Quantum Metrology with Photoelectrons Vol. 3 *Analysis methodologies*

Paul Hockett

Nov 10, 2022

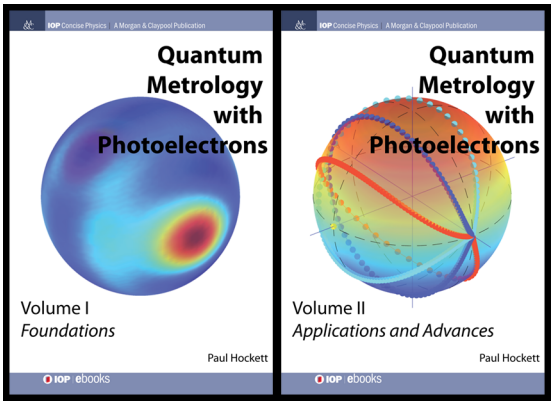
CONTENTS

Quantum Metrology with Photoelectrons Volume 3: *Analysis methodologies*, an open source executable book. This repository contains the source documents (mainly Jupyter Notebooks in Python) and notes for the book, as of Jan 2022 writing is in progress, and the current HTML build can be found online. The book is due to be finished in 2023, and will be published by IOP Press - see below for more details.

Series abstract

Photoionization is an interferometric process, in which multiple paths can contribute to the final continuum photoelectron wavefunction. At the simplest level, interferences between different final angular momentum states are manifest in the energy and angle resolved photoelectron spectra: metrology schemes making use of these interferograms are thus phase-sensitive, and provide a powerful route to detailed understanding of photoionization. In these cases, the continuum wavefunction (and underlying scattering dynamics) can be characterised. At a more complex level, such measurements can also provide a powerful probe for other processes of interest, leading to a more general class of quantum metrology built on phase-sensitive photoelectron imaging. Since the turn of the century, the increasing availability of photoelectron imaging experiments, along with the increasing sophistication of experimental techniques, and the availability of computational resources for analysis and numerics, has allowed for significant developments in such photoelectron metrology.

About the books



- Volume I covers the core physics of photoionization, including a range of computational examples. The material is presented as both reference and tutorial, and should appeal to readers of all levels. ISBN 978-1-6817-4684-5, http://iopscience.iop.org/book/978-1-6817-4684-5 (IOP Press, 2018)
- Volume II explores applications, and the development of quantum metrology schemes based on photoelectron measurements. The material is more technical, and will appeal more to the specialist reader. ISBN 978-1-6817-4688-3, http://iopscience.iop.org/book/978-1-6817-4688-3 (IOP Press, 2018)

Additional online resources for Vols. I & II can be found on OSF and Github.

- Volume III in the series will continue this exploration, with a focus on numerical analysis techniques, forging a closer link between experimental and theoretical results, and making the methodologies discussed directly accessible via new software. The book is due for publication by IOP due in 2023; this volume is also open-source, with a live HTML version at https://phockett.github.io/Quantum-Metrology-with-Photoelectrons-Vol3/ and source available at https://github.com/phockett/Quantum-Metrology-with-Photoelectrons-Vol3.

For some additional details and motivations (including topical video), see the ePSdata project.

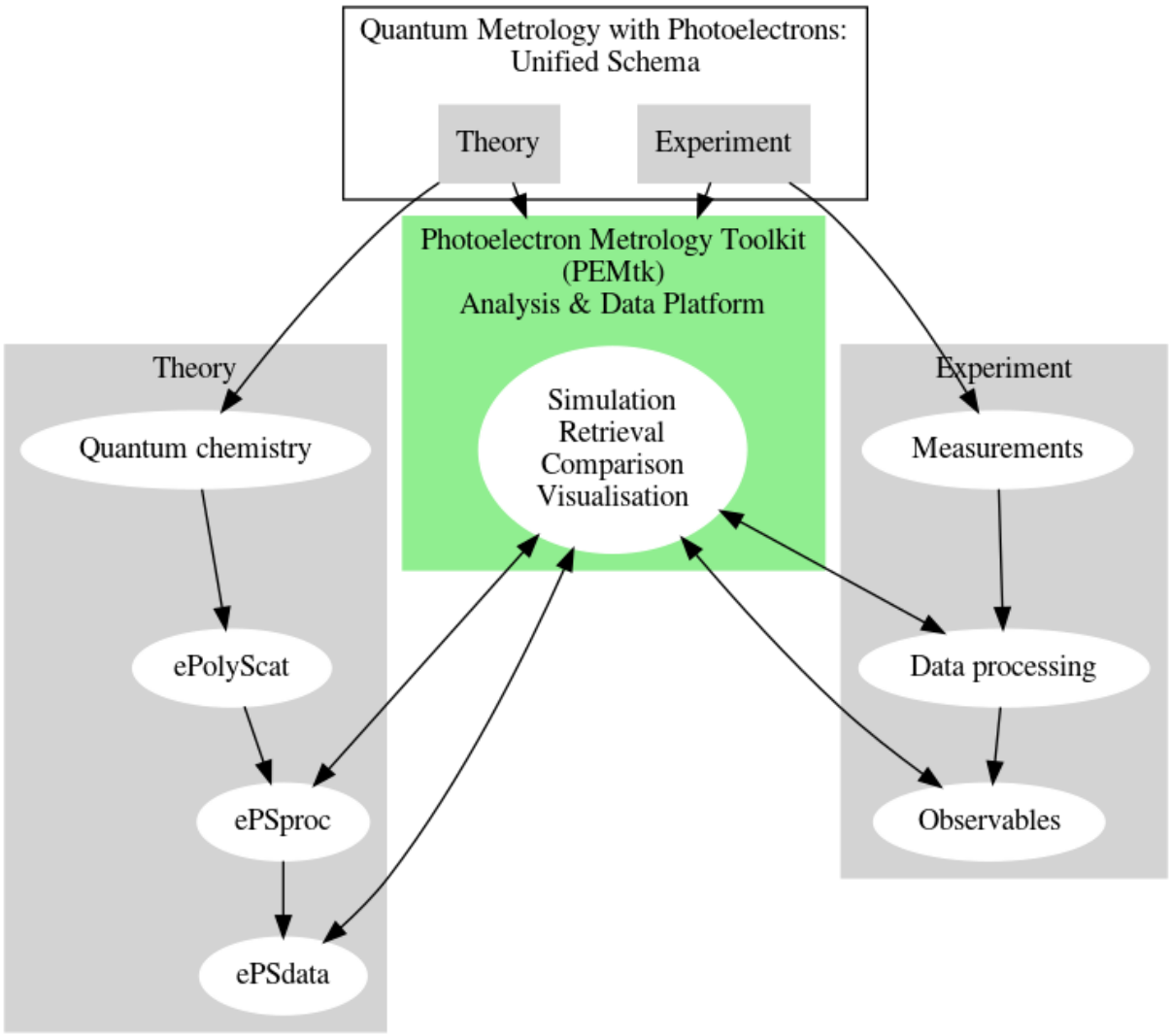
Technical details

This repository contains:

- `doc-source`: the source documents (mainly Jupyter Notebooks in Python)
- `notes`: additional notes for the book,
- the `gh-pages` branch contains the current HTML build, also available at https://phockett.github.io/Quantum-Metrology-with-Photoelectrons-Vol3/

The project has been setup to use the Jupyter Book build-chain (which uses Sphinx on the back-end) to generate HTML and Latex outputs for publication from source Jupyter notebooks & markdown files.

The work *within* the book will make use of the Photoelectron Metrology Toolkit platform for working with experimental & theoretical data.



Running code examples

Each Jupyter notebook (`*.ipynb`) can be treated as a stand-alone computational document. These can be run/used/modified independently with an appropriately setup python environment (details to follow).

Building the book

The full book can also be built from source:

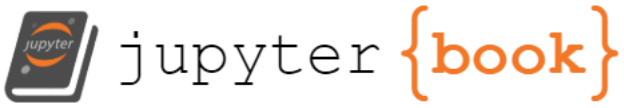
1. Clone this repository
2. Run `pip install -r requirements.txt` (it is recommended you do this within a virtual environment)
3. (Optional) Edit the books source files located in the `doc-source/` directory
4. Run `jupyter-book clean doc-source/` to remove any existing builds
5. For an HTML build:
    - Run `jupyter-book build doc-source/`
    - A fully-rendered HTML version of the book will be built in `doc-source/_build/html/`.
6. For a LaTex & PDF build:
    - Run `jupyter-book build doc-source/ --builder pdflatex`
    - A fully-rendered HTML version of the book will be built in `doc-source/_build/latex/`.

See https://jupyterbook.org/basics/building/index.html for more information.

Credits

This project is created using the open source Jupyter Book project and the executablebooks/cookiecutter-jupyter-book template.

To add: build env & main software packages (see automation for this…)



Part I

Frontmatter

CHAPTER

ONE

OVERVIEW

1.1 General overview

Vol. 3. will focus on analysis techniques for quantum metrology with photoelectrons, including:

- Interpreting experimental data.
- Extraction/reconstruction/determination of quantum mechanical properties (matrix elements, wavefunctions, density matrices) from experimental data.
- Comparison of experimental and theoretical data.
- New analysis methodologies & techniques.
- Introduction to newly-developed software platform (see below).

1.2 Provisional contents

1.2.1 Part 1: theory & software

General review & update of the topic, including recent theory developments.

1. Introduction

   a. Topic overview.

   b. Context of vol. 3 (following vols. 1 & 2).

   c. Aims: Vol. 3 in the series will continue the exploration of quantum metrology with photoelectrons, with a focus on numerical analysis techniques, forging a closer link between experimental and theoretical results, and making the methodologies discussed directly accessible via a new software platform/ecosystem.

2. Quantum metrology software platform/ecosystem overview

   a. Introduction to python packages for simulation, data analysis, and open-data.

   b. Photoelectron metrology toolkit (PEMtk) package/platform for experimental data processing & analysis. (See *pemtk.readthedocs.io*.)

   c. ePSproc package for theory & simulation. (See *epsproc.readthedocs.io*.)

   d. ePSdata platform for data/results library (see *ePSdata motivations*).

3. General method development: geometric tensor treatment of photoionization, fitting & matrix-inversion techniques

   a. Theory development overview - tensor methods (e.g. *ePSproc tensor methods*)

b. Direct molecular frame reconstruction via matrix-inversion methods (see Gregory, Margaret, Paul Hockett, Albert Stolow, and Varun Makhija. "Towards Molecular Frame Photoelectron Angular Distributions in Polyatomic Molecules from Lab Frame Coherent Rotational Wavepacket Evolution." *Journal of Physics B: Atomic, Molecular and Optical Physics* 54, no. 14 (July 2021): 145601.*DOI: 10.1088/1361-6455/ac135f*.)

4. Numerical implementation & analysis platform tools

   a. Tensor methods implementation in ePSproc/PEMtk.

   b. Information content analysis (inc. basis-set exploration, e.g. *PEMtk fitting demo*), see also vol. 2, sect. 12.1.

   c. Density matrix analysis. (e.g. *ePSproc density matrix method dev notes*)

   d. Generalised bootstrapping implementation in PEMtk (see vol. 2, sects. 11.3 & 12.3)

1.2.2 Part 2: numerical examples

Open-source worked examples using the new software platform.

1. Quantum metrology example: generalised bootstrapping for a homonuclear diatomic scattering system (N2)*

   a. Experimental data overview & simulation.

   b. Matrix element extraction (bootstrap protocol, see vol. 2, sects. 11.3 & 12.3) & statistical analysis.

   c. Direct molecular frame reconstruction via matrix-inversion methods.

   d. Comparison of methods.

   e. Information content/quantum information analysis. (See vol. 2, sect. 12.1.)

2. Quantum metrology example: generalised bootstrapping for a heteronuclear scattering system (CO)*

   a. Experimental data overview & simulation.

   b. Matrix element extraction (bootstrap protocol, see vol. 2, sects. 11.3 & 12.3) & statistical analysis.

   c. Direct molecular frame reconstruction via matrix-inversion methods.

   d. Comparison of methods.

   e. Information content/quantum information analysis. (See vol. 2, sect. 12.1.)

3. Quantum metrology example: generalised bootstrapping and matrix-inversion methods for a complex/general asymmetric top scattering system (C2H4 (ethylene))*

   a. Experimental data overview & simulation.

   b. Matrix element extraction (bootstrap protocol, see vol. 2, sects. 11.3 & 12.3) & statistical analysis.

   c. Direct molecular frame reconstruction via matrix-inversion methods.

   d. Comparison of methods.

   e. Information content/quantum information analysis.

4. Future directions & outlook

5. Summary & conclusions

* Exact choice of "simple" and "complex" systems may change, but should include a homonuclear diatomic and/or heteronuclear diatomic, and symmetric and asymmetric top polyatomic systems. May also include an atomic example.

Part II

Theory & software

CHAPTER

TWO

INTRODUCTION

The overall aim of Vol. 3 is to expand, explore, and illustrate, quantum metrology with photoelectrons: specifically, the application of new python-based tools to tackle problems in matrix element retrieval. The book itself is written as a set of Jupyter Notebooks, hence all the material herein is available directly to readers, and can be run locally to further explore the topic, or adapt the methodology to new problems

Whilst this volume aims to provide a self-contained text, and computational examples which may be used without extensive background knowledge, a brief introduction to the core physics and some recent extensions is presented herein. The unfamiliar reader is referred to Volume 1 of the series for a more detailed presentation, and gateway to the literature []. Following the topical introduction, the remainder of Part I introduces the main computational and software tools, recent theory developments, and concludes with a general overview for approaching matrix element retrieval numerically.

Part II details the application of these tools to a few specific cases, starting with a (relatively) simple homonuclear diatomic example, then escalating to a polyatomic asymetric top case.

