$\mathbf{ds}_{arpes_{plugin}}$

Oct 03, 2020

Welcome! `ds_arpes_plugin` is a python module that acts as a plugin to PIT, the python image tool provided by the data_slicer package. It connects the functionality provided by the arpys package to PIT.

CHAPTER 1

Usage

To load the plugin, run the following command inn the ipython console of PIT:

```
arpes = mw.load_plugin('ds_arpes_plugin')
```

(Of course you can use any variable name of your choice instead of `arpes`, but be aware that if you set up PIT to autoload this plugin, it will be available as `arpes`).

You are now able to load ARPES data into PIT by doing:

```
arpes.open('<filename>')
```

Of course, this requires a respective dataloader for the dataset you're trying to load to exist.

After loading, the usual `D` namespace is available as `arpes.D`, so you can, for example, access the data as `arpes.D.data`. Of course, at the same time PIT holds the data in its usual location and can be accessed by `pit.get_data()`.

If the data isn't displayed the way you'd expect, try a `roll_axes()`.

Note: Datasets that represent single ARPES spectra (i.e. 2D data) are still loaded by `arpys` as 3D arrays where one dimension has length 1. Somehow, this causes buggy behaviour in PIT and not all functionality may be available.

Basically, `ds_arpes_plugin` provides you all the tools of the arpys module through the `arpes.pp` and `arpes.dl` instances. Additionally, some convenience methods exist, most notably *`arpes.a2k`* for convenient angle to k space conversion. Check the *full code reference* for more.

CHAPTER 2

ds_arpes_plugin

2.1 ds_arpes_plugin package

2.1.1 Submodules

2.1.2 ds_arpes_plugin.ds_arpes_plugin module

exception ds_arpes_plugin.ds_arpes_plugin.**DatasetError**
Bases: Exception

Error raised when the type of data found does not conform to our expectations.

class ds_arpes_plugin.ds_arpes_plugin.**ARPES_Plugin**(**args*, ***kwargs*)
Bases: data_slicer.plugin.Plugin

A plugin which connects the analysis functionalities of the *aprys* module with PIT.

filename = '<missing filename>'

load_data(*filename*)
Load a set of ARPES data and bring it into PIT-friendly form. Also return the arpys data Namespace for inspection.

load(*filename*)
Load a set of ARPES data and bring it into PIT-friendly form. Also return the arpys data Namespace for inspection.

This is a convenience alias for *load_data*.

open(*filename*)
Load a set of ARPES data and bring it into PIT-friendly form. Also return the arpys data Namespace for inspection.

This is a convenience alias for *load_data*.

store (*filename*, *force=False*)
Store the data Namespace `D` in a pickle file *filename*. This can severely reduce loading times for certain filetypes.

dump (*filename*, *force=False*)
Store the data Namespace `D` in a pickle file *filename*. This can severely reduce loading times for certain filetypes.

This is a convenience alias for *store*.

a2k (*alpha_axis*, *beta_axis=None*, *dalpha=0*, *dbeta=0*, *orientation='horizontal'*, *work_func=4*, *units=0*, *hv=None*, *store=True*)
Convert the axes from angles to k-space. This updates the selected axes in the *pit.axes* and makes the change visible in the main plot. Notice that there will be no error message or anything if you happen to select nonsensical axes for *alpha_axis* and *beta_axis*, so check carefully if your result makes sense. The calculated KX and KY meshes (in the specified units) are stored in self.D, so the result can be retained when storing the data with *store*.

Parameters

alpha_axis	int; index of the axis containing the angles along the analyser slit. In PIT, 0 corresponds to the horizontal axis of the main plot, 1 to its vertical axis and 2 to the remaining 3rd axis.
beta_axis	int or None; index of the axis containing the angles perpendicular to the analyser slit. Can be left out (i.e. set to *None*) to only transform 1 axis. A constant value for the respective angle can then be specified with *dbeta*.
dalpha	float; angular offset along *alpha*.
dbeta	float; angular offset along *beta* or, if *beta_axis* is *None*, the value of *beta*.
orientation	str, must start with 'h' or 'v'; specifies the analyzer slit geometry (horizontal or vertical).
work_func	float; work function in eV.
units	float; toggle what units to use. - 0 corresponds to inverse Angstrom; - any nonzero value corresponds to units of pi/*units* (this is useful, e.g. to convert to units of pi/lattice_constant).
hv	float; used photon energy in eV. If not given, this will use the value stored in self.D (which is accurate in most cases, but not all, e.g. in photon-energy scans)
store	boolean; set to False to suppress storing the found KX and KY meshes directly in the data object *D*.

Returns

KX	array of shape (nkx, nky); mesh of k values in parallel direction in units of inverse Angstrom.
KY	array of shape (nkx, nky); mesh of k values in perpendicular direction in units of inverse Angstrom.

shift_axis (*shift*, *dim=0*, *store=True*)
Apply a linear *shift* to the axis along *dim*, both on PIT's data and in the *D* data object.

main_plot_normalize_per_segment (*dim=0*, *min=False*)
Apply the arpys function `normalize_per_segment` to the data in the main_plot and visualize the result.

Note: This result is not stored, does not affect other plots (like cut_plot and the x- and y-plots) and is lost the next time the main_plot is updated by any means. To create a more persisting result, see

normalize_per_segment

cut_plot_normalize_per_segment(*dim=0*, *min=False*)
Apply the arpys function normalize_per_segment to the data in the cut_plot and visualize the result.

Note: This result is not stored, does not affect other plots (like cut_plot and the x- and y-plots) and is lost the next time the cut_plot is updated by any means. To create a more persisting result, see *normalize_per_segment*

normalize_per_segment(*dim=0*, *min=False*)
Apply the arpys function normalize_per_segment to every slice along z.

Note: The result of this operation is stored, i.e. the dataset is updated. If you just want to have a quick look at what this operation might look like without applying it to the whole dataset, confer *main_plot_normalize_per_segment* or *cut_plot_normalize_per_segment*

apply_model(*model=<function ARPES_Plugin.<lambda>>*, *eps=0.1*)
Work in progress.

2.1.3 Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

Index