = 1
refill = 0
```

There are 400,000+ EIS level-0 files at present, but on a multi-core machine using the IDL bridge all of the files can be prepped in under 24 hours. We have prepped all of the available EIS files and saved them to standard fits files in the usual way. Some important points:

- *units* - As mentioned previously, the units for the output in these level-1 files is "photon events" or "counts." This means that the statistical uncertainty can usually be estimated as $\sqrt{N}$. When EISPAC loads the HDF5 files, it also calculates an estimate for the read noise contribution. It should be noted, however, that the read noise becomes significant only at very low flux levels (1–2 counts).
- *retain* - Note that the retain keyword preserves negative values. One of the jobs of `eis_prep` is to remove the pedestal from the CCD readout and any time-dependent dark current. Since the spectral windows are generally narrow, the estimate of the background can be too high and the subtracted intensities of the continuum can be negative. This will be dealt with during the fitting.
- *refill* - The warm pixel problem complicates the fitting of EIS line profiles. As discussed in the EIS software note #13 (found in SSW or on the MSSL EIS Wiki), interpolating the values of missing pixels appears to best reproduce the original data. This option is left off during `eis_prep` so that the level-1 fits file preserves the information on the missing pixels. As discussed below, the interpolation (via the refill option) is done during the read and this data is ultimately written to the HDF5 file. A mask indicating which pixels have been interpolated will be added to the HDF5 files in a future revision.

Here is an IDL code snippet related to reading the data by looping over the spectral windows.

```
for iwin=0, nwin-1 do begin
  d = eis_getwindata(eis_level1_filename, iwin, /refill, /quiet)
  eis_level1_data[iwin] = ptr_new(d)
endfor
```

[1] Freeland, S. L., & Handy, B. N. 1998, *Sol. Phys.*, **182**, 497

2.8.2 Writing the HDF5 Files

Each processed level-1 fits file was bundled up with the associated calibration and metadata and saved as a pair of two HDF5 files:

- **eis_YYYYMMDD_HHMMSS.data.h5** - Contains only the corrected photon counts within each spectral window. This is, by far, the larger of the two HDF5 files. However, they should not need to be updated or downloaded very often.
- **eis_YYYYMMDD_HHMMSS.head.h5** - Contains the original fit file index, calibration curves for each spectral window (used to convert counts into intensity values), and the corrected pointing information.

2.8.3 Internal Structure

Users will rarely, if ever, need to access the information inside the HDF5 files directly. EISPAC contains all of the functions needed to read the data and apply the calibration and pointing corrections. Nevertheless, the contents and data structure of the HDF5 files are summarized below.

Data Files (*.data.h5)

- **level1** (group)
  - **intensity_units** (dataset) - String with the level1 data units. This will usually be "counts"
  - **win##** (dataset) - array of floating point values with the photon counts in a given spectral window (e.g. `win00, win01, ... win24`). Note well, each set of EIS observations may have a different number of spectral windows, up to a maximum of 24 windows. Window numbers are numbered sequentially from 00; a given wavelength range may be assigned a different window number in each EIS study.

Header Files (*.head.h5)

- **ccd_offsets** (group)
  - **win##** (dataset) - array of CCD pointing offsets (in units of [arcsec] along the Solar-Y axis) for each wavelength value observed in a given spectral window. Computed in IDL using the function `eis_ccd_offset`. While the offset technically varies with wavelength, the difference within a single window is on the order of 0.05 arcsec. Therefore, the mean CCD offset within a window is commonly used.
- **exposure_times** (group)
  - **duration** (dataset) - array of exposure times for each raster position within the EIS observation
  - **duration_units** (string) - units of exposure times (usually "seconds").
- **index** (group) - complete FITS header from the original level-0 EIS data file.
- **instrumental_broadening** (group)
  - **slit_width** (dataset) - array of widths along the EIS slit, as computed by the IDL function `eis_slit_width`.
  - **slit_width_units** (string) - units of slit width (usually "Angstroms").
- **pointing** (group) - various arrays and reference values needed for correcting and updating the pointing values. Subarrays and values included: fovx, fovy, offset_x, offset_y, ref_time, solar_x, solar_y, x_scale, xcen, y_scale, & ycen.
- **radcal** (group)
  - **win##_pre** (dataset) - Pre-flight radiometric calibration curve for each spectral window in the observation.

- **times** (group)
  - **date_obs** (dataset) - array of starting timestamps for each raster position in the EIS observation.
  - **time_format** (string) - format code for the timestamps ("iso_8601").
- **wavelength** (group)
  - **wave_corr** (dataset) - combined array of wave correction factors due to all orbital and instrumental effects (see below).
  - **wave_corr_t** (dataset) - array of wave correction factors due to the orbital motion and instrument temperature. This is computed using the `hkwavecorr` method in IDL.
  - **wave_corr_tilt** (dataset) - array of wave correction factors due to the tilt EIS slit relative to the orientation of the CCD.
  - **win##** (dataset) - *uncorrected* wavelength arrays for each spectral window in the observation (in units of [Angstrom]).
- **wininfo** (group)
  - **nwin** (integer) - number of spectral windows in the EIS observation
  - **win##** (group) - dictionary of window information for a given spectral window. Values included: iwin, line_id, nl, wvl_max, wvl_min, xs.

The contents of the HDF5 files can be displayed using the `h5dump` command line tool, which is provided along with the Anaconda Python distribution platform or can be installed on its own. Example usage,

```
> h5dump -n eis_20190404_131513.data.h5
FILE_CONTENTS {
group      /
group      /level1
dataset    /level1/intensity_units
dataset    /level1/win00
dataset    /level1/win01
dataset    /level1/win02
dataset    /level1/win03
dataset    /level1/win04
dataset    /level1/win05
. . .
```

The actual data associated with each variable can be printed out using the `-d` option. For example,

```
> h5dump -d exposure_times/duration eis_20190404_131513.head.h5
HDF5 "eis_20190404_131513.head.h5" {
DATASET "exposure_times/duration" {
   DATATYPE  H5T_IEEE_F32LE
   DATASPACE  SIMPLE { ( 87 ) / ( 87 ) }
   DATA {
   (0): 40.0005, 40.0002, 40.0004, 40.0004, 39.9994, 40.0002, 39.9995, 40,
   (8): 40.0007, 39.9999, 40.0005, 40.0004, 39.9997, 40.0002, 39.9994,
   . . .
}
```

Citations

2.9 Acknowledgments

This work was sponsored by NASA's Hinode project. Hinode is a Japanese mission developed and launched by ISAS/JAXA, with NAOJ as domestic partner and NASA and STFC (UK) as international partners.

2.10 MPFIT Documentation

The text below is adapted from the `mpfit.py` documentation and gives a high-level overview of the theory and math behind how the code works. More details concerning input parameters and keywords can be found in the *`mpfit`* doc string.

2.10.1 Description

MPFIT[1] uses the Levenberg-Marquardt technique to solve the least-squares problem. In its typical use, MPFIT will be used to fit a user-supplied function (the "model") to user-supplied data points (the "data") by adjusting a set of parameters. MPFIT is based upon MINPACK-1 (LMDIF.F)[2] by More' and collaborators[3][4].

For example, a researcher may think that a set of observed data points is best modelled with a Gaussian curve. A Gaussian curve is parameterized by its mean, standard deviation and normalization. MPFIT will, within certain constraints, find the set of parameters which best fits the data. The fit is "best" in the least-squares sense; that is, the sum of the weighted squared differences between the model and data is minimized.

The Levenberg-Marquardt technique is a particular strategy for iteratively searching for the best fit. This particular implementation is drawn from MINPACK-1 (see NETLIB), and is much faster and more accurate than the version provided in the Scientific Python package in Scientific.Functions.LeastSquares. This version allows upper and lower bounding constraints to be placed on each parameter, or the parameter can be held fixed.

The user-supplied Python function should return an array of weighted deviations between model and data. In a typical scientific problem the residuals should be weighted so that each deviate has a gaussian sigma of 1.0. If `X` represents values of the independent variable, `Y` represents a measurement for each value of `X`, and `ERR` represents the error in the measurements, then the deviates could be calculated as follows:

```
DEVIATES = (Y - F(X)) / ERR
```

where `F` is the analytical function representing the model. You are recommended to use the convenience functions MPFITFUN and MPFITEXPR, which are driver functions that calculate the deviates for you. If `ERR` are the 1-sigma uncertainties in `Y`, then

```
TOTAL( DEVIATES^2 )
```

will be the total chi-squared value. MPFIT will minimize the chi-square value. The values of `X`, `Y` and `ERR` are passed through MPFIT to the user-supplied function via the `FUNCTKW` keyword.

Simple constraints can be placed on parameter values by using the `PARINFO` keyword to MPFIT. See below for a description of this keyword.

[1] Markwardt, C. B. 2009, in Astronomical Society of the Pacific Conference Series, Vol. 411, Astronomical Data Analysis Software and Systems XVIII, ed. D. A. Bohlender, D. Durand, & P. Dowler, 251

[2] MINPACK-1, Jorge More', available from netlib (www.netlib.org).

[3] "Optimization Software Guide," Jorge More' and Stephen Wright, SIAM, *Frontiers in Applied Mathematics*, Number 14.

[4] More', Jorge J., "The Levenberg-Marquardt Algorithm: Implementation and Theory," in *Numerical Analysis*, ed. Watson, G. A., Lecture Notes in Mathematics 630, Springer-Verlag, 1977.

MPFIT does not perform more general optimization tasks. See TNMIN instead. MPFIT is customized, based on MINPACK-1, to the least-squares minimization problem.

2.10.2 User Function

The user must define a function which returns the appropriate values as specified above. The function should return the weighted deviations between the model and the data. It should also return a status flag and an optional partial derivative array. For applications which use finite-difference derivatives – the default – the user function should be declared in the following way:

```
def myfunct(p, fjac=None, x=None, y=None, err=None)
        # Parameter values are passed in "p"
        # If fjac==None then partial derivatives should not be
        # computed.  It will always be None if MPFIT is called with
        # default flag.
        model = F(x, p)
        # Non-negative status value means MPFIT should continue,
    # negative means stop the calculation.
        status = 0
        return([status, (y-model)/err]
```

See below for applications with analytical derivatives.

The keyword parameters `X`, `Y`, and `ERR` in the example above are suggestive but not required. Any parameters can be passed to MYFUNCT by using the `functkw` keyword to MPFIT. Use MPFITFUN and MPFITEXPR if you need ideas on how to do that. The function *must* accept a parameter list, `P`.

In general there are no restrictions on the number of dimensions in `X`, `Y`, or `ERR`. However the deviates *must* be returned in a one-dimensional Numeric array of type Float.

User functions may also indicate a fatal error condition using the status return described above. If status is set to a number between -15 and -1 then MPFIT will stop the calculation and return to the caller.

2.10.3 Analytic Derivatives

In the search for the best-fit solution, MPFIT by default calculates derivatives numerically via a finite difference approximation. The user-supplied function need not calculate the derivatives explicitly. However, if you desire to compute them analytically, then the `AUTODERIVATIVE=0` keyword must be passed to MPFIT. As a practical matter, it is often sufficient and even faster to allow MPFIT to calculate the derivatives numerically, and so `AUTODERIVATIVE=0` is not necessary.

If `AUTODERIVATIVE=0` is used then the user function must check the parameter `FJAC`, and if `FJAC!=None` then return the partial derivative array in the return list.