2.1 Topical introduction

2.2 Context & aims for Vol. 3

As noted previously, Vol. 3 is somewhat distinct from the previous volumes in the series; although involving computational elements, Vols. 1 & 2 [] are more traditional publications. The material presented in this volume aims to continue the exploration of quantum metrology with photoelectrons, with a focus on numerical analysis techniques, forging a closer link between experimental and theoretical results, and making the methodologies discussed directly accessible via a new software platform/ecosystem. In order to fulfil this aim, Vol. 3 is a computational/computable document, with code directly available to readers. Each chapter or section is composed of a Jupyter Notebook (`.ipynb`), each of which can be modified and used independently.

To facilitate code transparency and reuse, the book is available via a Github repository, Quantum Metrology Vol. 3. An HTML version is also available, which includes interactive figures. A full introduction to the relevant tool-chain, including installation instructions, can be found in Chapter 3: *Quantum metrology software platform/ecosystem overview*.

CHAPTER

THREE

QUANTUM METROLOGY SOFTWARE PLATFORM/ECOSYSTEM OVERVIEW

STUB

In recent years, a unified Python codebase/ecosystem/platform has been in development to tackle various aspects of photoionization problems, including *ab initio* computations and experimental data handling, and (generalised) matrix element retrieval methods. The eponymous *Quantum Metrology with Photoelectrons* platform is introduced here, and is used for the analysis herein. The main aim of the platform is to provide a unifying data platform, and analysis routines, for photoelectron metrology, including new methods and tools, as well as a unifying bridge between these and existing tools. Fig. 3.1 provides a general overview of some of the main tools and tasks/layers.

As of late 2022, the new parts of the platform - primarily the Photoelectron Metrology Toolkit [] library - implement general data handling (although not a full experimental analysis toolchain), matrix element handling and retrieval, which will be the main topic of this volume. In the future, it is hoped that the platform will be extended to other theoretical and experimental methods, including full experimental data handling.

3.1 Analysis components

The two main components of the platform for analysis tasks, as used herein, are:

- The Photoelectron Metrology Toolkit [] (PEMtk) codebase aims to provide various general data handling routines for photoionization problems. At the time of writing, simulation of observables and fitting routines are implemented, along with some basic utility functions. Much of this is detailed herein, and more technical details and ongoing documentation case be found in the PEMtk documentation [].
- The ePSproc codebase [] aims to provide methods for post-processing with *ab initio* radial dipole matrix elements from ePolyScat (ePS) [], or equivalent matrix elements from other sources (dedicated support for R-matrix results from the RMT suite [] is in development). The core functionality includes the computation of AF and MF observables. Manual computation without known matrix elements is also possible, e.g. for investigating limiting cases, or data analysis and fitting - hence these routines also provide the backend functionality for PEMtk fitting routines. Again more technical details can be found in the ePSproc documentation [].

Other tools listed in Fig. 3.1 include:

- Quantum chemistry layer. The starting point for *ab initio* computations. For the examples herein, all computations made use of Gamess ("The General Atomic and Molecular Electronic Structure System") [] for electronic structure computations, and inputs to ePolyScat.
- ePolyScat (ePS) [] is an open-source tool for numerical computation of electron-molecule scattering & photoionization by Lucchese & coworkers. All matrix elements used herein were obtained via ePS calculations. For more details see ePolyScat website and manual [] and Refs. [].
- ePSdata is an open-data/open-science collection of ePS + ePSproc results [].

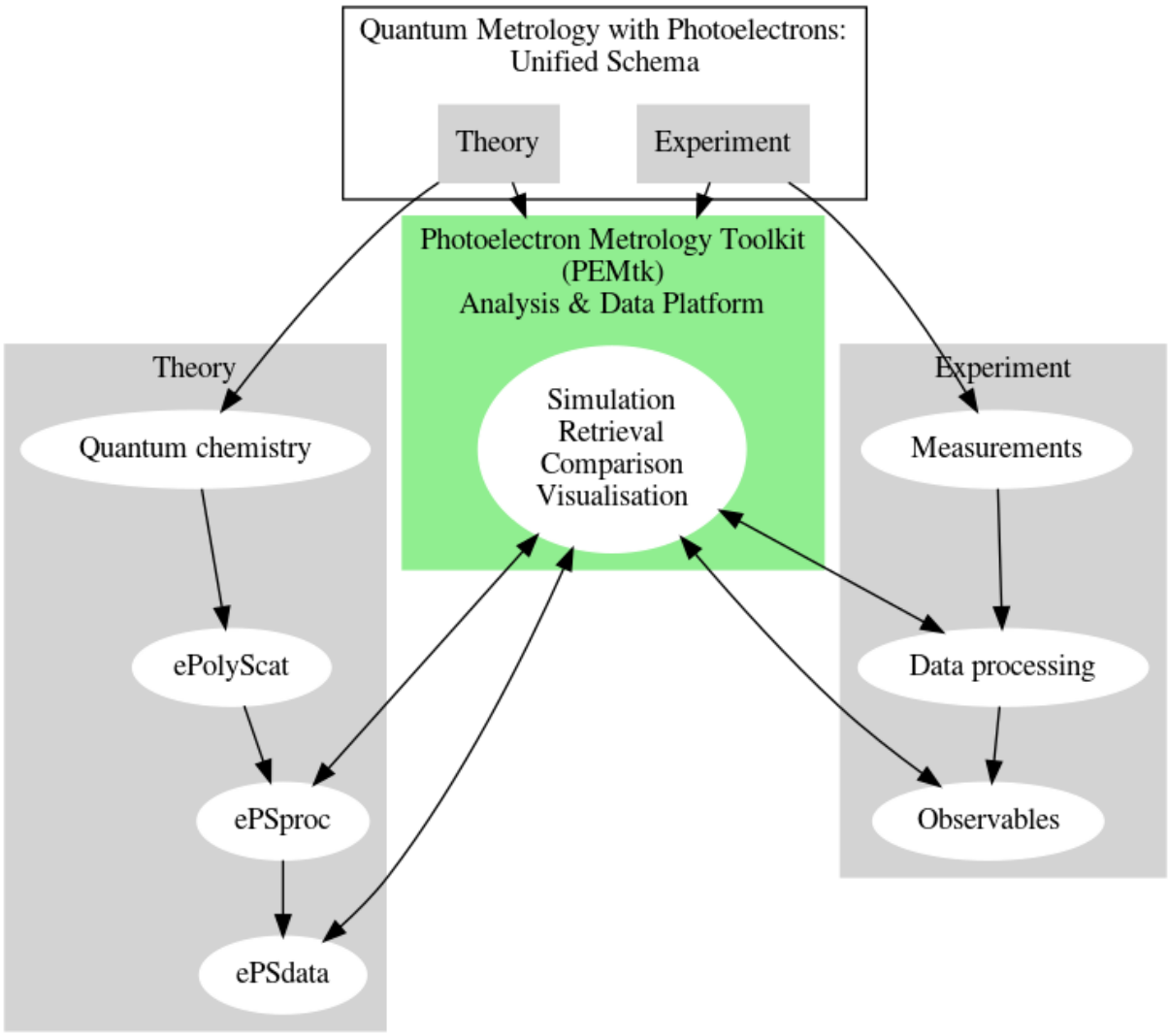


Fig. 3.1: Quantum metrology with photoelectrons ecosystem overview.

- ePSdata collects ePS datasets, post-processed via ePSproc (Python) in Jupyter notebooks, for a full open-data/open-science transparent pipeline.
- Source notebooks are available on the Github project pages, and notebooks + datasets via Zenodo repositories (one per dataset). Each notebook + dataset is given a Zenodo DOI for full traceability, and notebooks are versioned on Github.
- Note: ePSdata may also be linked or mirrored on the existing ePolyScat Collected Results OSF project, but will effectively supercede those pages.
- All results are released under <a href="https://creativecommons.org/licenses/by-nc-sa/4.0/">Creative Commons Attribution-NonCommercial-ShareAlike 4.0 (CC BY-NC-SA 4.0) license, and are part of our ongoing Open Science initiative.

A Docker-based distribution of various codes for tackling photoionization problems is also available from the Open Photoionization Docker Stacks [] project, which aims to make a range of these tools more accessible to interested researchers, and currently includes Docker builds for `ePS`, `ePSproc` and `PEMtk`.

Note that, at the time of writing, rotational wavepacket simulation is not yet implemented in the PEMtk suite, and these must be obtained via other codes. An intial build of the `limapack` suite for rotational wavepacket simulations is currently part of the Open Photoionization Docker Stacks [], but has yet to be tested.

```
For related tools and more Docker builds, see also the [Open Photoionization Docker␣
↪Stacks repo](https://github.com/phockett/open-photoionization-docker-stacks)
```

Part III

Test pages

CHAPTER

FOUR

FORMATTING TESTS

For directives and how-tos see:

- General Jupyter Book intro: https://jupyterbook.org/en/stable/content/myst.html
- https://jupyterbook.org/en/stable/content/index.html
- https://myst-parser.readthedocs.io/en/latest/faq/index.html (includes direct rst directive for native rst/Sphinx blocks).

Only outstanding point is use of raw latex? Is this possible…?

4.1 Links and refs

See https://jupyterbook.org/en/stable/content/references.html

- `{numref}` for numbered ref with custom text: Chapter 3 (always works in PDF, need to set numbered sections for HTML)
- `{ref}` for named ref: *Quantum metrology software platform/ecosystem overview*
- Combined for full number and name: Chapter 3: *Quantum metrology software platform/ecosystem overview*
- Nested as a link? Chapter 3: [](chpt:platformIntro)

4.1.1 Substitution tests

See https://jupyterbook.org/en/stable/content/content-blocks.html?highlight=substitutions#substitutions-and-variables-in-markdown

For global subs define in `_config.yml`. Uses Jinja on backend.

Basic: Something

Sub test with URL: Photoelectron Metrology Toolkit []

4.2 Fig testing (from file)

As a figure, see https://jupyterbook.org/en/stable/content/figures.html

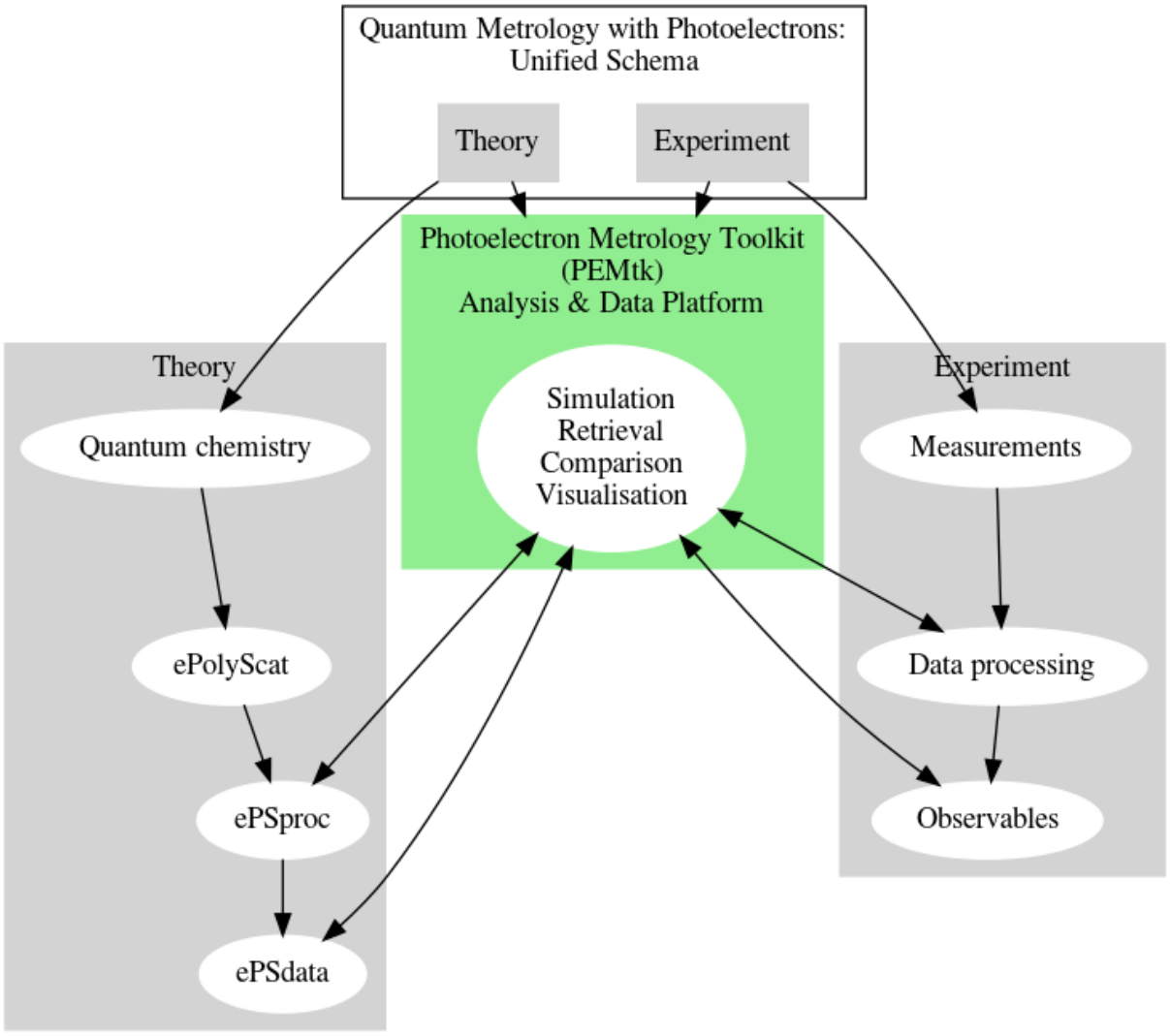
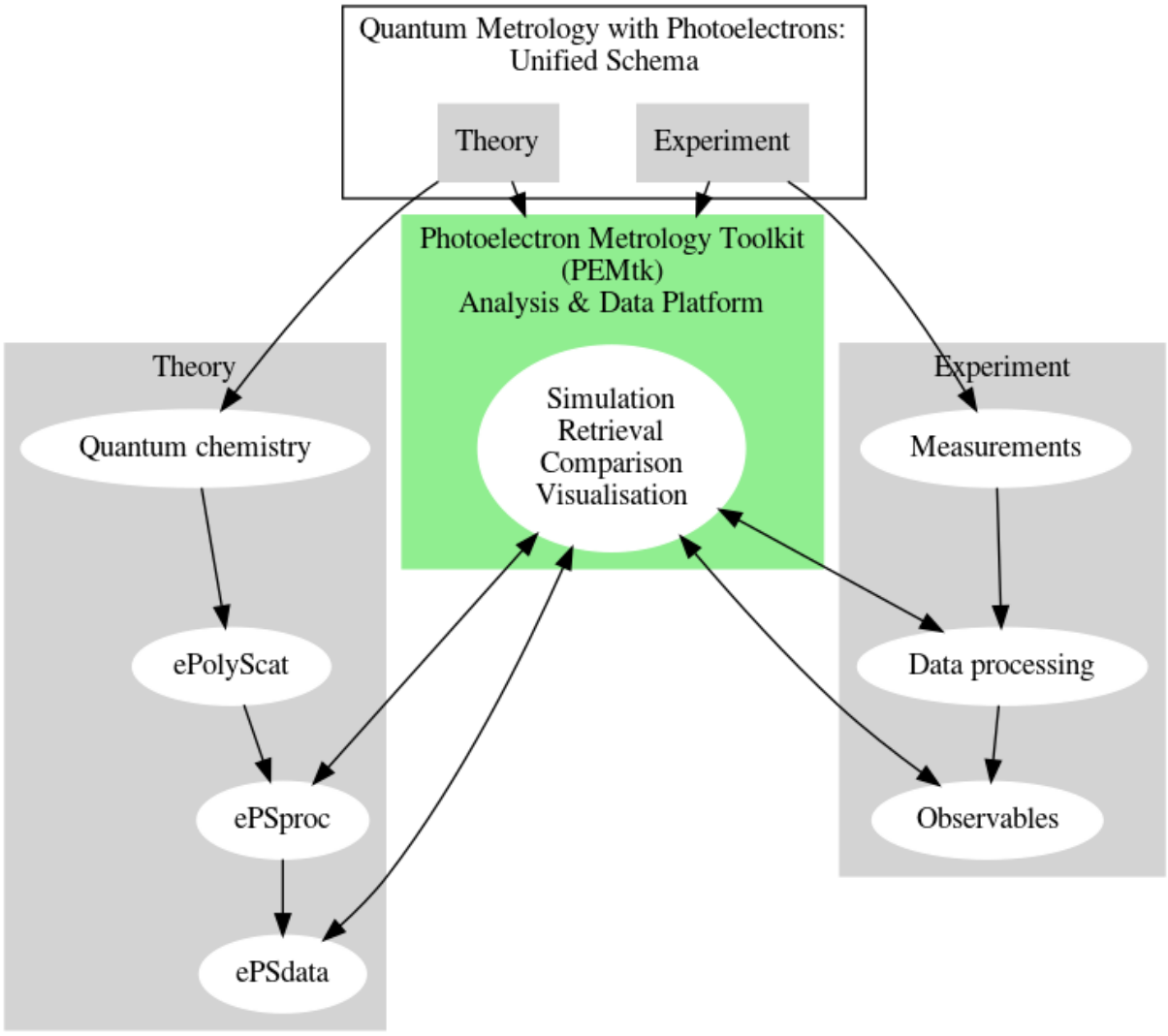


Fig. 4.1: Here is my figure caption!

4.3 Fig testing (from URL)

Seemed to work OK for both HTML and PDF output, although did get some build errors in latter case. For PDF, image copied locally to hashed-named file, so probably not ideal (and seems to break build…?).

Markdown image link



As a figure, see https://jupyterbook.org/en/stable/content/figures.html

4.4 Testing raw latex...

Both options fail in PDF and HTML export tests.

BETTER: JUST USE PANDOC FOR EXTENDED LATEX SECTIONS > Markdown. Will generally only need to fix minor issues (formatting and citations) in most cases anyway, if maths works.

For MyST markdown…?

- RST to MyST with Pandoc: https://github.com/executablebooks/rst2myst/issues/2 and https://github.com/executablebooks/rst-to-myst

Code cell with `%%latex`

```
%%latex
```

(continues on next page)

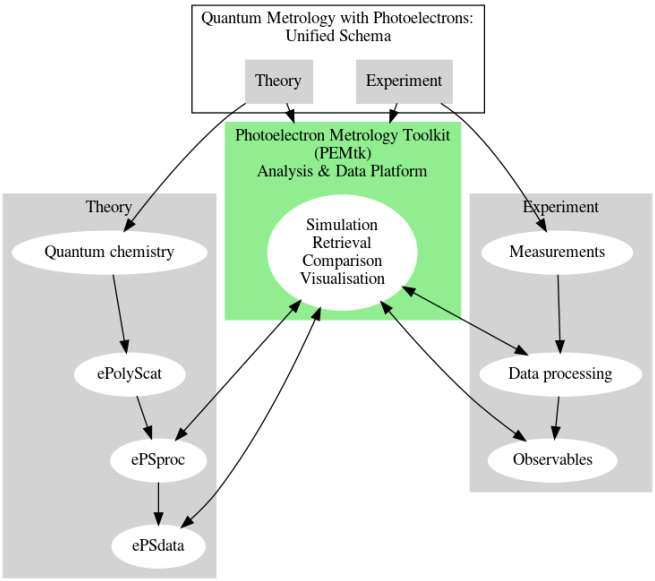


Fig. 4.2: Here is my figure caption!

(continued from previous page)

```
\bf{This is a test} \\
It allows latex $\alpha=\beta$ and test:

\begin{equation}
\alpha = \beta
\end{equation}
% this is a latex comment
```

Cell below marked as raw > latex in metadata.

With `{latex}` directive

With MyST-rst wrapper - this appears in PDF output, but not HTML. Also bold sticks for rest of output!

This is a test
It allows latex $\alpha = \beta$ and test:

$$\alpha = \beta \tag{4.1}$$

Testing maths only - OK in notebook, and all output forms too.

$$\alpha = \beta \tag{4.2}$$

4.5 Include

Working.

Include with `{include}` directive:

Some text to be included from another file.

CHAPTER

FIVE

PLOTLY SURFACE RENDER TEST

Quick tests for plotly compatibility.

For JupyterLab, need additional extensions - see https://plotly.com/python/getting-started/#jupyterlab-support:

- `conda install -c conda-forge -c plotly jupyter-dash`
- `jupyter labextension install jupyterlab-plotly`

In some cases may get partially working installation with, e.g., blank surface plots, or plots via HV only. This usually means JupyterLab needs a restart (and maybe a rebuild). For more see https://plotly.com/python/troubleshooting/

Also unstable behaviour in some Firefox builds? May be issue with add-ons? v106.0.3 on Win seems bad, but Linux versions seem OK?

Exports to HTML:

- **Manual from JupyterLab OK.**
- **Read the docs may be OK, but might depend on theme, see https://github.com/readthedocs/readthedocs.org/issues/4367. For default RTD theme, ensuring `sphinx_rtd_theme >= 1.0.0` in project `requirements.txt` should make things work. Alternatively, can manually set script load per page (see link above).**
- **Jupyter book (via Sphinx), fails to render unless `_config.yml` set as per https://jupyterbook.org/en/stable/interactive/interactive.html#plotly. But… seem to be missing figs in PDF output? (Although figs via ePSproc PAD plotting routines are appearing - may be missing some additional settings in this test notebook?)**

5.1 Plotly example

From https://plotly.com/python/3d-surface-plots/#passing-x-and-y-data-to-3d-surface-plot

On blank plots or `Javascript error "t is null".` try a browser refresh and/or Jupyterlab restart. Seeing this frequently in testing March 2022, Firefox 98.0.2, JupyterLab 3.3.2, see end of page for other versions - running via Holoviews wrapper seems to be more robust here?

```
import plotly.graph_objects as go
import pandas as pd
import numpy as np
# Read data from a csv
z_data = pd.read_csv('https://raw.githubusercontent.com/plotly/datasets/master/api_
↪docs/mt_bruno_elevation.csv')
z = z_data.values
```

(continues on next page)

(continued from previous page)

```
sh_0, sh_1 = z.shape
x, y = np.linspace(0, 1, sh_0), np.linspace(0, 1, sh_1)
fig = go.Figure(data=[go.Surface(z=z, x=x, y=y)])
fig.update_layout(title='Mt Bruno Elevation', autosize=False,
                  width=500, height=500,
                  margin=dict(l=65, r=50, b=65, t=90))
fig.show()
```

```
# From https://www.tutorialspoint.com/plotly/plotly_3d_scatter_and_surface_plot.htm
# Also empty
import numpy as np
x = np.outer(np.linspace(-2, 2, 30), np.ones(30))
y = x.copy().T # transpose
z = np.cos(x ** 2 + y ** 2)
trace = go.Surface(x = x, y = y, z =z )
data = [trace]
layout = go.Layout(title = '3D Surface plot')
fig = go.Figure(data = data)
# iplot(fig)
fig.show()
```

5.2 Holoviews 3D surface test

From: https://holoviews.org/gallery/demos/plotly/surface_3d.html

See also

- http://holoviews.org/reference/elements/matplotlib/Surface.html
- http://holoviews.org/reference/elements/plotly/Surface.html

This wrapper seems robust, and works even when Plotly native rendering is giving issues?

```
import numpy as np
import holoviews as hv
hv.extension('plotly')
```

5.2.1 Define data

```
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

surface = hv.Surface(Z, bounds=(-5, -5, 5, 5))
```

5.2.2 Plot

```
surface.opts(colorbar=True, width=500, height=500)
```

```
:Surface   [x,y]   (z)
```

5.2.3 Plotting with coords

BUT only supports regularly gridded coord systems (not x,y as grids). Trying to pass `(X,Y,Z)` in latter case gives `ValueError: Surface coordinates must be 1D arrays, x and y dimension(s) were found to have multiple dimensions. Either supply 1D arrays or use the QuadMesh element for curvilinear coordinates.`.

```
xs = np.arange(-4, 4, 0.25)
ys = np.arange(-4, 4, 0.25)
X, Y = np.meshgrid(xs, ys)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)

surface = hv.Surface((xs, ys, Z))

surface.opts(cmap='fire', height=500, width=500)
```

```
:Surface   [x,y]   (z)
```

5.3 Versions

```
import scooby
scooby.Report(additional=['xarray', 'jupyterlab','plotly','holoviews'])
```

```
--------------------------------------------------------------------------------
  Date: Tue Nov 08 13:16:43 2022 UTC

                OS : Linux
            CPU(s) : 64
           Machine : x86_64
      Architecture : 64bit
               RAM : 62.8 GiB
       Environment : Jupyter
       File system : btrfs

  Python 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:20:46)
  [GCC 9.4.0]

            xarray : 2022.3.0
        jupyterlab : 3.2.1
            plotly : 5.11.0
         holoviews : 1.15.2
             numpy : 1.20.3
```

(continues on next page)

(continued from previous page)

```
             scipy : 1.7.1
           IPython : 7.28.0
        matplotlib : 3.4.3
            scooby : 0.7.0
-------------------------------------------------------------------------------
```

March 2022 testing: Docker with jupyter/scipy-notebook base + Firefox 98.0.2.

CHAPTER

SIX

EPSPROC BASE AND MULTIJOB CLASS INTRO

16/10/20

As of Oct. 2020, v1.3.0-dev, basic data classes are now implemented, and are now the easiest/preferred method for using ePSproc (as opposed to calling core functions directly, as illustrated in the functions guide).

A brief intro and guide to use is given here.

Aims:

- **Provide unified data architecture for ePSproc, ePSdata and PEMtk.**
- **Wrap plotting and computational functions for ease of use.**
- **Handle multiple datasets inc. comparitive plots.**

6.1 Setup

```
# For module testing, include path to module here, otherwise use global installation
local = True

if local:
    import sys
    if sys.platform == "win32":
        modPath = r'D:\code\github\ePSproc'  # Win test machine
        winFlag = True
    else:
        modPath = r'/home/femtolab/github/ePSproc/'  # Linux test machine
        winFlag = False

    sys.path.append(modPath)

# Base
import epsproc as ep

# Class dev code
from epsproc.classes.multiJob import ePSmultiJob
from epsproc.classes.base import ePSbase
```

```
* pyevtk not found, VTK export not available.
```

6.2 ePSbase class

The ePSbase class wraps most of the core functionality, and will handle all ePolyScat output files in a single data directory. In general, we'll assume:

- **an ePS *job* constitutes a single ionization event/channel (ionizing orbital) for a given molecule, stored in one or more output files.**
- **the data dir contains one or more files, where each file will contain a set of symmetries and energies, with either**
  - **one file per ionizing event. In this case, each file will equate to one job, and one entry in the class datastructure.**
  - **a single ionizing event, where each file contains a different set of energies for the given event (*energy chunked* fileset). In this case, the files will be stacked, and the dir will equate to one job and one entry in the class datastructure.**

The class datastructure is (currently) a set of dictionaries, with entries per job as above, and various data for each job. In general the data is stored in Xarrays.

The `multiJob` class extends the base class with reading from multple directories.