```
def myfunct(p, fjac=None, x=None, y=None, err=None)
        # Parameter values are passed in "p"
        # If FJAC!=None then partial derivatives must be comptuer.
        # FJAC contains an array of len(p), where each entry
        # is 1 if that parameter is free and 0 if it is fixed.
        model = F(x, p)
        # Non-negative status value means MPFIT should continue,
    # negative means stop the calculation.
        status = 0
        if (dojac):
```

(continues on next page)

(continued from previous page)

```
            pderiv = zeros([len(x), len(p)], Float)
            for j in range(len(p)):
                    pderiv[:,j] = FGRAD(x, p, j)
        else:
            pderiv = None
        return([status, (y-model)/err, pderiv]
```

where `FGRAD(x, p, i)` is a user function which must compute the derivative of the model with respect to parameter `P[i]` at `X`. When finite differencing is used for computing derivatives (i.e., when AUTODERIVATIVE=1), or when MPFIT needs only the errors but not the derivatives the parameter FJAC=None.

Derivatives should be returned in the `PDERIV` array. `PDERIV` should be an MxN array, where M is the number of data points and N is the number of parameters. `dp[i,j]` is the derivative at the ith point with respect to the jth parameter.

The derivatives with respect to fixed parameters are ignored; zero is an appropriate value to insert for those derivatives. Upon input to the user function, `FJAC` is set to a vector with the same length as `P`, with a value of 1 for a parameter which is free, and a value of zero for a parameter which is fixed (and hence no derivative needs to be calculated).

If the data is higher than one dimensional, then the *last* dimension should be the parameter dimension. Example: fitting a 50x50 image, "dp" should be 50x50xNPAR.

2.10.4 Constraining Parameter Values

The behavior of MPFIT can be modified with respect to each parameter to be fitted. A parameter value can be fixed; simple boundary constraints can be imposed; limitations on the parameter changes can be imposed; properties of the automatic derivative can be modified; and parameters can be tied to one another.

These properties are governed by the `PARINFO` structure, which is passed as a keyword parameter to MPFIT.

`PARINFO` should be a list of dictionaries, one list entry for each parameter. Each parameter is associated with one element of the array, in numerical order. The dictionary can have the following keys (none are required, keys are case insensitive):

- **value** - the starting parameter value (but see the XALL parameter for more information).
- **fixed** - an "boolean" integer of 0 or 1 that determined whether the parameter is to be held fixed or not. If set (1), the parameter value will be held fixed. Fixed parameters are not varied by MPFIT, but are passed on to MYFUNCT for evaluation.
- **limited** - a two-element "boolean" integer array. If the first/second element is set, then the parameter is bounded on the lower/upper side. A parameter can be bounded on both sides. Both LIMITED and LIMITS must be given together.
- **limits** - a two-element float array. Gives the parameter limits on the lower and upper sides, respectively. A value will only be used as a limit if the corresponding value of LIMITED is set (=1) Both LIMITED and LIMITS must be given together.
- **parname** - a string, giving the name of the parameter. The fitting code of MPFIT does not use this tag in any way. However, the default iterfunct will print the parameter name if available.
- **step** - the step size to be used in calculating the numerical derivatives. If set to zero, then the step size is computed automatically. Ignored when AUTODERIVATIVE=0.
- **mpside** - the sidedness of the finite difference when computing numerical derivatives. This field can take four values:

  0 - one-sided derivative computed automatically

  1 - one-sided derivative (f(x+h) - f(x))/h

-1 - one-sided derivative (f(x) - f(x-h))/h

2 - two-sided derivative (f(x+h) - f(x-h))/(2*h)

Where "h" is the STEP parameter described above. The "automatic" one-sided derivative method will chose a direction for the finite difference which does not violate any constraints. The other methods do not perform this check. The two-sided method is in principle more precise, but requires twice as many function evaluations. Default: 0.

- **mpmaxstep** - the maximum change to be made in the parameter value. During the fitting process, the parameter will never be changed by more than this value in one iteration. A value of 0 indicates no maximum. Default: 0.
- **tied** - a string expression which "ties" the parameter to other free or fixed parameters. Any expression involving constants and the parameter array P are permitted. Example: if parameter 2 is always to be twice parameter 1 then use the following: parinfo(2).tied = '2 * p(1)'. Since they are totally constrained, tied parameters are considered to be fixed; no errors are computed for them. [NOTE: the PARNAME can't be used in expressions.]
- **mpprint** - if set to 1, then the default iterfunct will print the parameter value. If set to 0, the parameter value will not be printed. This tag can be used to selectively print only a few parameter values out of many. Default: 1 (all parameters printed)

Future modifications to the `PARINFO` structure, if any, will involve adding dictionary tags beginning with the two letters "MP". Therefore programmers are urged to avoid using tags starting with the same letters; otherwise they are free to include their own fields within the `PARINFO` structure, and they will be ignored.

PARINFO Example:

```
parinfo = [{'value':0., 'fixed':0, 'limited':[0,0],
           'limits':[0.,0.]} for i in range(5)]
parinfo[0]['fixed'] = 1
parinfo[4]['limited'][0] = 1
parinfo[4]['limits'][0]  = 50.
values = [5.7, 2.2, 500., 1.5, 2000.]
for i in range(5):
  parinfo[i]['value']=values[i]
```

A total of 5 parameters, with starting values of 5.7, 2.2, 500, 1.5, and 2000 are given. The first parameter is fixed at a value of 5.7, and the last parameter is constrained to be above 50.

2.10.5 Example

```
import mpfit
import numpy.oldnumeric as Numeric
x = arange(100, float)
p0 = [5.7, 2.2, 500., 1.5, 2000.]
y = (p[0] + p[1]*[x] + p[2]*[x**2] + p[3]*sqrt(x) +
     p[4]*log(x))
fa = {'x':x, 'y':y, 'err':err}
m = mpfit('myfunct', p0, functkw=fa)
print 'status = ', m.status
if (m.status <= 0):
    print 'error message = ', m.errmsg
print 'parameters = ', m.params
```

Minimizes sum of squares of MYFUNCT. MYFUNCT is called with the X, Y, and ERR keyword parameters that are given by FUNCTKW. The results can be obtained from the returned object m.

2.10.6 Theory Of Operation

There are many specific strategies for function minimization. One very popular technique is to use function gradient information to realize the local structure of the function. Near a local minimum the function value can be taylor expanded about x0 as follows:

```
f(x) = f(x0) + f'(x0) . (x-x0) + (1/2) (x-x0) . f''(x0) . (x-x0)
   -----   ---------------   ------------------------------- (1)
Order   0th                 1st                         2nd
```

Here f'(x) is the gradient vector of f at x, and f''(x) is the Hessian matrix of second derivatives of f at x. The vector x is the set of function parameters, not the measured data vector. One can find the minimum of f, f(xm) using Newton's method, and arrives at the following linear equation:

```
f''(x0) . (xm-x0) = - f'(x0)                                    (2)
```

If an inverse can be found for f''(x0) then one can solve for (xm-x0), the step vector from the current position x0 to the new projected minimum. Here the problem has been linearized (ie, the gradient information is known to first order). f''(x0) is symmetric NxN matrix, and should be positive definite.

The Levenberg-Marquardt technique is a variation on this theme. It adds an additional diagonal term to the equation which may aid the convergence properties:

```
(f''(x0) + nu I) . (xm-x0) = -f'(x0)                            (2a)
```

where I is the identity matrix. When nu is large, the overall matrix is diagonally dominant, and the iterations follow steepest descent. When nu is small, the iterations are quadratically convergent.

In principle, if f''(x0) and f'(x0) are known then xm-x0 can be determined. However the Hessian matrix is often difficult or impossible to compute. The gradient f'(x0) may be easier to compute, if even by finite difference techniques. So-called quasi-Newton techniques attempt to successively estimate f''(x0) by building up gradient information as the iterations proceed.

In the least squares problem there are further simplifications which assist in solving eqn (2). The function to be minimized is a sum of squares:

```
f = Sum(hi^2)                                                   (3)
```

where hi is the ith residual out of m residuals as described above. This can be substituted back into eqn (2) after computing the derivatives:

```
f'  = 2 Sum(hi  hi')
f'' = 2 Sum(hi' hj') + 2 Sum(hi hi'')                           (4)
```

If one assumes that the parameters are already close enough to a minimum, then one typically finds that the second term in f'' is negligible (or, in any case, is too difficult to compute). Thus, equation (2) can be solved, at least approximately, using only gradient information.

In matrix notation, the combination of eqns (2) and (4) becomes:

```
hT' . h' . dx = - hT' . h                                       (5)
```

Where h is the residual vector (length m), hT is its transpose, h' is the Jacobian matrix (dimensions n x m), and dx is (xm-x0). The user function supplies the residual vector h, and in some cases h' when it is not found by finite differences (see MPFIT_FDJAC2, which finds h and hT'). Even if dx is not the best absolute step to take, it does provide a good estimate of the best *direction*, so often a line minimization will occur along the dx vector direction.

The method of solution employed by MINPACK is to form the Q . R factorization of h', where Q is an orthogonal matrix such that QT . Q = I, and R is upper right triangular. Using h' = Q . R and the ortogonality of Q, eqn (5) becomes

```
(RT . QT) . (Q . R) . dx = - (RT . QT) . h
         RT . R . dx = - RT . QT . h                    (6)
              R . dx = - QT . h
```

where the last statement follows because R is upper triangular. Here, R, QT, and h are known so this is a matter of solving for dx. The routine MPFIT_QRFAC provides the QR factorization of h, with pivoting, and MPFIT_QRSOLV provides the solution for dx.

2.10.7 Authors

The original version of this software, called LMFIT, was written in FORTRAN as part of the MINPACK-1 package by XXX.

Craig Markwardt converted the FORTRAN code to IDL. The information for the IDL version is:

> Craig B. Markwardt, NASA/GSFC Code 662, Greenbelt, MD 20770 craigm@lheamail.gsfc.nasa.gov UPDATED VERSIONs can be found on my WEB PAGE: http://cow.physics.wisc.edu/~craigm/idl/idl.html

Mark Rivers created this Python version from Craig's IDL version.

> Mark Rivers, University of Chicago Building 434A, Argonne National Laboratory 9700 South Cass Avenue, Argonne, IL 60439 rivers@cars.uchicago.edu Updated versions can be found at: http://cars.uchicago.edu/software

Sergey Koposov converted the Mark's Python version from Numeric to numpy

> Sergey Koposov, University of Cambridge, Institute of Astronomy, Madingley road, CB3 0HA, Cambridge, UK koposov@ast.cam.ac.uk Updated versions can be found at: https://github.com/segasai/astrolibpy

2.10.8 Modification History

- Translated from MINPACK-1 in FORTRAN, Apr-Jul 1998, CM

- Translated from MPFIT (Craig Markwardt's IDL package) to Python, August, 2002. Mark Rivers
- Converted from Numeric to numpy (Sergey Koposov, July 2008)
- Added full Python 3 compatibility (Sergey Koposov, Feb 2017)

References

CHAPTER

THREE

CODE EXAMPLES

3.1 Downloading EIS Data with Fido

```
>>> from sunpy.net import Fido, attrs as a
>>> import eispac.net
>>> from eispac.net.attrs import FileType
>>> results = Fido.search(a.Time('2020-11-09 00:00:00','2020-11-09 01:00:00'),
...                       a.Instrument('EIS'),
...                       a.Physobs.intensity,
...                       a.Source('Hinode'),
...                       a.Provider('NRL'),
...                       a.Level('1'))
>>> results
<sunpy.net.fido_factory.UnifiedResponse object at ...>
Results from 1 Provider:

3 Results from the EISClient:
Source: https://eis.nrl.navy.mil/

       Start Time               End Time         ... Level   FileType
----------------------- ----------------------- ... ----- -----------
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1   HDF5 data
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1 HDF5 header
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1        FITS


>>> results = Fido.search(a.Time('2020-11-09 00:00:00','2020-11-09 01:00:00'),
...                       a.Instrument('EIS'),
...                       a.Physobs.intensity,
...                       a.Source('Hinode'),
...                       a.Provider('NRL'),
...                       a.Level('1'),
...                       FileType('HDF5 header'))
>>> results
<sunpy.net.fido_factory.UnifiedResponse object at ...>
Results from 1 Provider:

1 Results from the EISClient:
Source: https://eis.nrl.navy.mil/
```

(continues on next page)

(continued from previous page)

```
       Start Time               End Time        ... Level   FileType
----------------------- ----------------------- ... ----- -----------
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1 HDF5 header
```

3.2 Reading and Exploring EIS Data

Reading in a level-1 HDF5 data file and printing some metadata

```
>>> import eispac
>>> data_filename = 'eis_20190404_131513.data.h5'
>>> data_cube = eispac.read_cube(data_filename, 195.119)
>>> data_cube.meta.keys()
dict_keys(['filename_data', 'filename_head', 'wininfo', 'iwin', 'iwin_str',
           'index', 'pointing', 'wave', 'radcal', 'slit_width',
           'slit_width_units', 'ccd_offset', 'wave_corr', 'wave_corr_t',
           'wave_corr_tilt', 'date_obs', 'date_obs_format', 'duration',
           'duration_units', 'mod_index', 'aspect', 'aspect_ratio', 'notes'])

>>> data_cube.meta['pointing']['x_scale']
2.9952

>>> data_cube.meta['radcal']
array([8.06751  , 8.060929 , 8.054517 , 8.048271 , 8.042198 , 8.036295 ,
       8.030562 , 8.024157 , 8.017491 , 8.010971 , 8.0046015, 7.998385 ,
       7.9923196, 7.9864078, 7.980654 , 7.975055 , 7.969617 , 7.9643393,
       7.959224 , 7.9542727, 7.949487 , 7.9448686, 7.9404206, 7.9361415],
      dtype=float32)
```

Viewing information about the spectral windows available in a file.

```
>>> header_filename = 'eis_20190404_131513.head.h5'
>>> wininfo = eispac.read_wininfo(header_filename)
>>> wininfo.dtype.names
('iwin', 'line_id', 'wvl_min', 'wvl_max', 'nl', 'xs')

>>> print(wininfo)
[(0, 'Fe XI 180.400', 180.03426, 180.72559, 32, 661),
 (1, 'Ca XV 182.100', 181.75139, 182.44266, 32, 738),
 (2, 'Fe X 184.720', 183.82512, 185.5865 , 80, 831),
 (3, 'Fe XII 186.750', 186.3891 , 187.0802 , 32, 946)],
 ... ... ...
 (23, 'Mg VII 280.390', 279.7766 , 280.9996 , 56, 3720),
 (24, 'Fe XV 284.160', 283.89   , 284.40134, 24, 3905)]
```

3.3 Slicing an EISCube

Slicing an EISCube using real-world coordinates

```
>>> import eispac
>>> import astropy.units as u
>>> from astropy.coordinates import SkyCoord, SpectralCoord
>>> from astropy.wcs.utils import wcs_to_celestial_frame

>>> data_filename = 'eis_20190404_131513.data.h5'
>>> data_cube = eispac.read_cube(data_filename, 195.119)
>>> eis_frame = wcs_to_celestial_frame(data_cube.wcs)
>>> lower_left = [SpectralCoord(195.0, unit=u.angstrom),
...               SkyCoord(Tx=48, Ty=225, unit=u.arcsec, frame=eis_frame)]
>>> upper_right = [SpectralCoord(195.3, unit=u.AA),
...                SkyCoord(Tx=165, Ty=378, unit=u.arcsec, frame=eis_frame)]
>>> data_cutout = data_cube.crop(lower_left, upper_right)

>>> data_cube.dimensions
[512, 87, 24] pix

>>> data_cutout.dimensions
[154, 48, 14] pix
```

3.4 Basic Plotting

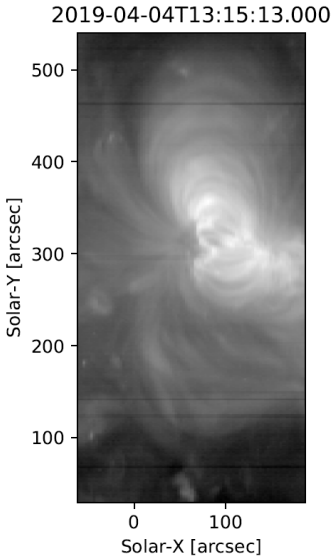
Making a log-scaled plot of intensities summed over the wavelength axis.

```
import numpy as np
import matplotlib.pyplot as plt
import eispac

data_filename = 'eis_20190404_131513.data.h5'
data_cube = eispac.read_cube(data_filename, 195.119)
raster_sum = np.sum(data_cube.data, axis=2) # or data_cube.sum_spectra().data
scaled_img = np.log10(raster_sum)

plt.figure()
plt.imshow(scaled_img, origin='lower', extent=data_cube.meta['extent_arcsec'], cmap='gray
↪')
plt.title(data_cube.meta['date_obs'][-1])
plt.xlabel('Solar-X [arcsec]')
plt.ylabel('Solar-Y [arcsec]')
plt.show()
```

Plotting a single line profile (in units of counts)

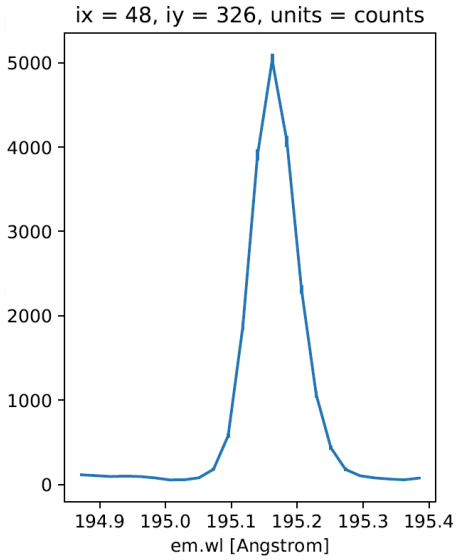
```
>>> ix = 48
>>> iy = 326
>>> spec = data_cube[iy,ix,:].remove_radcal()
```

(continues on next page)



(continued from previous page)

```
>>> spec_plot = spec.plot()
>>> spec_plot.set_title(f'ix = {ix}, iy = {iy}, units = counts')
```



3.5 Basic Example

Minimal example program that just loads an observation, fits all the spectra, and saves the results.

```
import eispac

if __name__ == '__main__':
    # input data and template files
```

(continues on next page)

(continued from previous page)

```
    data_filepath = './eis_20190404_131513.data.h5'
    template_filepath = './fe_12_195_119.2c.template.h5'

    # read fit template
    tmplt = eispac.read_template(template_filepath)

    # Read spectral window into an EISCube
    data_cube = eispac.read_cube(data_filepath, tmplt.central_wave)

    # Fit the data, then save it to disk and test loading it back in
    fit_res = eispac.fit_spectra(data_cube, tmplt, ncpu='max')
    save_filepaths = eispac.save_fit(fit_res, save_dir='cwd')
    FITS_file = eispac.export_fits(fit_res, save_dir='cwd')
    load_fit = eispac.read_fit(save_filepaths[0])
```

3.6 Advanced Example

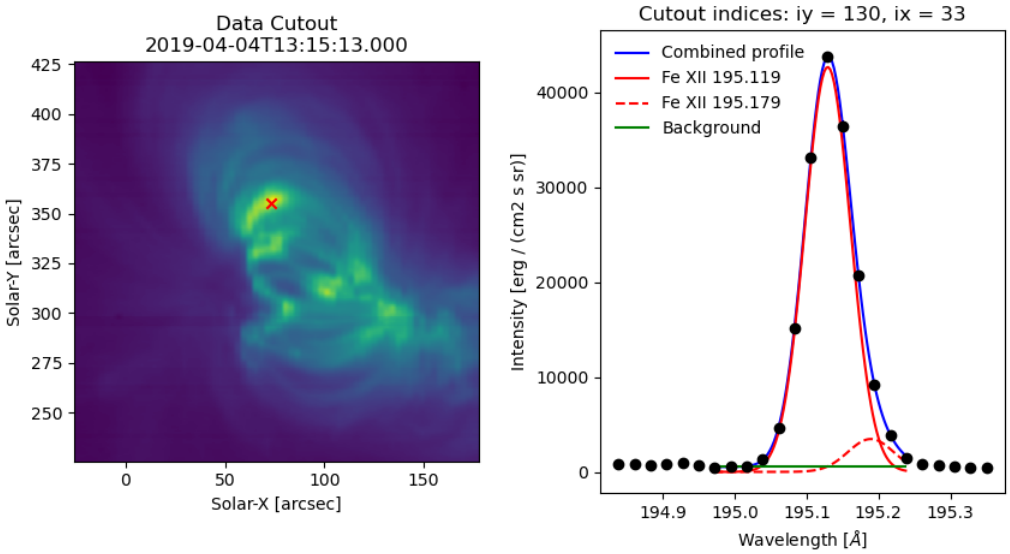
Complete example demonstrating reading data, slicing an EISCube, fitting the spectra, extracting the profile at the coordinates of maximum intensity, and finally making a nice overview plot.

```
import numpy as np
import matplotlib.pyplot as plt
import astropy.units as u
from astropy.coordinates import SkyCoord
from astropy.wcs.utils import wcs_to_celestial_frame
import eispac

if __name__ == '__main__':
    # Read in the fit template and EIS observation
    data_filepath = './eis_20190404_131513.data.h5'
    template_filepath = './fe_12_195_119.2c.template.h5'
    tmplt = eispac.read_template(template_filepath)
    data_cube = eispac.read_cube(data_filepath, tmplt.central_wave)

    # Select a cutout of the raster
    eis_frame = wcs_to_celestial_frame(data_cube.wcs)
    lower_left = [None, SkyCoord(Tx=-25, Ty=225, unit=u.arcsec, frame=eis_frame)]
    upper_right = [None, SkyCoord(Tx=175, Ty=425, unit=u.arcsec, frame=eis_frame)]
    raster_cutout = data_cube.crop(lower_left, upper_right)

    # Fit the data and save it to disk
    fit_res = eispac.fit_spectra(raster_cutout, tmplt, ncpu='max')
    save_filepaths = eispac.save_fit(fit_res, save_dir='cwd')

    # Find indices and world coordinates of max intensity
    sum_data_inten = raster_cutout.sum_spectra().data
    iy, ix = np.unravel_index(sum_data_inten.argmax(), sum_data_inten.shape)
    ex_world_coords = raster_cutout.wcs.array_index_to_world(iy, ix, 0)[1]
    y_arcsec, x_arcsec = ex_world_coords.Ty.value, ex_world_coords.Tx.value
```

(continues on next page)

(continued from previous page)

```
# Extract data profile and interpolate fit at higher spectral resolution
data_x = raster_cutout.wavelength[iy, ix, :]
data_y = raster_cutout.data[iy, ix, :]
data_err = raster_cutout.uncertainty.array[iy, ix, :]
fit_x, fit_y = fit_res.get_fit_profile(coords=[iy,ix], num_wavelengths=100)
c0_x, c0_y = fit_res.get_fit_profile(0, coords=[iy,ix], num_wavelengths=100)
c1_x, c1_y = fit_res.get_fit_profile(1, coords=[iy,ix], num_wavelengths=100)
c2_x, c2_y = fit_res.get_fit_profile(2, coords=[iy,ix], num_wavelengths=100)

# Make a multi-panel figure with the cutout and example profile
fig = plt.figure(figsize=[10,5])
plot_grid = fig.add_gridspec(nrows=1, ncols=2, wspace=0.3)

data_subplt = fig.add_subplot(plot_grid[0,0])
data_subplt.imshow(sum_data_inten, origin='lower', extent=cutout_extent)
data_subplt.scatter(x_arcsec, y_arcsec, color='r', marker='x')
data_subplt.set_title('Data Cutout\n'+raster_cutout.meta['mod_index']['date_obs'])
data_subplt.set_xlabel('Solar-X [arcsec]')
data_subplt.set_ylabel('Solar-Y [arcsec]')