6.2.1 Load data

Firstly, set the data path, instantiate a class object and load the data.

```
# Set for ePSproc test data, available from https://github.com/phockett/ePSproc/tree/
 ↪master/data
# Here this is assumed to be on the epsproc path
import os
dataPath = os.path.join(sys.path[-1], 'data', 'photoionization', 'n2_multiorb')
```

```
# Instantiate class object.
# Minimally this needs just the dataPath, if verbose = 1 is set then some useful␣
 ↪output will also be printed.
data = ePSbase(dataPath, verbose = 1)
```

```
# ScanFiles() - this will look for data files on the path provided, and read from␣
 ↪them.
data.scanFiles()
```

```
*** Job orb6 details
Key: orb6
Dir D:\code\github\ePSproc\data\photoionization\n2_multiorb, 1 files.
{   'batch': 'ePS n2, batch n2_1pu_0.1-50.1eV, orbital A2',
    'event': ' N2 A-state (1piu-1)',
    'orbE': -17.096913836366,
    'orbLabel': '1piu-1'}

*** Job orb5 details
Key: orb5
Dir D:\code\github\ePSproc\data\photoionization\n2_multiorb, 1 files.
{   'batch': 'ePS n2, batch n2_3sg_0.1-50.1eV, orbital A2',
    'event': ' N2 X-state (3sg-1)',
```

(continues on next page)

(continued from previous page)

```
    'orbE': -17.341816310545997,
    'orbLabel': '3sg-1'}
```

In this case, two files are read, and each file is a different ePS job - here the $3\sigma_g^{-1}$ and $1\pi_u^{-1}$ channels in N2. The keys for the job are also used as the job names.

6.2.2 Basic info & plots

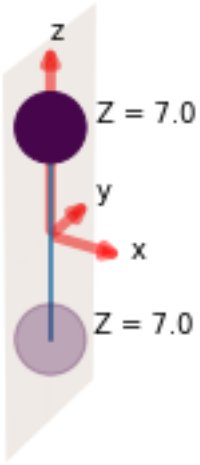
A few basic methods to summarise the data…

```
# Summarise jobs, this will also be output by scanFile() if verbose = 1 is set, as␣
↪illustrated above.
data.jobsSummary()
```

```
*** Job orb6 details
Key: orb6
Dir D:\code\github\ePSproc\data\photoionization\n2_multiorb, 1 files.
{   'batch': 'ePS n2, batch n2_1pu_0.1-50.1eV, orbital A2',
    'event': ' N2 A-state (1piu-1)',
    'orbE': -17.096913836366,
    'orbLabel': '1piu-1'}

*** Job orb5 details
Key: orb5
Dir D:\code\github\ePSproc\data\photoionization\n2_multiorb, 1 files.
{   'batch': 'ePS n2, batch n2_3sg_0.1-50.1eV, orbital A2',
    'event': ' N2 X-state (3sg-1)',
    'orbE': -17.341816310545997,
    'orbLabel': '3sg-1'}
```

```
# Molecular info
# Note that this is currently assumed to be the same for all jobs in the data dir.
data.molSummary()
```

```
*** Molecular structure
```



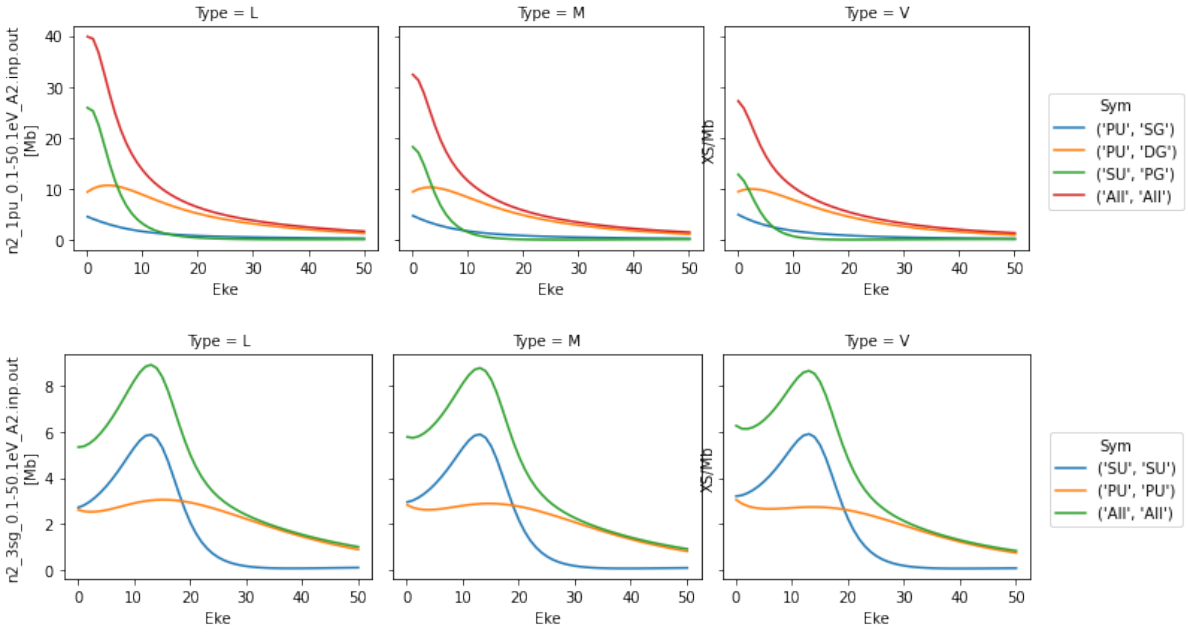

```
*** Molecular orbital list (from ePS output file)
EH = Energy (Hartrees), E = Energy (eV), NOrbGrp, OrbGrp, GrpDegen = degeneracies␣
↪and corresponding orbital numbering by group in ePS, NormInt = single centre␣
↪expansion convergence (should be ~1.0).
```

orb	props	Sym	EH	Occ	E	NOrbGrp	OrbGrp	GrpDegen	NormInt
1		SG	-15.6719	2.0	-426.454121	1.0	1.0	1.0	0.999532
2		SU	-15.6676	2.0	-426.337112	1.0	2.0	1.0	0.999458
3		SG	-1.4948	2.0	-40.675580	1.0	3.0	1.0	0.999979
4		SU	-0.7687	2.0	-20.917392	1.0	4.0	1.0	0.999979
5		SG	-0.6373	2.0	-17.341816	1.0	5.0	1.0	1.000000
6		PU	-0.6283	2.0	-17.096914	1.0	6.0	2.0	1.000000
7		PU	-0.6283	2.0	-17.096914	2.0	6.0	2.0	1.000000

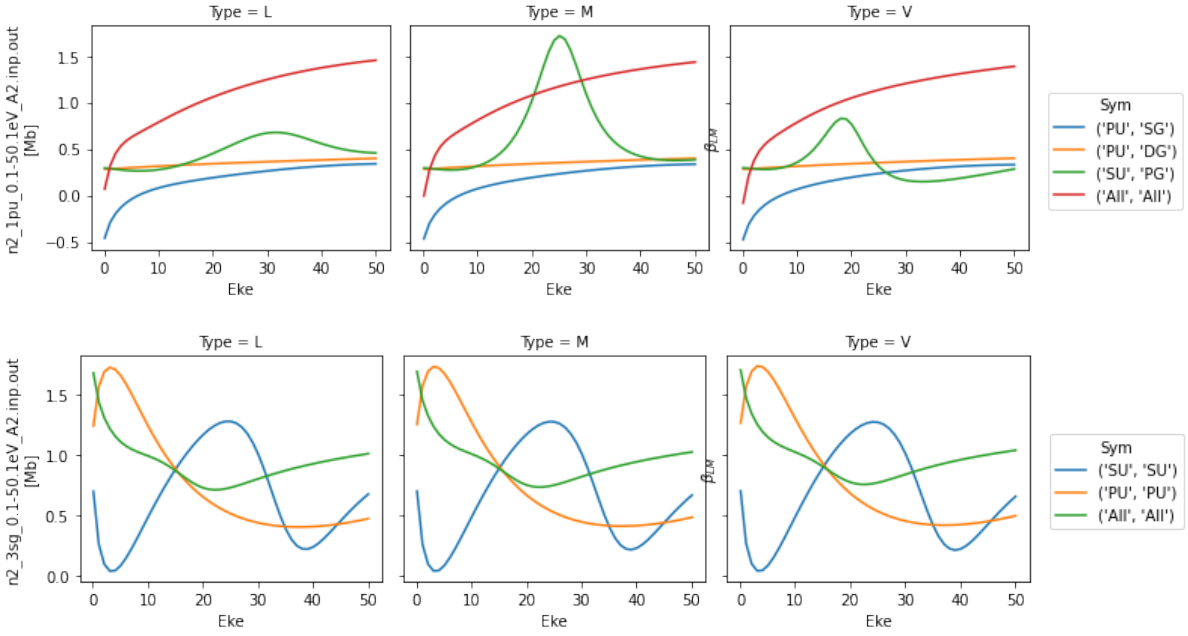
6.2.3 Plot cross-sections and betas

These are taken from the `GetCro` segments in the ePS output files, and correspond to results for an isotropic ensemble of molecules, i.e. observables in the lab frame (LF) for 1-photon ionization (see the ePS tutorial for more details).

```
# Minimal method call will plot cross-sections for all ePS jobs found in the data␣
↪directory.
data.plotGetCro()
```




```
# Plot beta parameters with the 'BETA' flag
data.plotGetCro(pType = 'BETA')
```



TODO: fix labelling here.

6.2.4 Compute MFPADs

The class currently wraps just the basic numerical routine for MFPADs. This defaults to computing MFPADs for all energies and (z, x, y) polarization geometries (where the z-axis is the molecular symmetry axis, and corresponds to the molecular structure plot shown above).

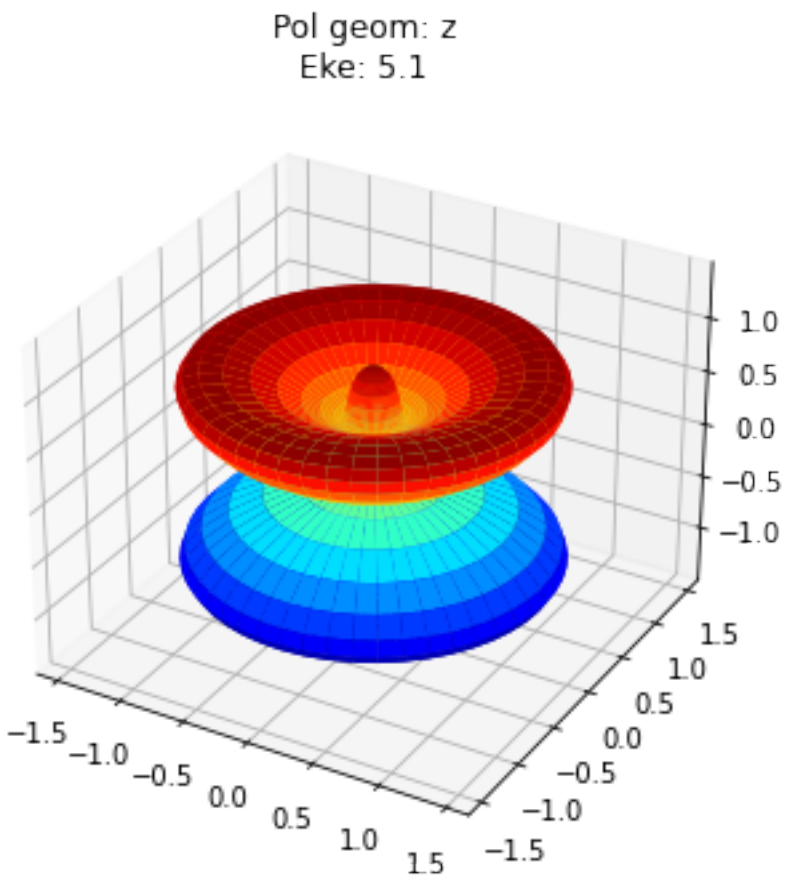
```
# Compute MFPADs...
data.mfpadNumeric()
```

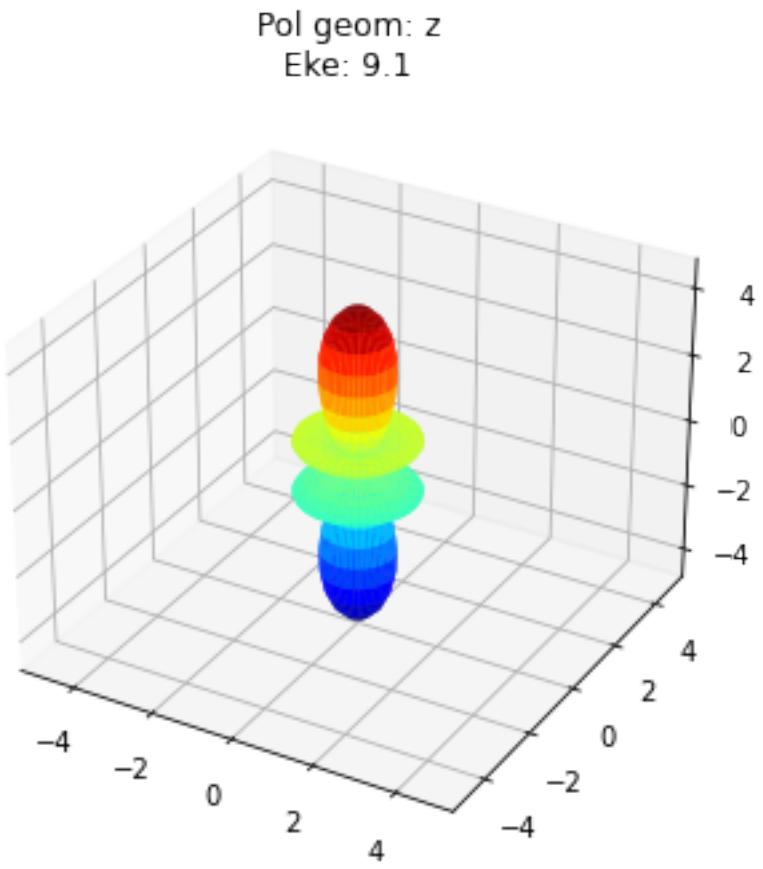
To plot it's advisable to set an enery slice, `Erange = [start, stop, step]`, since MFPADs are currently shown as individual plots in the default case, and there may be a lot of them.