profile_subplt = fig.add_subplot(plot_grid[0,1])
profile_subplt.errorbar(data_x, data_y, yerr=data_err, ls='', marker='o', color='k')
profile_subplt.plot(fit_x, fit_y, color='b', label='Combined profile')
profile_subplt.plot(c0_x, c0_y, color='r', label=fit_res.fit['line_ids'][0])
profile_subplt.plot(c1_x, c1_y, color='r', ls='--', label=fit_res.fit['line_ids'][1])
profile_subplt.plot(c2_x, c2_y, color='g', label='Background')
profile_subplt.set_title(f'Cutout indices: iy = {iy}, ix = {ix}')
profile_subplt.set_xlabel('Wavelength [$\AA$]')
profile_subplt.set_ylabel('Intensity ['+raster_cutout.unit.to_string()+']')
profile_subplt.legend(loc='upper left', frameon=False)
plt.show()
```



CHAPTER

FOUR

COMMUNITY GUIDELINES

As an open-source code licensed under the MIT license, you are free to download and modify `EISPAC` (see the license file for the technical details).

We welcome any and all community feedback and contributions. There are a few ways you can get involved.

4.1 User Support

The fastest way to get help with EISPAC is to either email a member of the NRL EISPAC team or, preferably, open an issue on GitHub.

A non-exhaustive list of possible issue topics:

- General help using the software
- Reporting a bug
- Suggesting new features
- Requesting additional details not covered in the documentation
- Proposing updates or extensions to the documentation

4.2 Contributing to the Code

Know your way around git / GitHub and want to directly help with the code? Great! Please feel free to fork the repository on GitHub. Before you start coding, please take a look at the currently open issues and consider addressing one that fits your skills and interests. If, instead, there is a bug or feature that you would like to work on that does not have an open issue, open a new issue first to let us know what you are working on. Later, once you have made your patch or extension to EISPAC, open an Pull Request on GitHub and we will begin the review process.

A few things to keep in mind:

- Please test your code before opening a pull request and confirm that it gives the expected result / fixes the target issue. If you add a new function or feature, consider adding a test function (this is not strictly necessary, we can help write unit and integration tests if you are unfamiliar with the process).
- Please appropriately comment and document your code. We try to use the numpydoc style for docstrings and follow the common Python coding standards as described by PEP8.
- The primary purpose of EISPAC is to provide tools for downloading, reading, and fitting EIS observations and support analysis done in conjunction with other solar and heliospheric packages (e.g. `SunPy`, `NDCube`, and `AstroPy`). As such, some advanced features may be out-of-scope for EISPAC itself. These topics, however, are

still important! So please contact us (via either email or opening an issue) and we can help determine where your idea fits best.

CHAPTER

FIVE

API REFERENCE

5.1 eispac core

All functions and classes in `core` are loaded into the top-level `eispac` namespace. It is therefore recommended to access them via calls like `eispac.FUNCTION()` rather than `eispac.core.FUNCTION()`

5.1.1 Functions

create_fit_dict(n_pxls, n_steps, n_wave, ...)	Dictionary to hold the fit parameters returned by fit_spectra()
export_fits(fit_result[, save_dir, verbose])	Save fit line intensites, velocites, and widths to a .fits file
fit_spectra(inten, template[, parinfo, ...])	Fit one or more EIS line spectra using mpfit (with multiprocessing).
fit_spectra_astropy(inten, template[, ...])	Fit one or more EIS line spectra using mpfit.
generate_astropy_model(template)	Create an Astropy fitting model using a template intended for MPFIT.
lineid_to_name(lineid)	Convert line IDs to filesystem-friendly strings
match_templates(eis_obs)	Generate a list of all template files that match an EIS file or window.
multigaussian(param, x[, n_gauss, n_poly])	Multigaussian model with a polynomial background
multigaussian_deviates(param[, x, y, error, ...])	Computes the deviates between a multigaussian model fit and input data.
read_cube([filename, window, apply_radcal, ...])	Load a single window of EIS data from an HDF5 file into an EISCube object
read_fit(filename[, verbose])	Load an EISFitResult object from an HDF5 file
read_template(filename)	Create *EISFitTemplate* from template file
read_wininfo(filename)	Read the window information from an EIS HDF5 header file
save_fit(fit_result[, save_dir, verbose])	Save an EISFitResult object to an HDF5 file (or files)
scale_guess(x, y, param, n_gauss, n_poly)	Scale inital guess of multigaussian model parameters to data values

create_fit_dict

eispac.core.**create_fit_dict**(*n_pxls*, *n_steps*, *n_wave*, *n_gauss*, *n_poly*, *data_units='unknown'*)

Dictionary to hold the fit parameters returned by fit_spectra()

Parameters

- **n_pxls** (*int*) – Number of pixels along each data slit
- **n_steps** (*int*) – Number of steps in the raster or sit-and-stare observation set
- **n_wave** (*int*) – number of wavelength points
- **n_gauss** (*int*) – Number of Gaussian functions in the combinted fit profile
- **n_poly** (*int*) – Degree of background polynomial
- **data_units** (*str, optional*) – String name of the data units (e.g. "counts", "erg / (cm2 s sr)", etc.) Default is "unknown"

Returns

output – Empty dictionary with the correct dimensions and keys for a fit result.

Return type

dict

export_fits

eispac.core.**export_fits**(*fit_result*, *save_dir=None*, *verbose=False*)

Save fit line intensites, velocites, and widths to a .fits file

Parameters

- **fit_result** (*EISFitResult* object) – Fit parameter results from eispac.fit_spectra()
- **save_dir** (str or pathlib.Path object, optional) – Directory where the fit results should be saved. If set to None, the results will be saved in the same folder as the source data. If set to, 'cwd', the results will be saved to the current working directory. Default is None.
- **verbose** (*bool, optional*) – If set to True, will print additional information to the console. Default is False.

Returns

output_filepath – Path object pointing to the output files. Each parameter is saved in a seperate file and, if there are more than one spectral lines fit, in the template, each line will be saved to its own set of files.

Return type

list of pathlib.Path objects

fit_spectra

eispac.core.**fit_spectra**(*inten*, *template*, *parinfo=None*, *wave=None*, *errs=None*, *min_points=7*, *ncpu='max'*, *unsafe_mp=False*, *ignore_warnings=False*, *skip_fitting=False*, *debug=False*)

Fit one or more EIS line spectra using mpfit (with multiprocessing).

Parameters

- **inten** (*EISCube object, array_like, or filepath*) – One or more intensity profiles to be fit. The code will loop over the data according to its dimensionality. 3D data is assumed to be a full EIS raster (or a sub region), 2D data is assumed to be a single EIS slit, and 1D data is assumed to be a single profile.
- **template** (*EISFitTemplate object, dict, or filepath*) – Either an EISFitTemplate, a 'template' dictionary, or the path to a template file.
- **parinfo** (*list, optional*) – List of dictionaries with fit parameters formatted for use with mpfit. Will supercede any parinfo lists loaded from an EISFitTemplate. Required if the 'template' parameter is given as a dictionary.
- **wave** (*array_like, optional*) – Associated wavelength values for the spectra. Required if 'inten' is given as an array and ignored otherwise.
- **errs** (*array_like, optional*) – Intensity error values for the spectra. Required if 'inten' is given as an array and ignored otherwise.
- **min_points** (*int, optional*) – Minimum number of good quality data points (i.e. non-zero values & errs) to be used in each fit. Spectra with fewer data points will be skipped. Must be a number >= the total number of fit parameters. Default is 7.
- **ncpu** (*int, optional*) – Number of cpu processes to parallelize over. Must be less than or equal to the total number of cores the system has. If set to 'max' or None, the code will use the maximum number of cores available. Default is 'max'. Important: due to the specifics of how the multiprocessing library works, any statements that call fit_spectra() using ncpu > 1 MUST be wrapped in a "if __name__ == '__main__':" statement in the top-level program. If such a "name guard" statement is not detected, this function will fall back to using a single process.
- **unsafe_mp** (*bool, optional*) – If set to True (and ncpu > 0), will use multiprocessing even if there is no "name guard" in use (see above). Used by the console script "eit_fit_files". Default is False (name guard enforced). Disabling the name guard runs the risk of spawning infinite processes if run incorrectly. USE AT YOUR OWN RISK!
- **ignore_warnings** (*bool, optional*) – If set to True, will silence the warning about a missing or disabled name guard (we are serious at it, be careful). Default is False.
- **skip_fitting** (*bool, optional*) – If set to True, will skip the fitting altogether and just return an empty EISFitResult instance. Used mainly for testing. Default is False.
- **debug** (*bool, optional*) – If set to True, will print some extra information useful for debugging development versions of the code. Default is False.

Returns

fit_res – An EISFitResult object containing the output fit parameters.

Return type

EISFitResult class instance

fit_spectra_astropy

eispac.core.**fit_spectra_astropy**(*inten*, *template*, *parinfo=None*, *wave=None*, *errs=None*, *min_points=10*)

Fit one or more EIS line spectra using mpfit.

Parameters

- **inten** (*EISCube object or array_like*) – One or more intensity profiles to be fit. The code will loop over the data according to its dimensionality. 3D data is assumed to be a full EIS raster (or a sub region), 2D data is assumed to be a single EIS slit, and 1D data is assumed to be a single profile.
- **template** (*EISFitTemplate object or dict*) – Either an EISFitTemplate or the just a 'template' dictionary.
- **parinfo** (*dict, optional*) – Dictionary of fit parameters formatted for use with mpfit. Required if the 'template' parameter is given just a dict and ignored otherwise.
- **wave** (*array_like, optional*) – Associated wavelength values for the spectra. Required if 'inten' is given an array and ignored otherwise.
- **errs** (*array_like, optional*) – Intensity error values for the spectra. Required if 'inten' is given an array and ignored otherwise.
- **min_points** (*int, optional*) – Minimum number of good quality data points (i.e. non-zero values & errs) to be used in each fit. Spectra with fewer data points will be skipped. Default is 10.

Returns

fit_res – An EISFitResult object containing the output fit paramaters.

Return type

EISFitResult class instance

generate_astropy_model

eispac.core.**generate_astropy_model**(*template*)

Create an Astropy fitting model using a template intended for MPFIT.

Parameters

template (*EISFitTemplate object or string*) – Either a single MPFIT-style template object or a string containing the full path to a template file. The template MUST contain both 'template' and 'parinfo' attributes.

Returns

combined_model – A fully initialized CompoundModel object with the same parameter values and constraints (including tied parameter functions) as given in the 'parinfo' attribute of the input template.

Return type

CompoundModel object from Astropy.modeling

lineid_to_name

eispac.core.**lineid_to_name**(*lineid*)

Convert line IDs to filesystem-friendly strings

Parameters
lineid (*str*) – Line ID string (e.g. Fe XII 195.119)

Returns
name – Line name as a strong suitable for filenames (e.g. fe_12_195_119)

Return type
str

match_templates

eispac.core.**match_templates**(*eis_obs*)

Generate a list of all template files that match an EIS file or window.

Parameters
eis_obs (*EISCube* object, str, or pathlib.Path) – EIS data to use for searching. If given an EISCube, will only find templates that match the selected spectral window. If given the filepath to an EIS level-1 HDF5 file, will find all templates that match each window in the observation.

Returns
matched_templates – List of template files that match the selected spectral window. If a filepath was input, a list of lists will be returned instead. Each sublist contains the matched templates for the corresponding window (e.g. matched_templates[0] would contain the list of templates matching the first window in the data file).

Return type
list or list of lists

multigaussian

eispac.core.**multigaussian**(*param*, *x*, *n_gauss=1*, *n_poly=0*)

Multigaussian model with a polynomial background

Parameters

- **param** (*array_like*) – Model fit parameters. There must be 3*n_gauss + n_poly param values. For each Gaussian component, the parameters are assumed to have the following order: [peak, centroid, width]
- **x** (*array_like*) – Independent variable values to evaluate the function at. For EIS data, this will usually correspond to wavelength values.
- **n_gauss** (*int, optional*) – Number of Gaussian components. Default is "1"
- **n_poly** (*int, optional*) – Number of background polynomial terms. Common values are: o (no background), 1 (constant), and 2 (linear). Default is "0"

Returns
f – Function evaluated at each x value

Return type
array_like

multigaussian_deviates

eispac.core.**multigaussian_deviates**(*param*, *x=None*, *y=None*, *error=None*, *n_gauss=None*, *n_poly=None*, *fjac=None*, *debug=False*)

Computes the deviates between a multigaussian model fit and input data.

Parameters

- **param** (*array_like*) – Model fit parameters. There must be 3*n_gauss + n_poly param values. For each Gaussian component, the parameters are assumed to have the following order: [peak, centroid, width]
- **x** (*array_like*) – Independent variable values to evaluate the function at. For EIS data, this will usually correspond to wavelength values.
- **y** (*array_like*) – Measured values. For EIS data, this will usually correspond to intensity observations.
- **error** (*array_like*) – Error values for each measurment.
- **n_gauss** (*int*) – Number of Gaussian components.
- **n_poly** (*int*) – Number of background polynomial terms. Common values are: 0 (no background), 1 (constant), and 2 (linear).
- **fjac** (*None*) – Used by mpfit. When fjac == None, partial derivatives will NOT be calculated. This is the default for mpfit.
- **debug** (*boolean, optional*) – Toggles 'debugging mode'. If set to 'True', then the code will print an error statement and exit when it encounters invalid inputs or empty data. Default it 'False', which will result in just a negative status flag.

Return type
[status, deviates]

read_cube

eispac.core.**read_cube**(*filename=None*, *window=0*, *apply_radcal=True*, *radcal=None*, *abs_errs=True*, *count_offset=None*, *debug=False*)

Load a single window of EIS data from an HDF5 file into an EISCube object

Parameters

- **filename** (str or pathlib.Path object) – Name of either the data or head HDF5 file for a single EIS observation
- **window** (*int, float, or str, optional*) – Requested spectral window number or the value of any wavelength within the requested window. Default is '0'
- **apply_radcal** (*bool, optional*) – If set to True, will apply the pre-flight radiometric calibration curve found in the HDF5 header file and set units to erg/(cm^2 s sr). If set to False, will simply return the data in units of photon counts. Default is True.
- **radcal** (*array_like, optional*) – User-inputted radiometric calibration curve to be applied to the data.
- **abs_errs** (*bool, optional*) – If set to True, will calulate errors based on the absolute value of the counts. This allows for reasonable errors to be estimated for valid negative count values that are the result of the dark count subtraction method (not bad or filled data). Default it True.

- **count_offset** (*int or float, optional*) – Constant value to add to the count array before error estimate or calibration. Could be useful for testing data processing methods. Default is None (no count offset).
- **debug** (*bool, optional*) – If set to True, will return a dictionary with the raw counts and metadata instead of an EISCube class instance. Useful for examining data files that fail to load properly.

Returns
output_cube – An EISCube class instance containing the requested spectral data window

Return type
EISCube class instance

read_fit

eispac.core.**read_fit**(*filename*, *verbose=False*)

Load an EISFitResult object from an HDF5 file

Parameters

- **filename** (str or pathlib.Path object) – String or path to the fit result file that should be loaded.
- **verbose** (*bool, optional*) – If set to True, will print the name of each data variable read in. Default is False.

Returns
fit_result – Copy of the fit results loaded from the file.

Return type
EISFitResult object

read_template

eispac.core.**read_template**(*filename*)

Create *EISFitTemplate* from template file

Parameters
filename (str or pathlib.Path) – Path to template file

Returns
cls – Object containing the fit template

Return type
EISFitTemplate class instance

read_wininfo

eispac.core.**read_wininfo**(*filename*)

Read the window information from an EIS HDF5 header file

Parameters
filename (str or pathlib.Path object) – Name of either the data or head HDF5 file for a single EIS observation

Returns
wininfo – A record array with the information for all spectral windows in the EIS observation

Return type
numpy.recarray

save_fit

eispac.core.**save_fit**(*fit_result*, *save_dir=None*, *verbose=False*)

Save an EISFitResult object to an HDF5 file (or files)

Parameters

- **fit_result** (*EISFitResult object*) – Fit parameter results from eispac.fit_spectra()
- **save_dir** (*str or pathlib.Path object, optional*) – Directory where the fit results should be saved. If set to None, the results will be saved in the same folder as the source data. If set to, 'cwd', the results will be saved to the current working directory. Default is None.
- **verbose** (*bool, optional*) – If set to True, will print the name of each data variable saved. Default is False.

Returns
output_filepath – Path object pointing to the output file. If there are more one than one spectral lines fit, multiple copies of the same output file will be saved but with different filenames.

Return type
list or pathlib.Path object

scale_guess

eispac.core.**scale_guess**(*x*, *y*, *param*, *n_gauss*, *n_poly*)

Scale inital guess of multigaussian model parameters to data values

Parameters

- **x** (*array_like*) – Independent variable values. For EIS data, this will usually correspond to wavelength values.
- **y** (*array_like*) – Observed data values. ForEIS data, this will be either raw counts or calibrated intensity measurements
- **param** (*array_like*) – Model fit parameters. There must be 3*n_gauss + n_poly param values. For each Gaussian component, the parameters are assumed to have the following order: [peak, centroid, width]
- **n_gauss** (*int, optional*) – Number of Gaussian components. Default is "1"
- **n_poly** (*int, optional*) – Number of background polynomial terms. Common values are: 0 (no background), 1 (constant), and 2 (linear). Default is "0"

Returns
newparam – Array of scaled model parameters.

Return type
array_like

5.1.2 Classes

EISCube(*args, **kwargs)	EIS Level-1 Data Cube
EISFitResult([wave, template, parinfo, ...])	Object containing the results from fitting one or more EIS window spectra
EISFitTemplate(filename, template, parinfo)	Representation of fitting parameters for a particular line or lines
EISMap(data[, header])	EIS fit parameter map.

EISCube

class eispac.core.**EISCube**(**args*, ***kwargs*)

Bases: NDCube

EIS Level-1 Data Cube

Subclass of NDCube. Accepts all of the standard arguments and keywords of ndcube.NDCube, as well as a few EIS specific parameters.

Parameters

- **data** (numpy.ndarray) – The array holding the actual data in this object.
- **wcs** (astropy.wcs.WCS, optional) – The WCS object containing the axes' information, optional only if data is an astropy.nddata.NDData object.
- **uncertainty** (*array_like, optional*) – Uncertainty in the dataset. Ideally, it should have an attribute "uncertainty_type" that defines what kind of uncertainty is stored, for example "std" for standard deviation or "var" for variance. A metaclass defining such an interface is NDUncertainty, however it's use in not mandatory. If the uncertainty has no type attribute, the uncertainty is stored as UnknownUncertainty. Defaults to None.
- **mask** (*array_like, optional*) – Mask for the dataset. Masks should follow the numpy convention that valid data points are marked by False and invalid ones with True. Defaults to None.
- **meta** (*dict-like object, optional*) – Additional meta information about the dataset. If no meta is provided an empty collections.OrderedDict is created. Default is None.
- **unit** (*Unit-like or str, optional*) – Unit for the dataset. Strings that can be converted to a Unit are allowed. Default is None.
- **copy** (*bool, optional*) – Indicates whether to save the arguments as copy. True copies every attribute before saving it while False tries to save every parameter as reference. Note however that it is not always possible to save the input as reference. Default is False.
- **wavelength** (numpy.ndarray, optional) – Numpy array with the corrected wavelength values for each location within the EIS raster. Must have the same dimensionality as the input data. If not given, will initialize the .wavelength property with an array of zeros.
- **radcal** (numpy.ndarray, optional) – Array of the radiometric calibration curve currently applied to the input data cube. Required if you wish to use the .apply_radcal() and .remove_radcal() methods

Attributes Summary

radcal	Current radiometric calibration curve
wavelength	Corrected wavelength values observed by EIS

Methods Summary

apply_radcal([input_radcal])	Apply a radiometric calibration curve (user-inputted or preflight)
crop_by_coords(*args, **kwargs)	REMOVED in NDCube 2.0
remove_radcal()	Remove the applied radiometric calibration and convert data to counts
smooth_cube([width])	Smooth the data along one or more spatial axes.
sum_spectra([wave_range, units])	Sum the data along the spectral axis.

Attributes Documentation

radcal

Current radiometric calibration curve

wavelength

Corrected wavelength values observed by EIS

Methods Documentation

apply_radcal(*input_radcal=None*)

Apply a radiometric calibration curve (user-inputted or preflight)

Parameters
input_radcal (*array_like, optional*) – User-inputted radiometric calibration curve. If set to None, will use the preflight radcal curve from the .meta dict. Default is None

Returns
output_cube – A new EISCube class instance containing the calibrated data

Return type
EISCube class instance

crop_by_coords(**args*, ***kwargs*)

REMOVED in NDCube 2.0

remove_radcal()

Remove the applied radiometric calibration and convert data to counts

Returns
output_cube – A new EISCube class instance containing the photon count data

Return type
EISCube class instance

smooth_cube(*width=3*, ***kwargs*)

Smooth the data along one or more spatial axes.

Parameters

- **width** (list or single value of ints, floats, or Quantity) – Number of pixels or angular distance to smooth over. If given a single value, only the y-axis will be smoothed. Floats and angular distances will be converted to the nearest whole pixel value. If a width value is even, width + 1 will be used instead. Default is width = 3
- ****kwargs** (*keywords or dict*) – Keyword arguments to be passed to the astropy.convolution.convolve() function.

Returns

output_cube – A new EISCube class instance containing the smoothed data

Return type

EISCube class instance

sum_spectra(*wave_range=None*, *units=Unit('Angstrom')*)

Sum the data along the spectral axis.

Parameters

- **wave_range** (list of ints, floats, or Quantity) – Wavelength range to sum over. Values can be input as either [min, max] or [center, half width]. Units can be specified using either Astropy units instances or by inputting a pair of ints or floats and then also using the "units" keyword. If wave_range is set to None, then entire spectra will be summed over. Default is None.
- **units** (str or Quantity) – Units to be used for the wavelength range if wave_range is given a list of ints or floats. Will be ignored if either wave_range is None or is given a list with Astropy units. Default is 'Angstrom'.

Returns

output_cube – A new 2D NDCube class instance containing the summed data (NB: not a full EISCube!)

Return type

NDCube class instance

EISFitResult

class eispac.core.**EISFitResult**(*wave=None*, *template=None*, *parinfo=None*, *func_name='multigaussian'*, *data_units='unknown'*, *radcal='unknown'*, *empty=False*)

Bases: object

Object containing the results from fitting one or more EIS window spectra

Parameters

- **wave** (*array_like*) – Wavelength values of the data being fit. Used only to determine the correct dimensions for the output arrays.
- **template** (*dict*) – Fit template parameters and metadata contained in the "template" attribute of an *EISFitTemplate* object.
- **parinfo** (*dict*) – Fit parameter initial values and constraints in the format expectd by mpfit. Normally found in the 'parinfo' attribute of an EISFitTemplate object

- **func_name** (*str, optional*) – String name of function that will be fit to the data. Must be one of the functions defined in the `fitting_functions` submodule. Default is "multi-gaussian".

template

Full copy of the input template dictionary

Type
dict

parinfo

Full copy of the input parinfo list

Type
list of dicts

funcinfo

Function component information generated from the template dict

Type
dict

n_pxls

Number of pixels along the y-axis

Type
int

n_steps

Number of raster steps or sit-and-stare exposures in the observation

Type
int

n_wave

Number of wavelength values in the window

Type
int

n_gauss

Number of Guassian functions in the fit

Type
int

n_poly

Number of terms in the polynomial background

Type
int

func_name

Name of the function fit to the data

Type
str

fit_func

Copy of the actual Python function named by 'fit_func'

Type
function

fit

Dictionary of output fit parameters

Type
dict

Attributes Summary

radcal	Current radiometric calibration curve

Methods Summary

apply_radcal([input_radcal])	Apply a radiometric calibration curve (user-inputted or preflight)
calculate_velocity([component, rest_wave, ...])	Calculate the Doppler velocity for a selected gaussian component
get_aspect_ratio()	Return the data aspect ratio (y-scale/x-scale) as a float
get_fit_profile([component, coords, ...])	Calculate the fit intensity profile (total or component) at a location.
get_map([component, measurement])	Return an EISMap of either intensity, velocity, or width
get_params([component, param_name, coords, ...])	Extract parameters values by component number, name, or pixel coords
plot_fov(end_time[, color, lw, ls])	Return a patch of the raster FOV for plotting on an image.
remove_radcal()	Remove the applied radiometric calibration and convert data to counts
rot_fov(end_time)	Return pointing information for the raster rotated to the input time.
shift2wave(array[, wave])	Shift an array from this fit to the desired wavelength

Attributes Documentation

radcal

Current radiometric calibration curve

Methods Documentation

apply_radcal(*input_radcal=None*)

Apply a radiometric calibration curve (user-inputted or preflight)

Parameters
input_radcal (*array_like, optional*) – User-inputted radiometric calibration curve. If set to None, will use the preflight radcal curve from the .meta dict. Default is None

Returns
output_fit – A new EISFitResult class instance containing the calibrated params

Return type
EISFitResult class instance

calculate_velocity(*component=0*, *rest_wave=None*, *update_results=True*, ***kwargs*)

Calculate the Doppler velocity for a selected gaussian component

Parameters

- **component** (*int, optional*) – Integer number of the fit gaussian. Default is 0 (first line in fit)
- **rest_wave** (*float or str, optional*) – Rest wavelength value in units of [Angstrom]. If given a string, will check to see if it is a spectral line ID and, if it is, will try to extract the rest wavelength. If set to None, will use the initial value specified in the .parinfo dictionary. Default is None.
- **update_results** (*bool, optional*) – If set to True, will update the .fit['vel'] and .fit['err_vel'] arrays in this. Default is True.
- ****kwargs** (*dict or keywords, optional*) – Optional keywords to pass to eispac.instr.calc_velocity()

Returns

- **velocity** (numpy.ndarray) – Array of calculated Doppler velocities
- **rel_error** (numpy.ndarray) – Array of relative error values for the velocities

get_aspect_ratio()

Return the data aspect ratio (y-scale/x-scale) as a float

get_fit_profile(*component=None*, *coords=None*, *num_wavelengths=None*, *wave_range='auto'*, *use_mask=True*)

Calculate the fit intensity profile (total or component) at a location.

Parameters

- **component** (*int or list, optional*) – Integer number (or list of ints) of the functional component(s). If set to None, will return the total combined fit profile. Default is None.