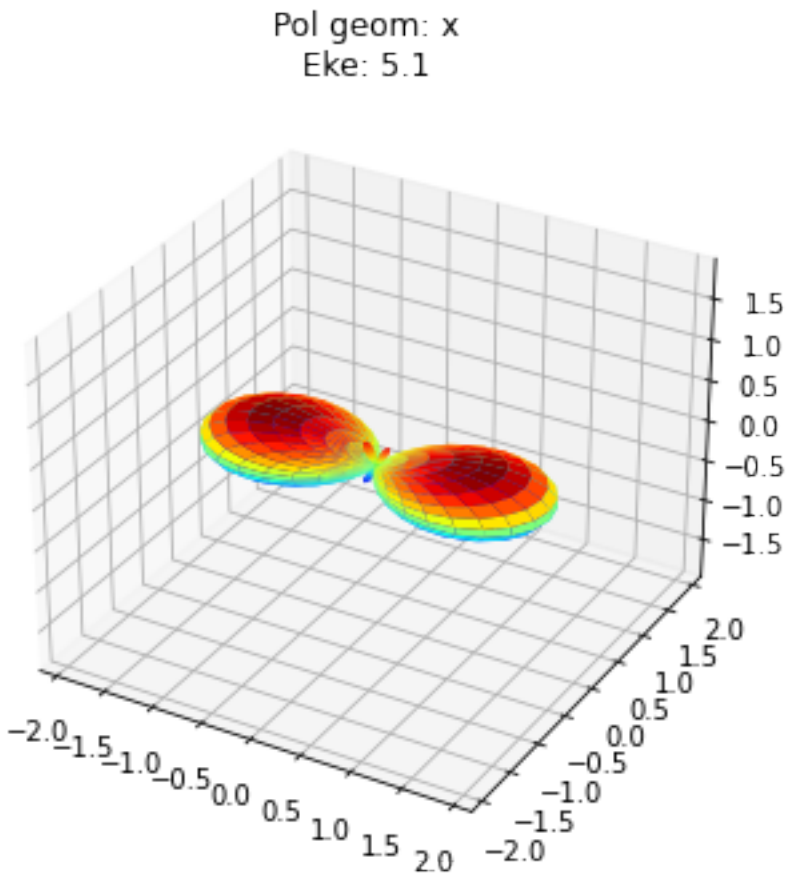
We'll also just set for a single key here, otherwise all jobs will be plotted.

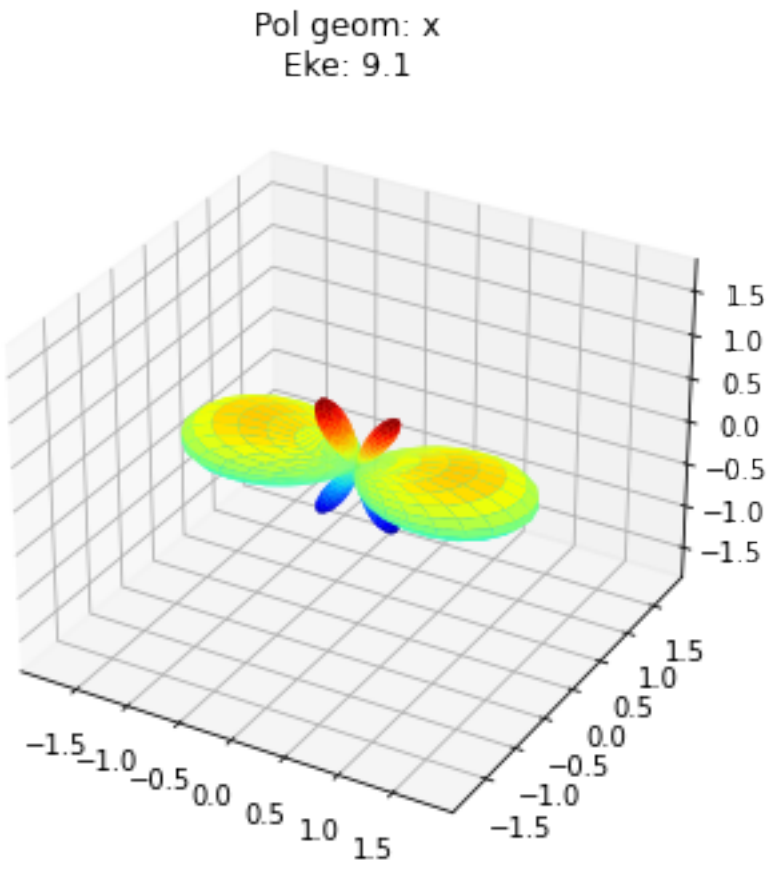
```
data.padPlot(keys = 'orb5', Erange = [5, 10, 4])

#TODO: fix plot layout!
```

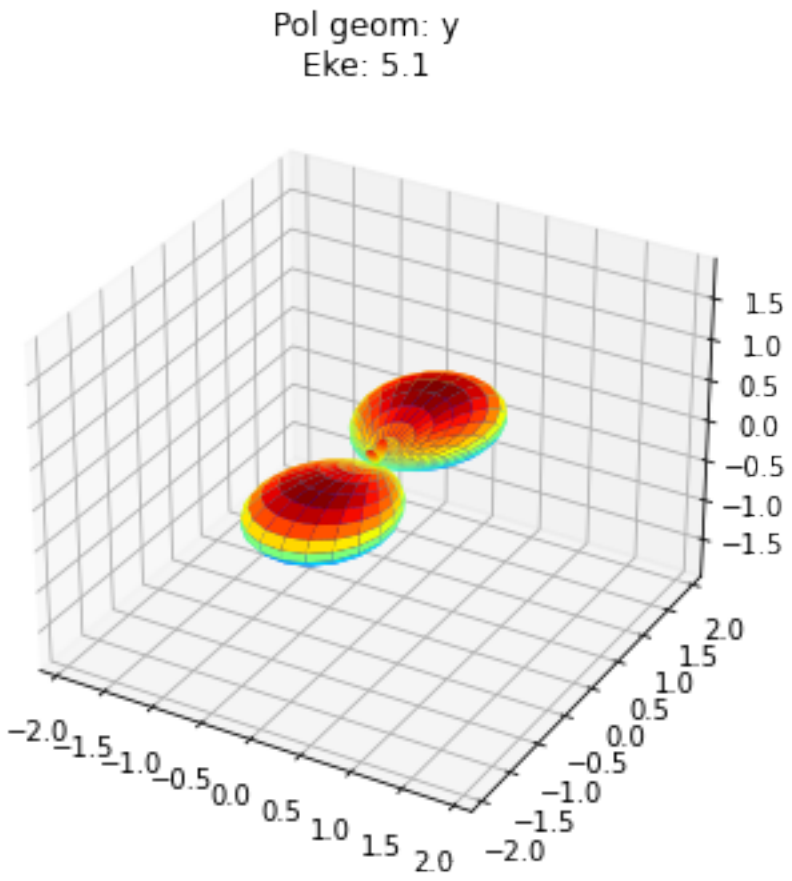
```
Found dims ('Labels', 'Phi', 'Theta', 'Eke', 'Sym'), summing to reduce for plot.␣
↪Pass selDims to avoid.
Sph plots: Pol geom: z
Plotting with mpl
Data dims: ('Phi', 'Theta', 'Eke'), subplots on Eke
Sph plots: Pol geom: x
Plotting with mpl
Data dims: ('Phi', 'Theta', 'Eke'), subplots on Eke
Sph plots: Pol geom: y
Plotting with mpl
Data dims: ('Phi', 'Theta', 'Eke'), subplots on Eke
```

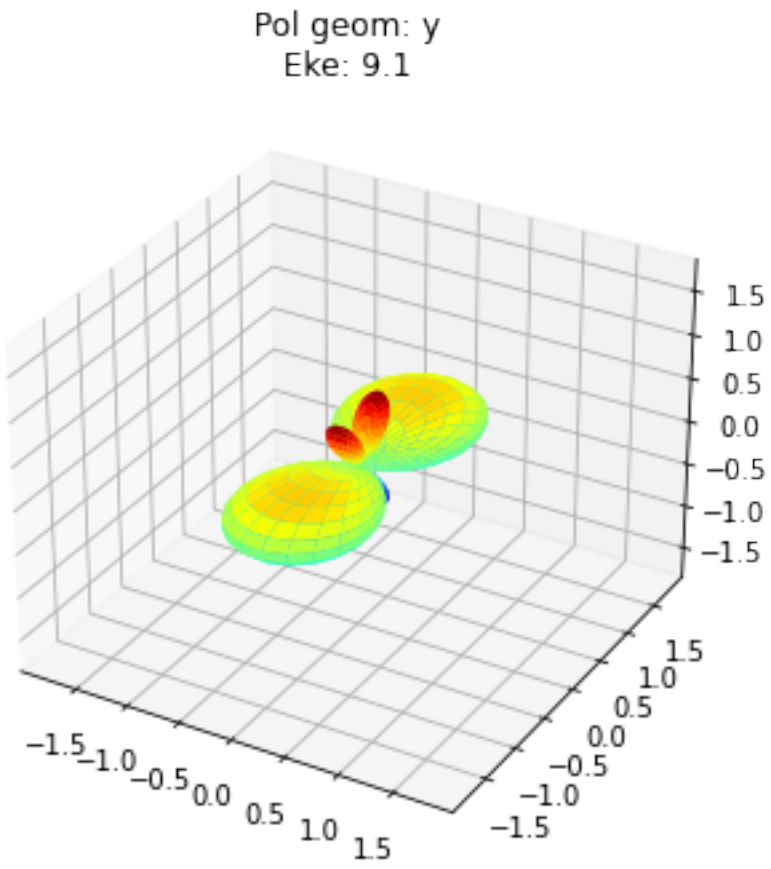

Pol geom: z
Eke: 5.1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5


Pol geom: z
Eke: 9.1
4
2
0
−2
−4
−4
−2
0
2
4
4
2
0
−2
−4


Pol geom: x
Eke: 5.1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0


Pol geom: x
Eke: 9.1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

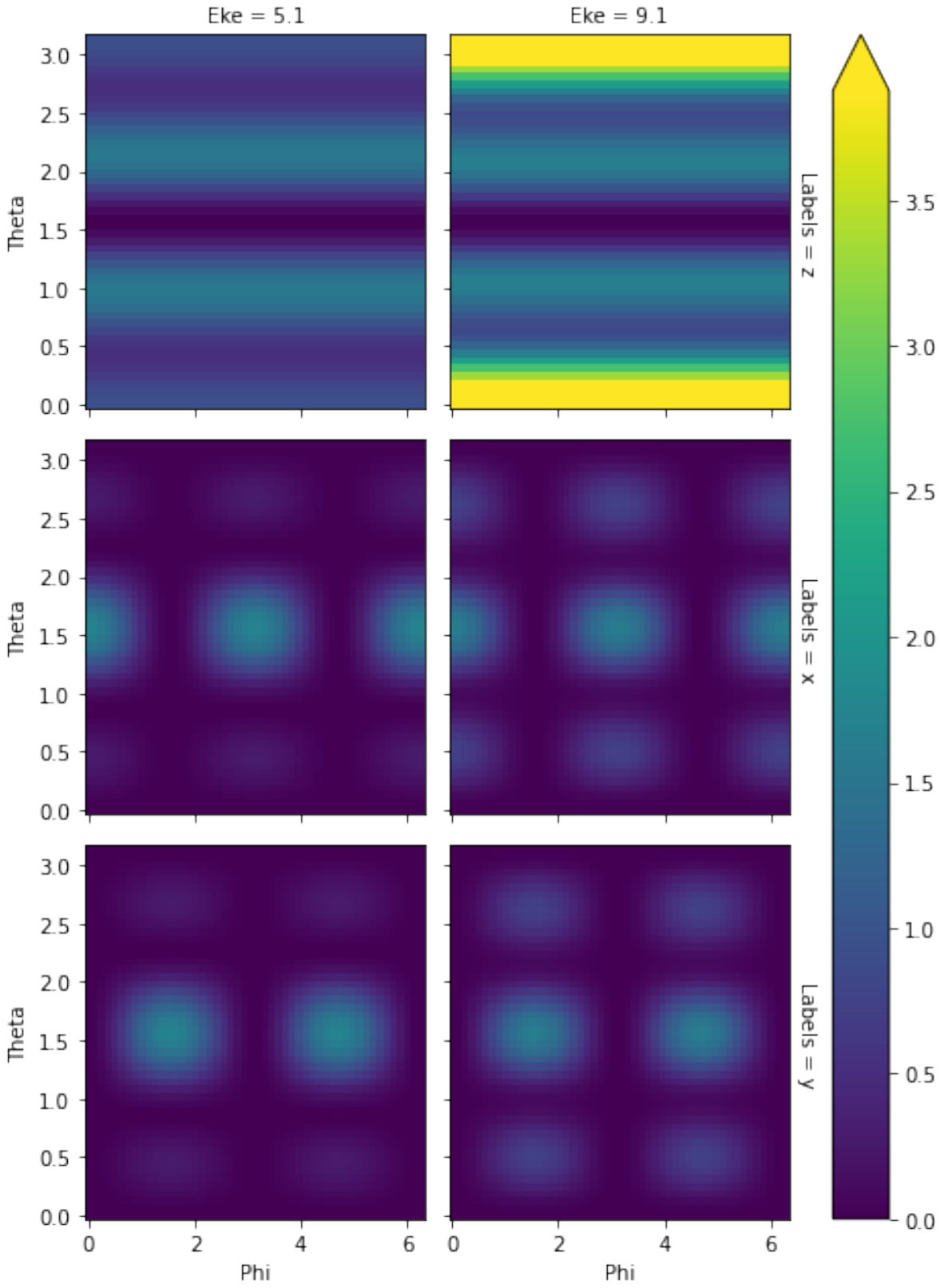

Pol geom: y
Eke: 5.1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0



To view multiple results in a more concise fashion, a Cartesian gridded output is also available.

```
data.padPlot(keys = 'orb5', Erange = [5, 10, 4], pStyle='grid')
```

```
Found dims ('Labels', 'Phi', 'Theta', 'Eke', 'Sym'), summing to reduce for plot.␣
↪Pass selDims to avoid.
Grid plot: 3sg-1
```


Eke = 5.1
Eke = 9.1
Theta
Phi
Labels = z
Labels = x
Labels = y
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
2
4
6
3.5

```
# Various other args can be passed...

# Set a plotting backend, currently 'mpl' (Matplotlib - default) or 'pl' (Plotly -␣
↪interactive, but may give issues in some environments)
backend = 'pl'

# Subselect on dimensions, this is set as a dictionary for Xarray selection (see␣
↪http://xarray.pydata.org/en/stable/indexing.html#indexing-with-dimension-names)
selDims = {'Labels':'z'}  # Plot z-pol case only.

data.padPlot(Erange = [5, 10, 4], selDims=selDims, backend = backend)
```

```
Found dims ('Phi', 'Theta', 'Eke', 'Sym'), summing to reduce for plot. Pass␣
↪selDims to avoid.
Sph plots: 1piu-1
Plotting with pl
```


```
Found dims ('Phi', 'Theta', 'Eke', 'Sym'), summing to reduce for plot. Pass␣
↪selDims to avoid.
Sph plots: 3sg-1
Plotting with pl
```

6.2.5 Compute β_{LM} parameters

For computation of β_{LM} parameters the class wraps functions from `epsproc.geom`, which implement a tensor method. This is quite fast, although memory heavy, so may not be suitable for very large problems. (See the method development pages for more info, more concise notes to follow).

- **Functions are provided for MF and AF problems (which is the general case, and will equate to the LF case for an unaligned ensemble).**
- **For the MF the class wraps `ep.geom.mfblmXprod`, see the method development page for more info, more concise notes to follow.**
- **For the AF the class wraps `ep.geom.afblmXprod`, see the method development page for more info, more concise notes to follow.**

Compute MF β_{LM} and PADs

Here's a quick demo for the default MF cases, which will give parameters corresponding to the (z, x, y) polarization geometries computed by the numerical routine above.

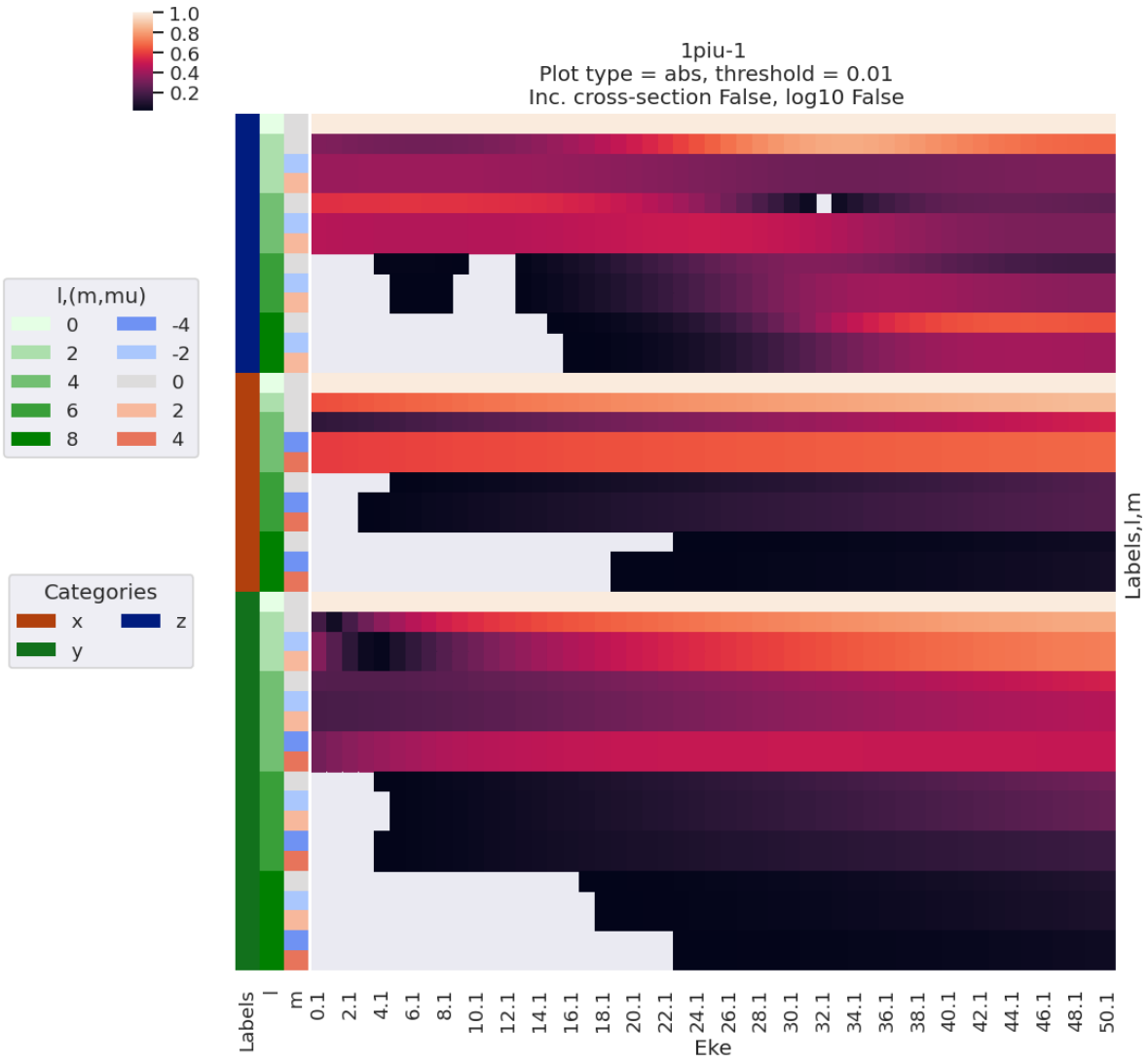
```
data.MFBLM()
```

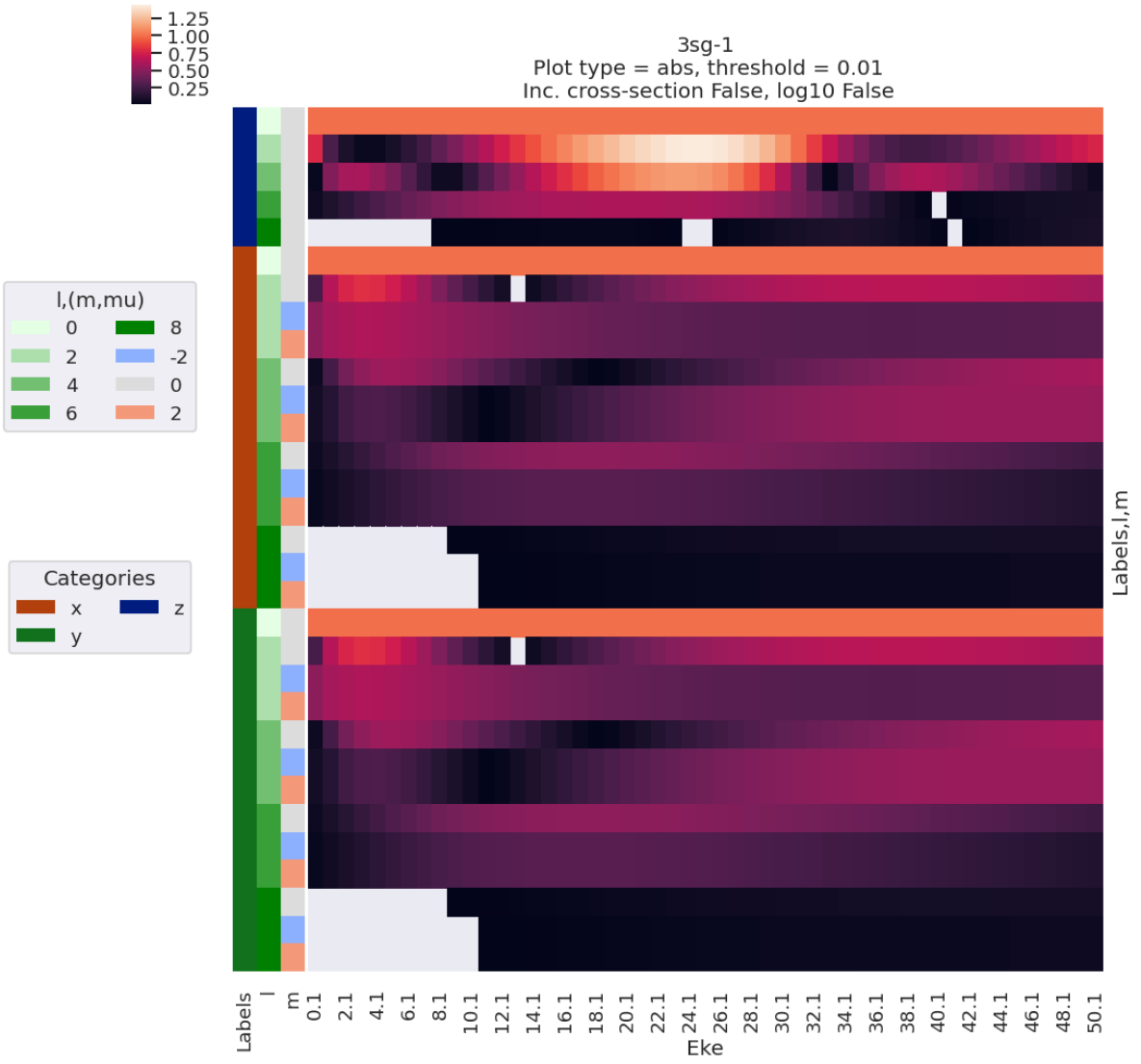
```
Calculating MF-BLMs for job key: orb6
Return type BLM.

Calculating MF-BLMs for job key: orb5
Return type BLM.
```

Plotting still needs to improve… but ep.lmPlot() is a robust way to plot everything.

```
# data.BLMplot(dataType = 'MFBLM')  # HORRIBLE OUTPUT at the moment!!!
data.lmPlot(dataType = 'MFBLM')
```

```
Plotting data n2_1pu_0.1-50.1eV_A2.inp.out, pType=a, thres=0.01, with Seaborn
Plotting data n2_3sg_0.1-50.1eV_A2.inp.out, pType=a, thres=0.01, with Seaborn
```

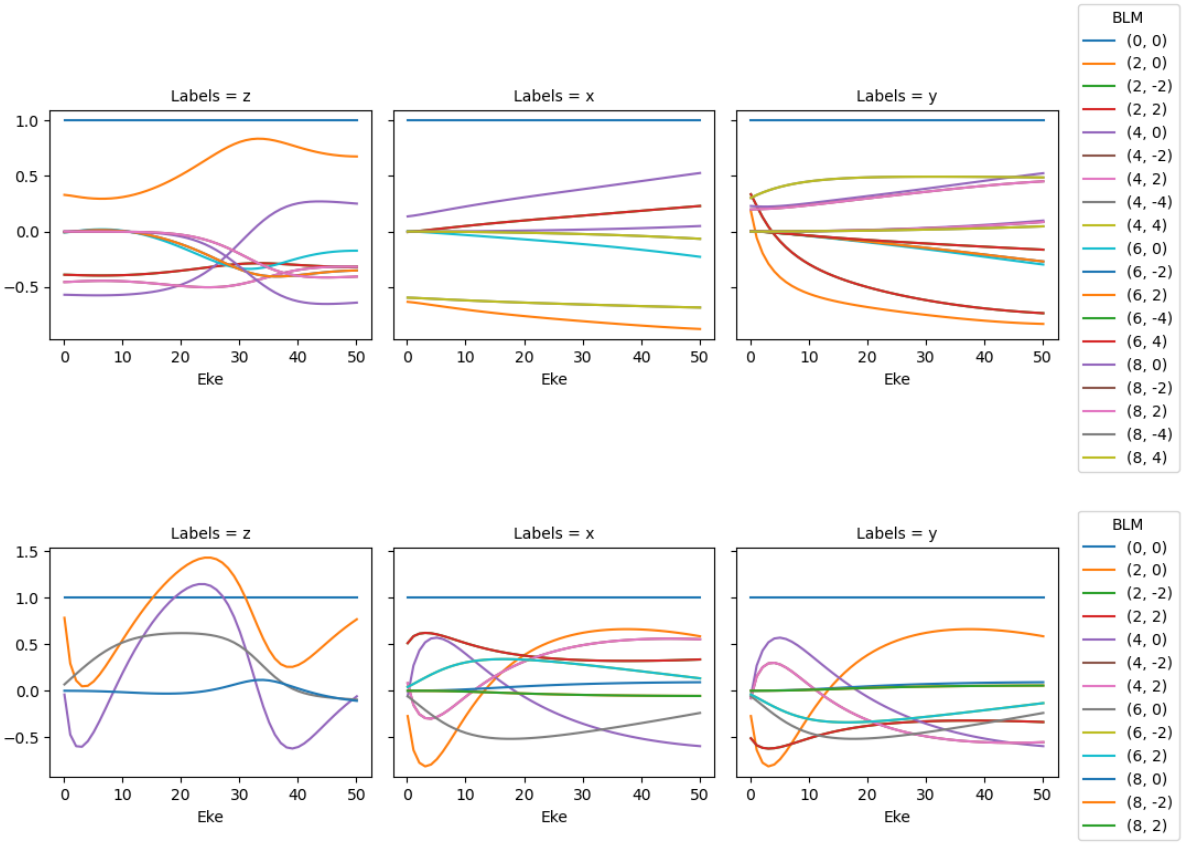

1.0
0.8
0.6
0.4
0.2
1piu-1
Plot type = abs, threshold = 0.01
Inc. cross-section False, log10 False
l,(m,mu)
0
2
4
6
8
-4
-2
0
2
4
Categories
x
y
z
Labels,l,m
Labels
l
m
0.1
2.1
4.1
6.1
8.1
10.1
12.1
14.1
16.1
18.1
20.1
22.1
24.1
26.1
28.1
30.1
32.1
34.1
36.1
38.1
40.1
42.1
44.1
46.1
48.1
50.1
Eke