- **coords** (*list or tuple, optional*) – Array (Y, X) coordinates of the requested datapoint. If set to None, will instead return the profile at all locations. Default is None
- **num_wavelengths** (*int, optional*) – Number of wavelength values to compute the fit intensity at. These values will be equally spaced and span the entire fit window. If set to None, will use the observed wavelength values. Default is None.
- **wave_range** (*list or array, optional*) – List or array with two elements giving the wavelength range to use for calculating the intensity profile.
- **use_mask** (*bool, optional*) – If set to True and num_wavelengths == None (i.e. intensity is computed at observed wavelength values), then apply the mask that was used for the fitting process to filter out bad data or observations outside of fit template range.

Returns

- **fit_wave** (numpy.ndarray) – Wavelength values
- **fit_inten** (numpy.ndarray) – Fit intensity values

get_map(*component=0*, *measurement='intensity'*, ***kwargs*)

Return an EISMap of either intensity, velocity, or width

Parameters

- **component** (*int, optional*) – Integer number of the fit gaussian. Default is 0 (first line in fit)
- **measurement** (*string, optional*) – Measured parameter to create the map for. Choose from "intensity", "velocity", or "width". Default is "intensity"
- ****kwargs** (*dict or keywords, optional*) – Optional keywords to pass to EISMap

Returns

output_map – EISMap of the requested measurement.

Return type

EISMap class instance

get_params(*component=None*, *param_name=None*, *coords=None*, *casefold=False*)

Extract parameters values by component number, name, or pixel coords

Parameters

- **component** (*int or list, optional*) – Integer number (or list of ints) of the functional component(s). If set to None, will return all parameters that match "param_name". Default is None.
- **param_name** (*str, optional*) – String name of the requested parameter. If set to None, will not filter based on paramater name. Default is None
- **coords** (*list or tuple, optional*) – Array (Y, X) coordinates of the requested datapoint. If set to None, will instead return the parameters at all locations. Default is None
- **casefold** (*bool, optional*) – If set to True, will ignore case when extracting parameters by name. Default is False.

Returns

- **param_vals** (numpy.ndarray) – Parameter values
- **param_errs** (numpy.ndarray) – Estimated parameter errors

plot_fov(*end_time*, *color='red'*, *lw=1*, *ls='-'*)

Return a patch of the raster FOV for plotting on an image.

Parameters

- **end_time** (any time format that can be parsed by parse_time) – Time at which the rotated EIS pointing is desired. Usually should be after the first observation in the EIS raster.
- **color** (*str, optional*) – Color of the output rectangle. Default is "red".
- **lw** (*int, optional*) – Linewidth of the output rectangle. Default is 1.
- **ls** (*str, optional*) – Line style of the output rectangle. Default is "-" (solid line).

Returns

rect – Matplotlib Rectangle patch. Useful for plotting the EIS FOV on a context image taken with a different instrument.

Return type

Rectangle

remove_radcal()

Remove the applied radiometric calibration and convert data to counts

Returns
output_cube – A new EISFitResult class instance containing the photon count data

Return type
EISFitResult class instance

rot_fov(*end_time*)

Return pointing information for the raster rotated to the input time.

Parameters
end_time (any time format that can be parsed by parse_time) – Time at which the rotated EIS pointing is desired. Usually should be after the first observation in the EIS raster.

Returns
fov – Dictionary with the estimated EIS raster center coords and FOV. This is calculated using the SunPy function solar_rotate_coordinate

Return type
dict

shift2wave(*array*, *wave=195.119*)

Shift an array from this fit to the desired wavelength

EISFitTemplate

class eispac.core.**EISFitTemplate**(*filename*, *template*, *parinfo*)

Bases: object

Representation of fitting parameters for a particular line or lines

Parameters

- **filename** (str or pathlib.Path) – Path to fitting template file
- **template** (*dict*) – Dictionary of template parameters
- **parinfo** (*list*) – List of fitting parameters where each entry is a dict

Attributes Summary

central_wave	Wavelength value in the center of the template wavelength range
funcinfo	List of dicts specifying each subcomponent function used in the template

Methods Summary

get_funcinfo(template)	Return a list of dictionaries where each entry describes the parameters for one of the fitting basis functions
read_template(filename)	Create *EISFitTemplate* from template file

Attributes Documentation

central_wave

Wavelength value in the center of the template wavelength range

funcinfo

List of dicts specifying each subcomponent function used in the template

Methods Documentation

static get_funcinfo(*template*)

Return a list of dictionaries where each entry describes the parameters for one of the fitting basis functions

Parameters
template (*list*) –

Returns
funcinfo

Return type
list

classmethod read_template(*filename*)

Create *EISFitTemplate* from template file

Parameters
filename (str or pathlib.Path) – Path to template file

Returns
cls – Object containing the fit template

Return type
EISFitTemplate class instance

EISMap

class eispac.core.**EISMap**(*data*, *header=None*, ***kwargs*)

Bases: GenericMap

EIS fit parameter map.

The EUV Imaging Spectrometer (EIS) is part of the Hinode mission and was sponsored by the Japan Aerospace Exploration Agency (JAXA), the United Kingdom Space Agency (UKSA), and National Aeronautics and Space Administration (NASA) with contributions from ESA and Norway. Hinode was launched on September 22, 2006 at 21:36 UTC from the Uchinoura Space Center in Japan and continues to operate. EIS observes two wavelength ranges in the extreme ultraviolet, 171—212 Å and 245—291 Å with a spectral resolution of about 22 mÅ and a plate scale of 100 per pixel.

This data structure is designed to hold the fit parameters derived from multi-gaussian spectral fits to level 1, wavelength-resolved EIS rasters. These maps can contain the intensity, doppler velocity, or line width.

Notes

Measurement errors are stored in a binary table. To load them correctly, you must pass the .fits file directly to `eispac.EISMap` instead of using sunpy.map.Map

References

- `**Instrument Paper: Culhane, J. L., Harra, L. K., James, A. M., et al.**
  2007, Sol. Phys., 243, 19`_

Notes

A number of the properties of this class are returned as two-value named tuples that can either be indexed by position ([0] or [1]) or be accessed by the names (.x and .y) or (.axis1 and .axis2). Things that refer to pixel axes use the `.x`, `.y` convention, where x and y refer to the FITS axes (x for columns y for rows). Spatial axes use `.axis1` and `.axis2` which correspond to the first and second axes in the header. `axis1` corresponds to the coordinate axis for `x` and `axis2` corresponds to `y`.

This class assumes that the metadata adheres to the FITS 4 standard. Where the CROTA2 metadata is provided (without PC_ij) it assumes a conversion to the standard PC_ij described in section 6.1 of . Calabretta & Greisen (2002)

Warning: If a header has CD_ij values but no PC_ij values, CDELT values are required for this class to construct the WCS. If a file with more than two dimensions is feed into the class, only the first two dimensions (NAXIS1, NAXIS2) will be loaded and the rest will be discarded.

Attributes Summary

date	Image observation time.
date_average	Average time of the image acquisition.
date_end	Time of the end of the image acquisition.
date_start	Time of the beginning of the image acquisition.
measurement	Measurement wavelength.
nickname	An abbreviated human-readable description of the map-type; part of the Helioviewer data model.
observatory	Observatory or Telescope name.
processing_level	Returns the FITS processing level if present.
spatial_units	Image coordinate units along the x and y axes (i.e.
wavelength	Wavelength of the observation.
waveunit	The `Unit` of the wavelength of this observation.

Methods Summary

is_datasource_for(data, header, **kwargs)	Determines if header corresponds to an EIS image.

Attributes Documentation

date

date_average

date_end

date_start

measurement

nickname

observatory

processing_level

spatial_units

wavelength

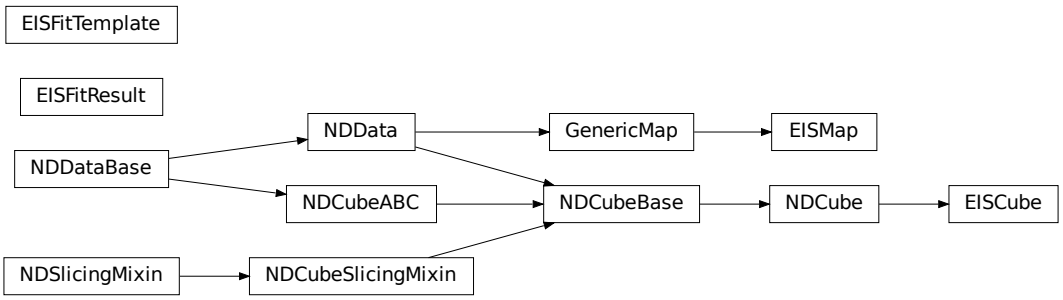
waveunit

Methods Documentation

classmethod is_datasource_for(*data*, *header*, ***kwargs*)

Determines if header corresponds to an EIS image. Used to register EISMap with the sunpy.map.Map factory.

5.1.3 Class Inheritance Diagram



5.2 eispac download

5.2.1 Functions

download_db([download_dir, source])	Download the offical EIS as-run SQLite database
run_eis_catalog([db_dir])	Launch the "eis_catalog" GUI tool""

download_db

eispac.download.**download_db**(*download_dir=None*, *source='nrl'*)

Download the offical EIS as-run SQLite database

Parameters

download_dir (str or pathlib.Path object, optional) – Local directory where the database should be saved. If there is already an EIS as-run database in the target directory, it will be overwritten. Defaults to the same directory as this function (e.g. ../eispac/download/).

Returns

local_filepath – Full filepath to the downloaded database. If the download failed, for any reason, the filename will end with .part (any exisiting database will NOT be overwirtten with a partial or failed download).

Return type

str

run_eis_catalog

eispac.download.**run_eis_catalog**(*db_dir=None*)

Launch the "eis_catalog" GUI tool""

Parameters

db_dir (*str or pathlib.Path object, optional*) – Directory containing the "eis_cat.sqlite" file. If None, EISPAC will for the catalog in an existing SSW installation or ask if you want to download it. Default is None. Note: any filename included with the path will be ignored (please do not rename your local catalog).

Return type

None

5.2.2 Classes

download_hdf5_data([filename, source, ...])	An object for downloading EIS HDF5 level1 files

download_hdf5_data

class eispac.download.**download_hdf5_data**(*filename=None*, *source='nrl'*, *local_top='data_eis'*, *datetree=False*, *nodata=False*, *nohead=False*, *overwrite=False*, *headonly=False*, *max_conn=2*)

Bases: `object`

An object for downloading EIS HDF5 level1 files

input is parsed to construct the remote and local filenames, curl is spawned to download the files (what if it doesn't exist?), if files exist locally they are skipped.

Parameters

- **filename** (`str` or `list`) – An EIS filename. Such as, eis_l0_20200311_213413.fits, eis_l1_20200311_213413.fits.gz, /some_path/eis_l0_20200311_213413.fits. Can be a single path or a list of paths
- **datetree** (`bool`) – Create a local path organized by date (YYYY/MM/DD)
- **local_top** (`str`) – Top of the local path (e.g., data_eis)
- **nodata** (`bool`) – Don't download data files
- **nohead** (`bool`) – Don't download head files
- **overwrite** (`bool`) – Download even if file exists locally
- **headonly** (`bool`) – Equivalent to `nodata` + `overwrite`
- **max_conn** (*int*) – Max number of download connections that parfive will use

Methods Summary

check_local_dir(local_file)	check if local dir exits.
construct_filenames(input_filename)	convert input into remote and local filenames
download()	download data and head files, unless told not to
download_file(remote_filepath, local_dir, ...)	Use parfive to download the file
parse_input_filename(input_filename)	convert an eis fits filename into an hdf5 filename; extract year, month, day
process_input(this_input)	input can be a filename or a list of filenames

Methods Documentation

check_local_dir(*local_file*)

check if local dir exits. If not, create it

construct_filenames(*input_filename*)

convert input into remote and local filenames

download()

download data and head files, unless told not to

download_file(*remote_filepath*, *local_dir*, *local_name*)

Use parfive to download the file

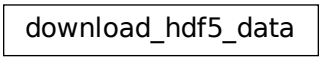
parse_input_filename(*input_filename*)

convert an eis fits filename into an hdf5 filename; extract year, month, day

process_input(*this_input*)

input can be a filename or a list of filenames

5.2.3 Class Inheritance Diagram



5.3 eispac extern

This subpackage contains third-party code that is included in `eispac` for convenience or efficiency.

5.3.1 Classes

mpfit(fcn[, xall, functkw, parinfo, ftol, ...])	Perform Levenberg-Marquardt least-squares minimization, based on MINPACK-1.

mpfit

class eispac.extern.**mpfit**(*fcn*, *xall=None*, *functkw={}*, *parinfo=None*, *ftol=1e-10*, *xtol=1e-10*, *gtol=1e-10*, *damp=0.0*, *maxiter=200*, *factor=100.0*, *nprint=1*, *iterfunct='default'*, *iterkw={}*, *nocovar=0*, *rescale=0*, *autoderivative=1*, *quiet=0*, *diag=None*, *epsfcn=None*, *debug=0*)

Bases: `object`

Perform Levenberg-Marquardt least-squares minimization, based on MINPACK-1.

Parameters

- **fcn** (*function*) – The function to be minimized. The function should return the weighted deviations between the model and the data, as described above.
- **xall** (*array-like*) – An array of starting values for each of the parameters of the model. The number of parameters should be fewer than the number of measurements. This parameter is optional if the parinfo keyword is used (but see parinfo). The parinfo keyword provides a mechanism to fix or constrain individual parameters.
- **autoderivative** (*{1, 0}*) –
- **(1)** (*If this is set*) – automatically via a finite differencing procedure. If not set (0),

- **computed** (*derivatives of the function will be*) – automatically via a finite differencing procedure. If not set (0),
- **your** (*then fcn must provide the (analytical) derivatives. To supply*) –
- **derivatives** (*own analytical*) – Default: set (=1)
- **autoderivative=0.** (*explicitly pass*) – Default: set (=1)
- **ftol** (*float, optional*) – A nonnegative input variable. Termination occurs when both the actual and predicted relative reductions in the sum of squares are at most ftol (and status is accordingly set to 1 or 3). Therefore, ftol measures the relative error desired in the sum of squares.

  Default: 1E-10
- **functkw** (*dict, optional*) – A dictionary which contains the parameters to be passed to the user-supplied function specified by fcn via the standard Python keyword dictionary mechanism. This is the way you can pass additional data to your user-supplied function without using global variables.

  Consider the following example, if functkw =
  {'xval':[1.,2.,3.], 'yval':[1.,4.,9.], 'errval':[1.,1.,1.]}

  then the user-supplied function should be declared like this:
  def myfunct(p, fjac=None, xval=None, yval=None, errval=None):

  Default: {}, No extra parameters are passed to the function
- **gtol** (*float, optional*) – A nonnegative input variable. Termination occurs when the cosine of the angle between fvec and any column of the jacobian is at most gtol in absolute value (and status is accordingly set to 4). Therefore, gtol measures the orthogonality desired between the function vector and the columns of the jacobian.

  Default: 1e-10
- **iterkw** (*dict, optional*) – The keyword arguments to be passed to iterfunct via the dictionary keyword mechanism. This should be a dictionary and is similar in operation to FUNCTKW.

  Default: {}, No arguments are passed.
- **iterfunct** (*function*) – The name of a function to be called upon each NPRINT iteration of the MPFIT routine. It should be declared in the following way:

  def iterfunct(myfunct, p, iter, fnorm, functkw=None, parinfo=None,
  quiet=0, dof=None, [iterkw keywords here])

  # perform custom iteration update

  iterfunct must accept all three keyword parameters (FUNCTKW, PARINFO and QUIET).

  myfunct - The user-supplied function to be minimized, p - The current set of model parameters iter - The iteration number functkw - The arguments to be passed to myfunct. fnorm - The chi-squared value. quiet - Set when no textual output should be printed. dof - The number of degrees of freedom, normally the number of

  points less the number of free parameters.

  See below for documentation of parinfo.

  In implementation, iterfunct can perform updates to the terminal or graphical user interface, to provide feedback while the fit proceeds. If the fit is to be stopped for any reason, then iterfunct should return

- **None** (*a status value between -15 and -1. Otherwise it should return*) – (e.g. no return statement) or 0. In principle, iterfunct should probably
- **values** (*not modify the parameter*) –
- **the** (*because it may interfere with*) –
- **iterfunct=None** (*algorithm's stability. In practice it is allowed. Set*) –
- **internal** (*if there is no user-defined routine and you don't want the*) –
- **called.** (*default routine be*) – Default: an internal routine is used to print the parameter values.
- **maxiter** (*int, optional*) – The maximum number of iterations to perform. If the number is exceeded, then the status value is set to 5 and MPFIT returns. Default: 200 iterations
- **nocovar** (*{0, 1}*) – Set this keyword to prevent the calculation of the covariance matrix before returning (see COVAR) Default: clear (=0), The covariance matrix is returned
- **nprint** (*int, optional*) – The frequency with which iterfunct is called. A value of 1 indicates that iterfunct is called with every iteration, while 2 indicates every other iteration, etc. Note that several Levenberg-Marquardt attempts can be made in a single iteration. Default: 1
- **parinfo** (*list of dicts, optional*) –

  Provides a mechanism for more sophisticated constraints to be placed on
  parameter values. When parinfo is not passed, then it is assumed that all parameters are free and unconstrained. Values in parinfo are never modified during a call to MPFIT. PARINFO should be a list of

- **dictionaries** –
- **can** (*one list entry for each parameter. The dictionary*) –
- **insensitive)** (*have the following keys (all keys are optional and case*) – 'value' : float the starting parameter value (see also the XALL parameter).

  'fixed' : {0, 1} If set (1), the parameter value will be held fixed. Fixed parameters are not varied by MPFIT, but are passed on to MYFUNCT for evaluation.

  'limited' : two-element int array. If the first/second element is set (1), then the parameter is bounded on the lower/upper side. A parameter can be bounded on both sides. Both LIMITED and LIMITS must be given together.

  'limits' : two-element float array. Gives the parameter limits on the lower and upper sides, respectively. A value will only be used as a limit if the corresponding value of LIMITED is set (=1). Both LIMITED and LIMITS must be given together.

  'parname' : str Name of the parameter. The fitting code of MPFIT does not use this tag in any way. However, the default iterfunct will print the parameter name, if available.

  'step' : float Step size to be used in calculating the numerical derivatives. If set to zero, then the step size is computed automatically. Ignored when AUTODERIVATIVE=0.

  'mpside' : {0, 1, -1, 2} The sidedness of the finite difference when computing

    numerical derivatives. This field can take four values:

      0 - one-sided derivative computed automatically 1 - one-sided derivative (f(x+h) - f(x))/h -1 - one-sided derivative (f(x) - f(x-h))/h 2 - two-sided derivative (f(x+h) - f(x-h))/(2*h)

    Where H is the STEP parameter described above. The "automatic"

one-sided derivative method will chose a direction for the finite difference which does not violate any constraints. The other methods do not perform this check. The two-sided method is in principle more precise, but requires twice as many function evaluations. Default: 0.

'mpmaxstep' : float The maximum change to be made in the parameter value. During the fitting process, the parameter will never be changed by more than this value in one iteration. A value of 0 indicates no maximum. Default: 0.

'tied' : str String expression which "ties" the parameter to other free or fixed parameters. Any expression involving constants and the parameter array "P" are permitted. Example: if parameter 2 is always to be twice parameter 1 then use the following:

parinfo[2].tied = '2 * p[1]'.
Since they are totally constrained, tied parameters are

considered to be fixed; no errors are computed for them. NOTE: the PARNAME can't be used in expressions.

'mpprint' : {1, 0} If set to 1, then the default iterfunct will print the

parameter value. If set to 0, the parameter value will not

be printed. This tag can be used to selectively print only a few parameter values out of many. Default: 1 (all parameters printed)

Default value: None, All parameters are free and unconstrained.

- **quiet** (*{0, 1}*) – Set this keyword = 1 when no textual output should be printed by MPFIT
- **damp** (*float, optional*) – A scalar number, indicating the cut-off value of residuals where "damping" will occur. Residuals with magnitudes greater than this number will be replaced by their hyperbolic tangent. This partially mitigates the so-called large residual problem inherent in least-squares solvers (as for the test problem CURVI, http://www.maxthis.com/curviex.htm). A value of 0 indicates no damping.
- **Note** (*DAMP doesn't work with autoderivative=0. Default: 0*) –
- **xtol** (*float, optional*) –

  A nonnegative input variable. Termination occurs when the relative error
  between two consecutive iterates is at most xtol (and status is accordingly set to 2 or 3). Therefore, xtol measures the relative error desired in the approximate solution. Default: 1E-10

Returns

The results are attributes of this class, e.g. mpfit.status, mpfit.errmsg, mpfit.params, npfit.niter, mpfit.covar.

.status
[int] An integer status code is returned. All values greater than zero

can represent success (however .status == 5 may indicate failure to
converge). It can have one of the following values:

-16
A parameter or function value has become infinite or an undefined number. This is usually a consequence of numerical overflow in the user's model function, which must be avoided.

-15 to -1
These are error codes that either MYFUNCT or iterfunct may return to terminate the

fitting process. Values from -15 to -1 are reserved for the user functions and will not clash with MPFIT.

0 Improper input parameters.

1 Both actual and predicted relative reductions in the sum of squares are at most ftol.

2 Relative error between two consecutive iterates is at most xtol

3 Conditions for status = 1 and status = 2 both hold.

4 The cosine of the angle between fvec and any column of the jacobian is at most gtol in absolute value.

5 The maximum number of iterations has been reached.

6 ftol is too small. No further reduction in the sum of squares is possible.

7 xtol is too small. No further improvement in the approximate solution x is possible.

8 gtol is too small. fvec is orthogonal to the columns of the jacobian to machine precision.

.fnorm

[float] The value of the summed squared residuals for the returned parameter values.

.covar

[array-like] The covariance matrix for the set of parameters returned by MPFIT. The matrix is NxN where N is the number of parameters. The square root of the diagonal elements gives the formal 1-sigma statistical errors on the parameters if errors were treated "properly" in fcn. Parameter errors are also returned in .perror.

To compute the correlation matrix, pcor, use this example:
cov = mpfit.covar pcor = cov * 0. for i in range(n):

for j in range(n):
pcor[i,j] = cov[i,j]/sqrt(cov[i,i]*cov[j,j])

If nocovar is set or MPFIT terminated abnormally, then .covar is set to a scalar with value None.

.errmsg

[str] A string error or warning message is returned.

.nfev

[int] The number of calls to MYFUNCT performed.

.niter

[int] The number of iterations completed.

.perror

[array-like] The formal 1-sigma errors in each parameter, computed from the covariance matrix. If a parameter is held fixed, or if it touches a boundary, then the error is reported as zero.

If the fit is unweighted (i.e. no errors were given, or the weights were uniformly set to unity), then .perror will probably not represent the true parameter uncertainties.

If you can assume that the true reduced chi-squared value is unity – meaning that the fit is implicitly assumed to be of good quality – then the estimated parameter uncertainties can be computed by scaling .perror by the measured chi-squared value.

dof = len(x) - len(mpfit.params) # deg of freedom # scaled uncertainties pcerror = mpfit.perror * sqrt(mpfit.fnorm / dof)

Return type
mpfit class object

Attributes Summary

blas_enorm32
blas_enorm64

Methods Summary

calc_covar(rr[, ipvt, tol])
call(fcn, x, functkw[, fjac])
defiter(fcn, x, iter[, fnorm, functkw, ...])
enorm(vec)
fdjac2(fcn, x, fvec[, step, ulimited, ...])
lmpar(r, ipvt, diag, qtb, delta, x, sdiag[, par])
parinfo([parinfo, key, default, n])
qrfac(a[, pivot])
qrsolv(r, ipvt, diag, qtb, sdiag)
tie(p[, ptied])

Attributes Documentation

blas_enorm32 = <fortran dnrm2>

blas_enorm64 = <fortran dnrm2>

Methods Documentation

calc_covar(*rr*, *ipvt=None*, *tol=1e-14*)

call(*fcn*, *x*, *functkw*, *fjac=None*)

defiter(*fcn*, *x*, *iter*, *fnorm=None*, *functkw=None*, *quiet=0*, *iterstop=None*, *parinfo=None*, *format=None*, *pformat='%.10g'*, *dof=1*)

enorm(*vec*)

fdjac2(*fcn*, *x*, *fvec*, *step=None*, *ulimited=None*, *ulimit=None*, *dside=None*, *epsfcn=None*, *autoderivative=1*, *functkw=None*, *xall=None*, *ifree=None*, *dstep=None*)

lmpar(*r*, *ipvt*, *diag*, *qtb*, *delta*, *x*, *sdiag*, *par=None*)

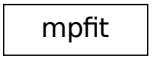
parinfo(*parinfo=None*, *key='a'*, *default=None*, *n=0*)

qrfac(*a*, *pivot=0*)

qrsolv(*r*, *ipvt*, *diag*, *qtb*, *sdiag*)

tie(*p*, *ptied=None*)

5.3.2 Class Inheritance Diagram



5.4 eispc instr

5.4.1 Functions

calc_read_noise(input_wave)	Calculate the read noise counts for a given wavelength spectrum
calc_velocity(observed_wave, rest_wave[, ...])	Calculate the Doppler velocity of a given line profile
ccd_offset(wavelength)	Calculate the spatial offset of a line relative to He II 256 Å

calc_read_noise

eispac.instr.**calc_read_noise**(*input_wave*)

Calculate the read noise counts for a given wavelength spectrum

Parameters
input_wave (float, list, tuple, or ndarray) – Wavelength values to compute the read noise at

Returns
read_noise_counts – read noise in units of [photon counts]. Will have the same dimensions as the input_wave array.

Return type
float or ndarray

calc_velocity

eispac.instr.**calc_velocity**(*observed_wave*, *rest_wave*, *corr_method='column'*)

Calculate the Doppler velocity of a given line profile

Parameters

- **observed_wave** (list, tuple, or ndarray) – Observed centroid wavelength values for a given spectral line in units of [Angstrom]
- **rest_wave** (*float or str*) – Rest wavelength value in units of [Angstrom]. If given a string, will check to see if it is a spectral line ID and, if it is, will try to extract the rest wavelength.
- **corr_method** (*str, optional*) – Method used to roughly correct for spacecraft temperature variations. Choose from 'column', 'image', or None. Default is 'column'

Returns

velocity –

Doppler velocity in units of [km/s]. Computed using the equation,
vel = c*(obs_wave - rest_wave)/rest_wave

Return type
ndarray

ccd_offset

eispac.instr.**ccd_offset**(*wavelength*)

Calculate the spatial offset of a line relative to He II 256 Å

Spatial offset of the specified wavelength relative to He II 256 Å. If you see a feature in the He II 256 image at coordinate Y, then the corrected coordinate Y' for any other wavelength λ is,

$$Y' = Y - o(\lambda),$$

where $o(\lambda)$ is the CCD offset. Note that the spatial coordinate system for EIS is evalulated for He II 256 Å by cross-correlating with SOT.

The value of `ccd_offset` represents the number of pixels that a feature seen in a wavelength sits above the He II 256 Å image on the CCD.

The following goes into the calculation of EIS CCD offset:

- the tilt of the grating is assumed to be linear with wavelength
- the tilt for SW was derived by [young09]
- the tilt for LW was *assumed* to be the same as for SW
- the offset between SW and LW was measured by co-aligning images from Fe VIII 185.21 Å and Si VII 275.35 Å

Note: This routine is a (nearly) verbatim translation of the IDL version of eis_ccd_offset written by Peter Young. The above documentation has been adapted from that routine.

Parameters

wavelength (*array_like*) – Wavelength of the spectral line(s) of interest in units of [Å]

Returns

offset – The spatial offset between the specified wavelength and He II 256.32. The value represents how many pixels above He II 256.32 the specified wavelength sits on the EIS detectors.

Return type

ndarray

References

5.5 eispc net

5.5.1 Classes

EISClient()	Provides access to the level 1 EIS data in HDF5 and FITS format.

EISClient

class eispac.net.**EISClient**

Bases: GenericClient

Provides access to the level 1 EIS data in HDF5 and FITS format.

This data is hosted by the Naval Research Laboratory.

Examples