Line-plots are available with the `BLMplot` method, although this currently only supports the Matplotlib backend, and may have problems with dims in some cases (work in progress!).

```
data.BLMplot(dataType='MFBLM', thres = 1e-2)  # Passing a threshold value here will␣
↪remove any spurious BLM parameters.
```

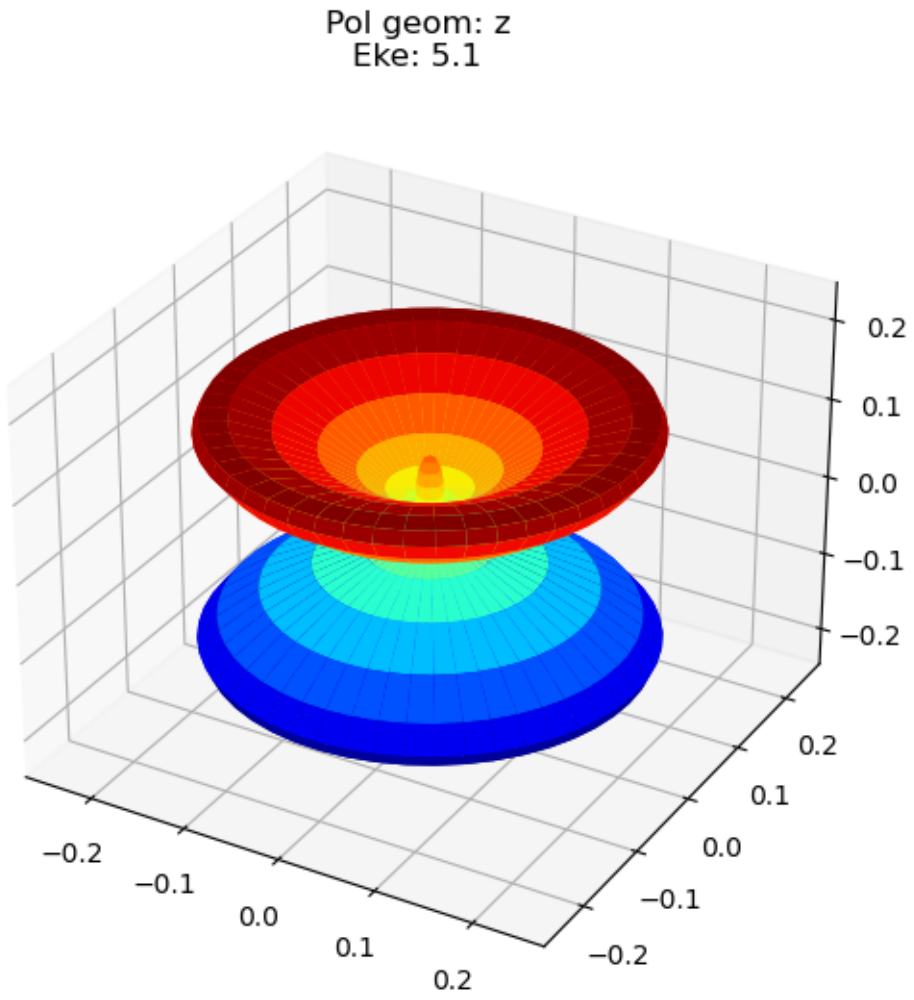
```
Dataset: orb6, 1piu-1, MFBLM
Dataset: orb5, 3sg-1, MFBLM
```

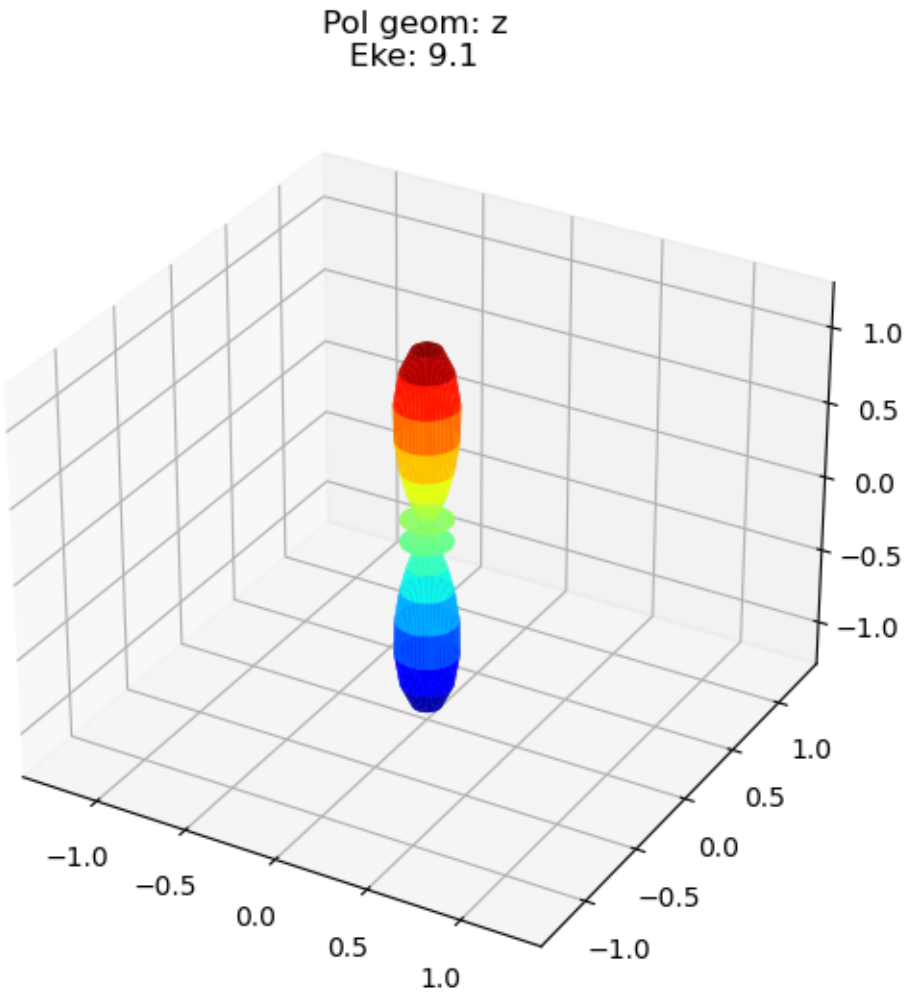


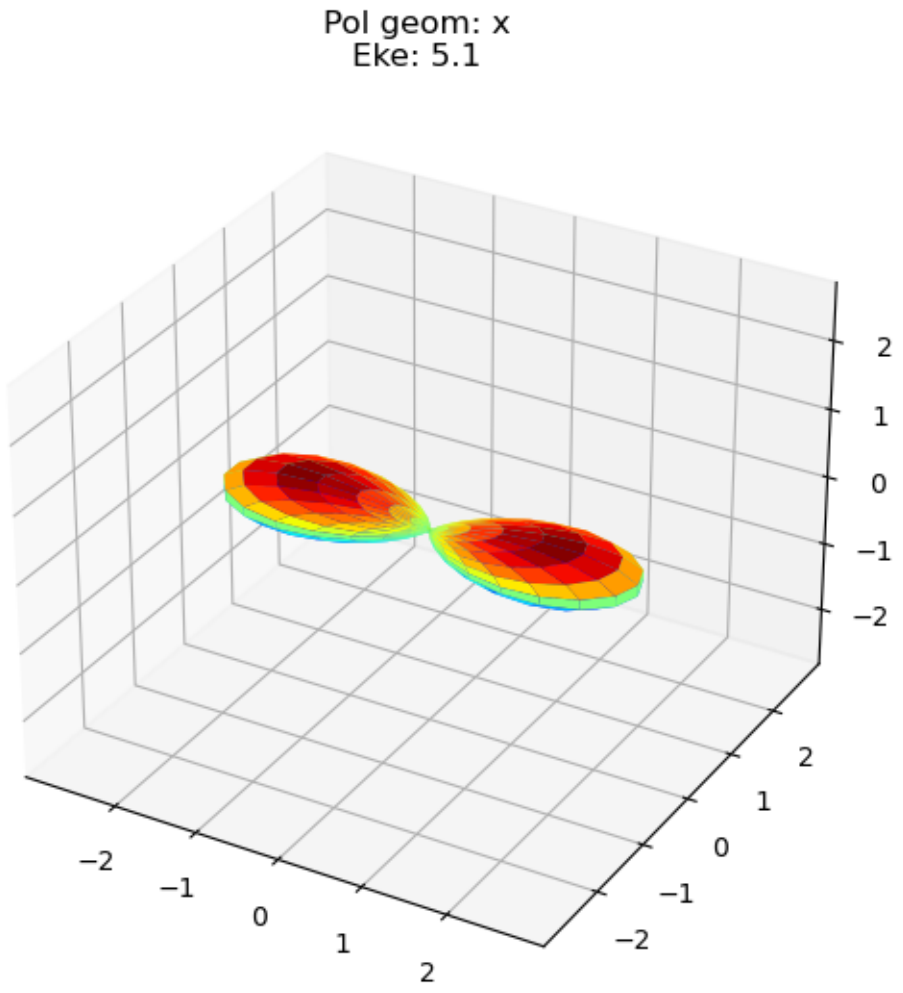
Polar plots are available for these distributions using the `padPlot()` method if the `dataType` is passed.

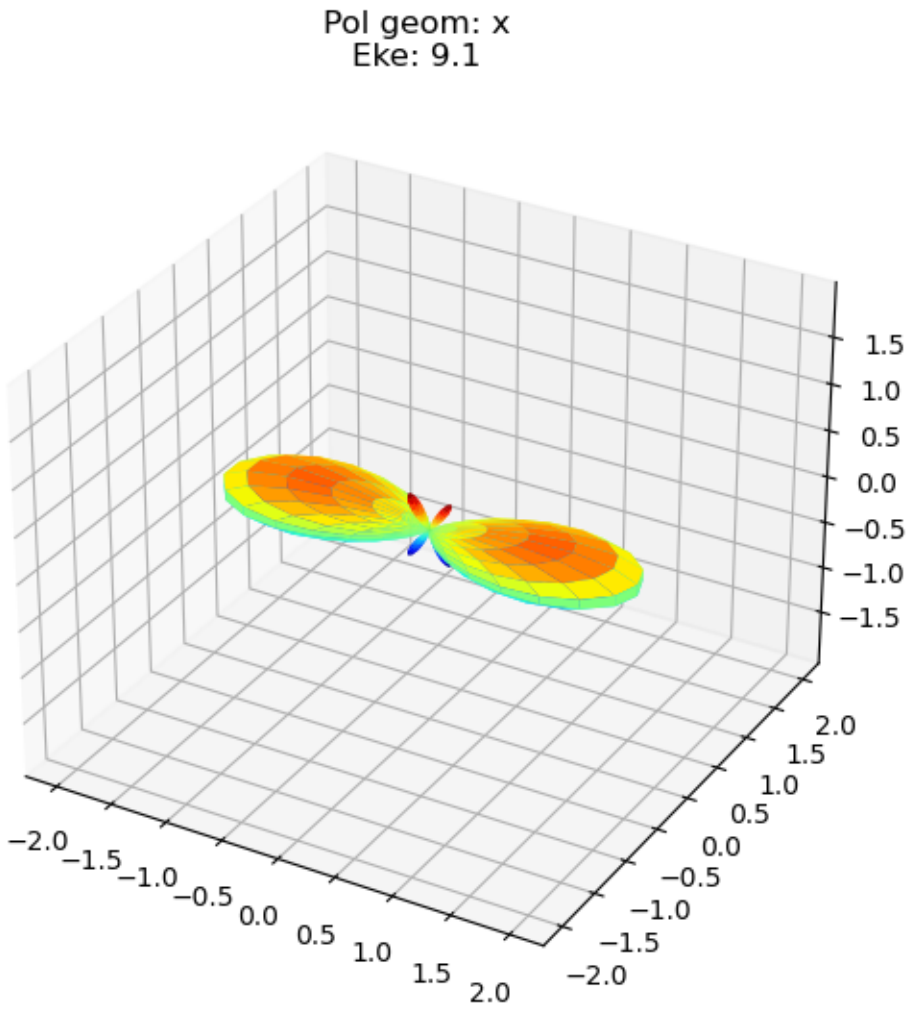
```
data.padPlot(keys = 'orb5', Erange = [5, 10, 4], dataType='MFBLM')
```