```
>>> from sunpy.net import Fido, attrs as a
>>> import eispac.net
>>> from eispac.net.attrs import FileType
>>> results = Fido.search(a.Time('2020-11-09 00:00:00','2020-11-09 01:00:00'),
...                       a.Instrument('EIS'),
...                       a.Physobs.intensity,
...                       a.Source('Hinode'),
...                       a.Provider('NRL'),
```

(continues on next page)

(continued from previous page)

```
...                       a.Level('1'))
>>> results
<sunpy.net.fido_factory.UnifiedResponse object at ...>
Results from 1 Provider:

3 Results from the EISClient:
Source: https://eis.nrl.navy.mil/

       Start Time               End Time        ... Level   FileType
----------------------- ----------------------- ... ----- -----------
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1   HDF5 data
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1 HDF5 header
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1        FITS


>>> results = Fido.search(a.Time('2020-11-09 00:00:00','2020-11-09 01:00:00'),
...                       a.Instrument('EIS'),
...                       a.Physobs.intensity,
...                       a.Source('Hinode'),
...                       a.Provider('NRL'),
...                       a.Level('1'),
...                       FileType('HDF5 header'))
>>> results
<sunpy.net.fido_factory.UnifiedResponse object at ...>
Results from 1 Provider:

1 Results from the EISClient:
Source: https://eis.nrl.navy.mil/

       Start Time               End Time        ... Level   FileType
----------------------- ----------------------- ... ----- -----------
2020-11-09 00:10:12.000 2020-11-09 00:10:12.999 ...     1 HDF5 header
```

Attributes Summary

baseurl_fits	
baseurl_hdf5	
info_url	This should return a string that is a URL to the data server or documentation on the data being served.
pattern_fits	
pattern_hdf5	

Methods Summary

post_search_hook(i, matchdict)	Helper function used after `search()` which makes the extracted metadata representable in a query response table.
register_values()	This enables the client to register what kind of Attrs it can use directly.
search(*args, **kwargs)	Query this client for a list of results.

Attributes Documentation

baseurl_fits =
'https://eis.nrl.navy.mil/level1/fits/%Y/%m/%d/eis_er_%Y%m%d_%H%M%S.fits'

baseurl_hdf5 =
'https://eis.nrl.navy.mil/level1/hdf5/%Y/%m/%d/eis_%Y%m%d_%H%M%S.(\\w){4}.h5'

info_url

pattern_fits = '{}/{year:4d}/{month:2d}/{day:2d}/
eis_er_{:8d}_{hour:2d}{minute:2d}{second:2d}.{FileType}'

pattern_hdf5 = '{}/{year:4d}/{month:2d}/{day:2d}/
eis_{:8d}_{hour:2d}{minute:2d}{second:2d}.{FileType}'

Methods Documentation

post_search_hook(*i*, *matchdict*)

Helper function used after `search()` which makes the extracted metadata representable in a query response table.

Parameters

- **exdict** (dict) – Represents metadata extracted from files.
- **matchdict** (dict) – Contains attr values accessed from `register_values()` and the search query itself.

Returns

rowdict – An Ordered Dictionary which is used by `QueryResponse` to show results.

Return type

OrderedDict

classmethod register_values()

This enables the client to register what kind of Attrs it can use directly.

Returns

A dictionary with key values of Attrs and the values are a tuple of ("Attr Type", "Name", "Description").

Return type

dict

search(**args*, ***kwargs*)

Query this client for a list of results.

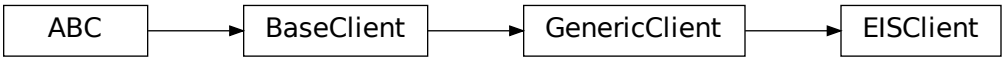
Parameters

- ***args** (tuple) – sunpy.net.attrs objects representing the query.
- ****kwargs** (dict) – Any extra keywords to refine the search.

Return type

A QueryResponse instance containing the query result.

5.5.2 Class Inheritance Diagram



5.5.3 eispac.net.attrs Module

Classes

FileType(value)	Specifies the type of EIS level 1 file

FileType

class eispac.net.attrs.**FileType**(*value*)

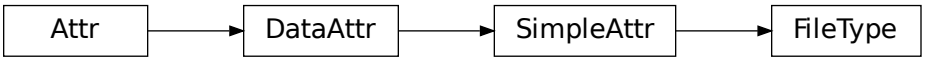
Bases: SimpleAttr

Specifies the type of EIS level 1 file

Parameters

value (str) – Possible values are "HDF5 data" or "HDF5 header" to retrieve the data and header files, respectively, in HDF5 format, or "FITS" to retrieve the FITS files. Inputs are not case sensitive.

Class Inheritance Diagram



5.6 eispac templates

5.6.1 Functions

run_eis_browse_templates([filepath])	Launch the "eis_browse_templates" GUI tool""

run_eis_browse_templates

eispac.templates.**run_eis_browse_templates**(*filepath=None*)

Launch the “eis_browse_templates” GUI tool””

Parameters

filepath (*str or pathlib.Path object, optional*) – Filepath to an EIS level-1 HDF5 data or header file. If None, you may broswe for and select the observation file from within the GUI. Default is None.

Return type

None

5.6.2 Classes

EISTemplateLocator([filename_head, verbose, ...])	Helper class used by the "eis_browse_templates" GUI

EISTemplateLocator

class eispac.templates.**EISTemplateLocator**(*filename_head=None*, *verbose=False*, *ignore_local=True*)

Bases: object

Helper class used by the “eis_browse_templates” GUI

Finds fit templates matching all data windows in a given EIS observation and generates lists of the filepaths. Not intended for direct use. For a more user-friendly function with similar capabilities, see *match_templates*

Parameters

- **filename_head** (*str, optional*) – Filepath to a EIS Level-1 HDF5 file. Both data and header filepaths are accepted.

- **verbose** (*bool, optional*) – If set to True, will automatically print a list of all found templates. Default is False.
- **ignore_local** (*bool, optional*) – If set to True, will look for templates in either SSW or those distributed with EISPAC. If set to False, will instead ONLY look for templates in a local directory named either "eis_template_dir" or "eis_fit_templates". Default is True.

Methods Summary

find_templates()	Find all fit template files available
make_text_list()	Generate a text list of all templates found
match_templates()	Match each observation window and with all relevant fit templates
parse_input_filename(input_filename)	Read an EIS level-1 filename and determine the header filepath
print_text_list()	Print the list of found templates
read_wininfo()	"Read the window information from the header file

Methods Documentation

find_templates()

Find all fit template files available

make_text_list()

Generate a text list of all templates found

match_templates()

Match each observation window and with all relevant fit templates

parse_input_filename(*input_filename*)

Read an EIS level-1 filename and determine the header filepath

print_text_list()

Print the list of found templates

read_wininfo()

"Read the window information from the header file

5.6.3 Class Inheritance Diagram

5.7 eispac util

5.7.1 Functions

calc_intensity_range(intensity[, pmin, ...])	Compute an intensity range for an image based on the histogram.
rot_xy(xcen, ycen, start_time, end_time)	Compute the rotation of a point on the Sun using solar_rotate_coordinate
scale_intensity(intensity, irange[, log])	Scale an array of intensities to have values in the range of 0 to 1.

calc_intensity_range

eispac.util.**calc_intensity_range**(*intensity*, *pmin=1*, *pmax=99*, *lower=0.01*, *imin=10*)

Compute an intensity range for an image based on the histogram. Uses np.percentile.

Parameters

- **intensity** (*array-like*) – The intensity array. Can have any dimensions.
- **pmin** (*float, optional*) – Lower cutoff percentile. Must be a value between 0 and 100, inclusive. Default is 1.
- **pmax** (*float, optional*) – Upper cutoff percentile. Must be a value between 0 and 100, inclusive. Default is 99.
- **lower** (*float, optional*) – Scale factor to calculate the minimum intensity range value using imin = lower*imax. Default is 0.01
- **imin** (*float, optional*) – Limit on the minimum intensity range value. Default is 10

Returns

irange – two-element list of [imin, imax]

Return type

list

rot_xy

eispac.util.**rot_xy**(*xcen*, *ycen*, *start_time*, *end_time*)

Compute the rotation of a point on the Sun using solar_rotate_coordinate

Parameters

- **xcen** (*array-like*) – Solar-x in arcsec (scalar)
- **ycen** (*array-like*) – Solar y in arcsec (scalar)
- **start_time** (any time format that can be parsed by parse_time) – Obstime of the coordinate represented by xcen and ycen
- **end_time** (any time format that can be parsed by parse_time) –

Returns

new – Coordinate rotated by the time delta between start_time and end_time

Return type
SkyCoord

Examples

```
>>> new = rot_xy(0, 0, start_time='2021-JAN-01 00:00', end_time='2021-JAN-01 01:00')
>>> print(new.Tx, new.Ty)
9.47188arcsec 0.0809565arcsec
```

scale_intensity

eispac.util.**scale_intensity**(*intensity*, *irange*, *log=True*)

Scale an array of intensities to have values in the range of 0 to 1. Can be useful for plotting.

Parameters

- **intensity** (*array_like*) – The intensity array. Can have any dimensions.
- **irange** (*array_like*) – Min and max intensity values used for the scaling. Intensity values outside this range will be clipped to the nearest bound.
- **log** (*bool, optional*) – If set to True, will scale the log of the intensity values. Default is True.

Returns

scaled – Array of scaled intensity values. Will have the same dimensions as the input intensity array. The caling is calculated using the equation scaled = (intensity - imin) / (imax - imin), where imin and imax are specified by the input "irange" parameter.

Return type
array_like

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[young09] Young, P.R., et al., 2009, A&A, 495, 587

PYTHON MODULE INDEX

e

INDEX

G

I

L

M

N

O

P

Q

R

S

T

W