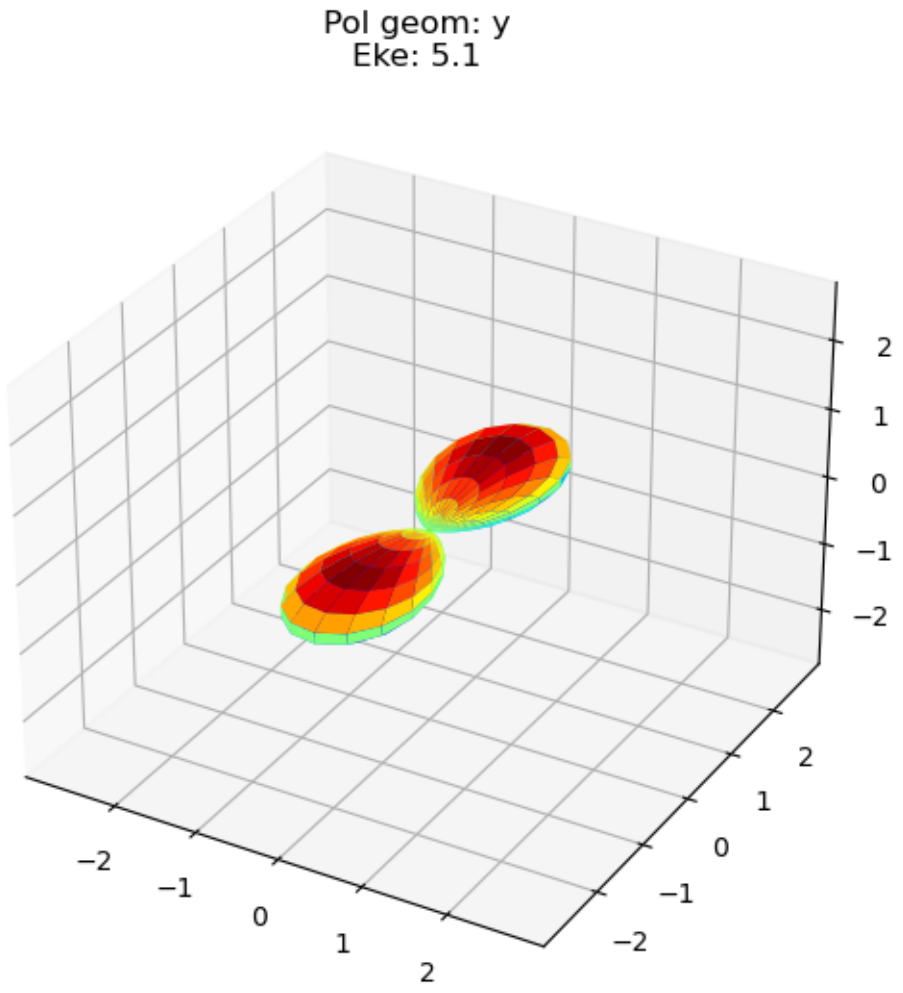
```
Using default sph betas.
Sph plots: Pol geom: z
Plotting with mpl
Data dims: ('Eke', 'Phi', 'Theta'), subplots on Eke
Sph plots: Pol geom: x
Plotting with mpl
Data dims: ('Eke', 'Phi', 'Theta'), subplots on Eke
Sph plots: Pol geom: y
Plotting with mpl
Data dims: ('Eke', 'Phi', 'Theta'), subplots on Eke
```

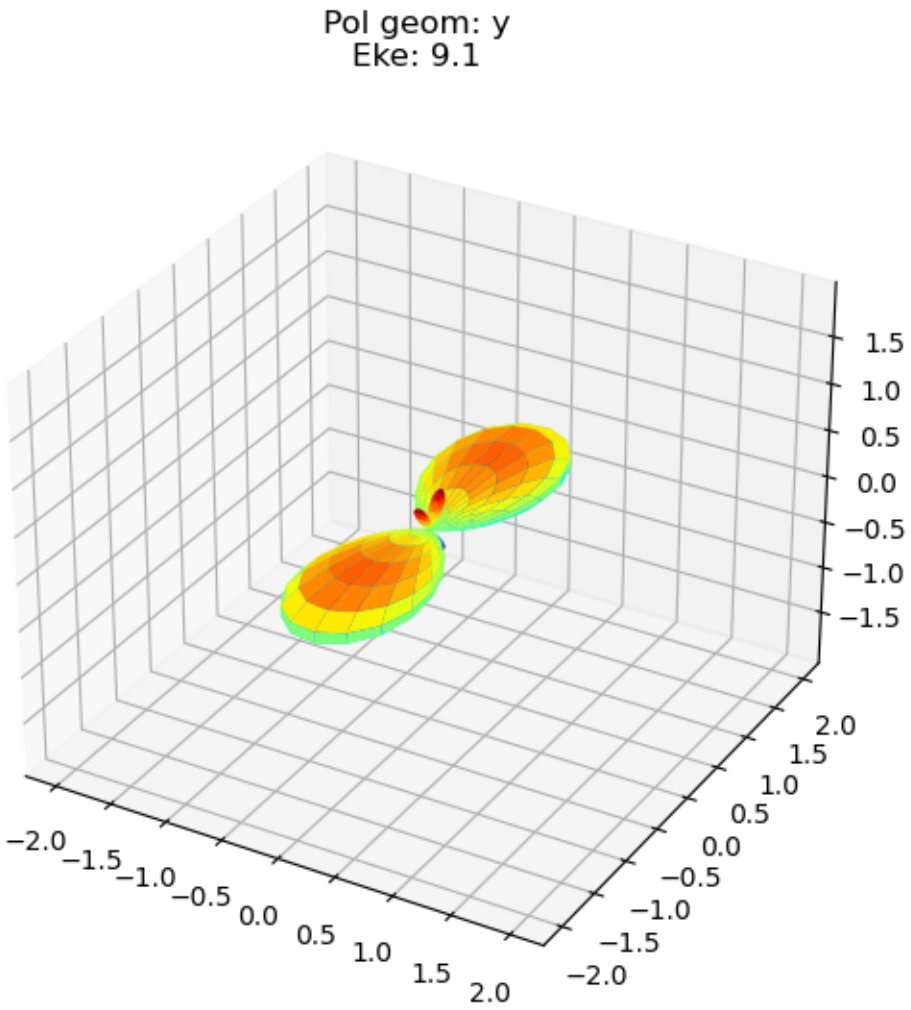

Pol geom: z
Eke: 5.1
0.2
0.1
0.0
−0.1
−0.2
−0.2
−0.1
0.0
0.1
0.2
0.2
0.1
0.0
−0.1
−0.2


Pol geom: z
Eke: 9.1
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.0
0.5
0.0
−0.5
−1.0


Pol geom: x
Eke: 5.1
2
1
0
−1
−2
−2
−1
0
1
2
2
1
0
−1
−2


Pol geom: x
Eke: 9.1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0


Pol geom: y
Eke: 5.1
2
1
0
−1
−2
−2
−1
0
1
2
2
1
0
−1
−2



Compute LF/AF β_{LM} and PADs

Here's a quick demo for the default AF case (isotropic distribution, hence == LF case).

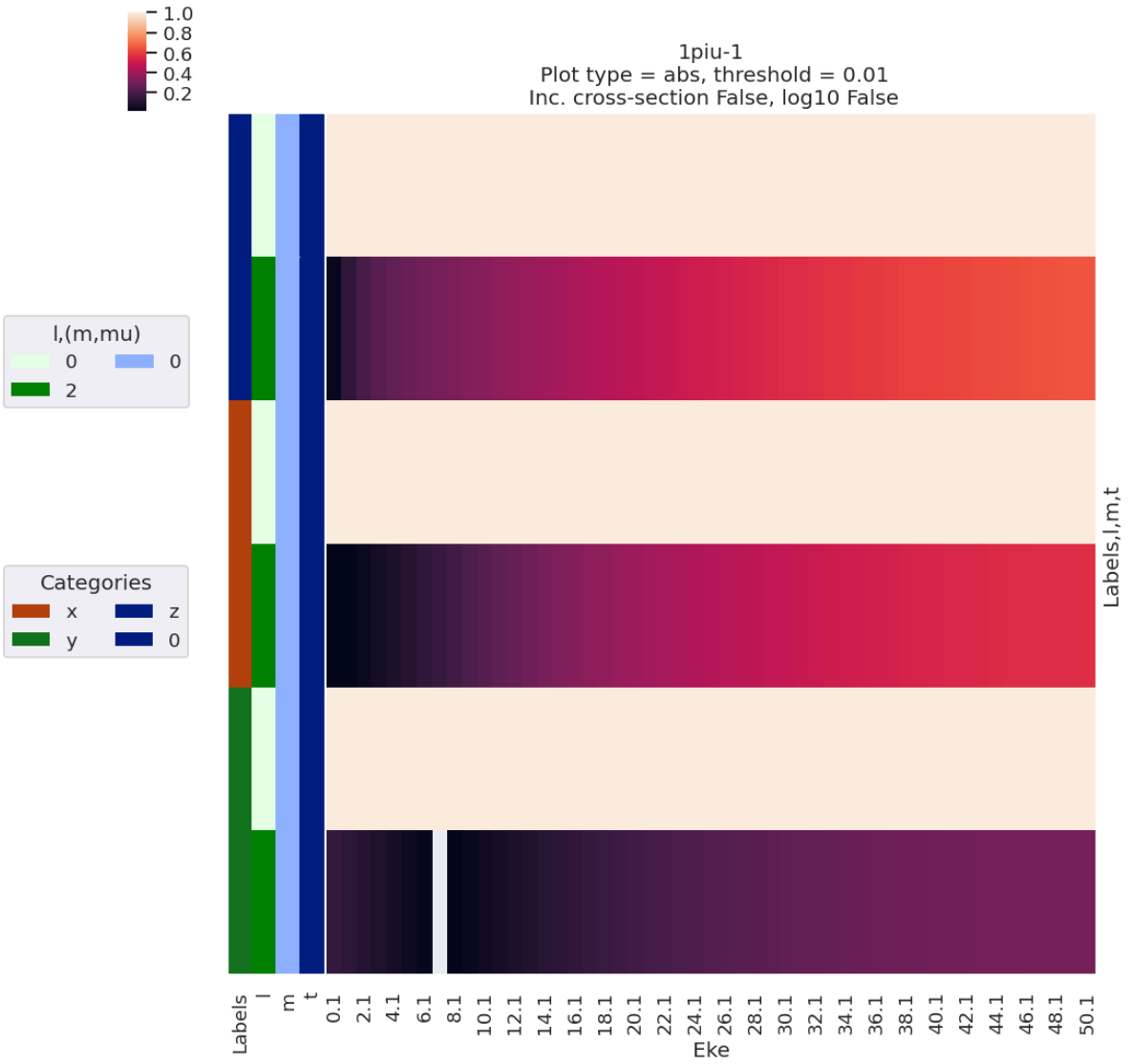
```
data.AFBLM()
```

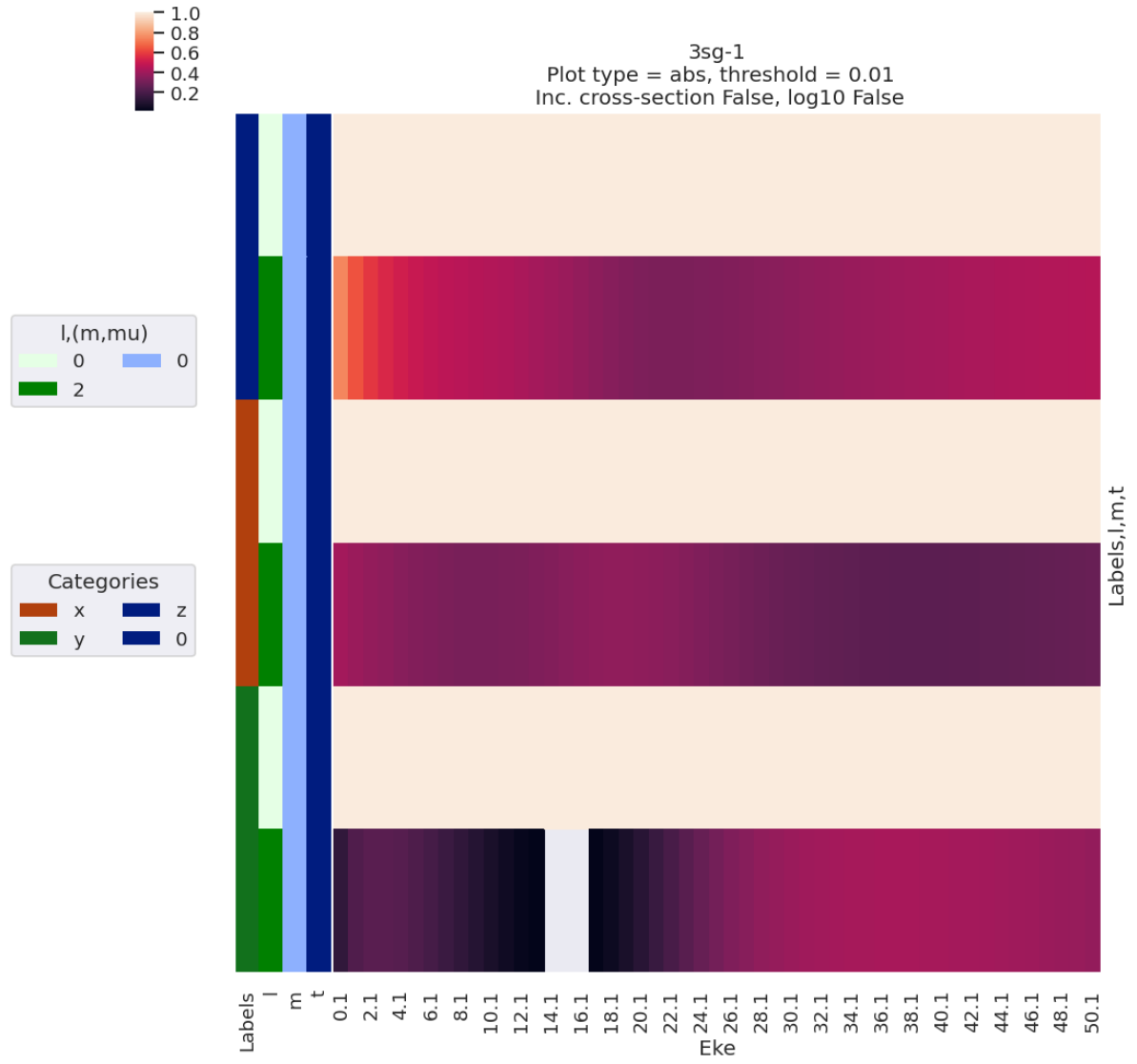
```
Calculating AF-BLMs for job key: orb6

Calculating AF-BLMs for job key: orb5
```

Plotting still needs to improve… but ep.lmPlot() is a robust way to plot everything.

```
# data.BLMplot(dataType = 'MFBLM')  # HORRIBLE OUTPUT at the moment!!!
data.lmPlot(dataType = 'AFBLM')
```

```
Plotting data n2_1pu_0.1-50.1eV_A2.inp.out, pType=a, thres=0.01, with Seaborn
Plotting data n2_3sg_0.1-50.1eV_A2.inp.out, pType=a, thres=0.01, with Seaborn
```

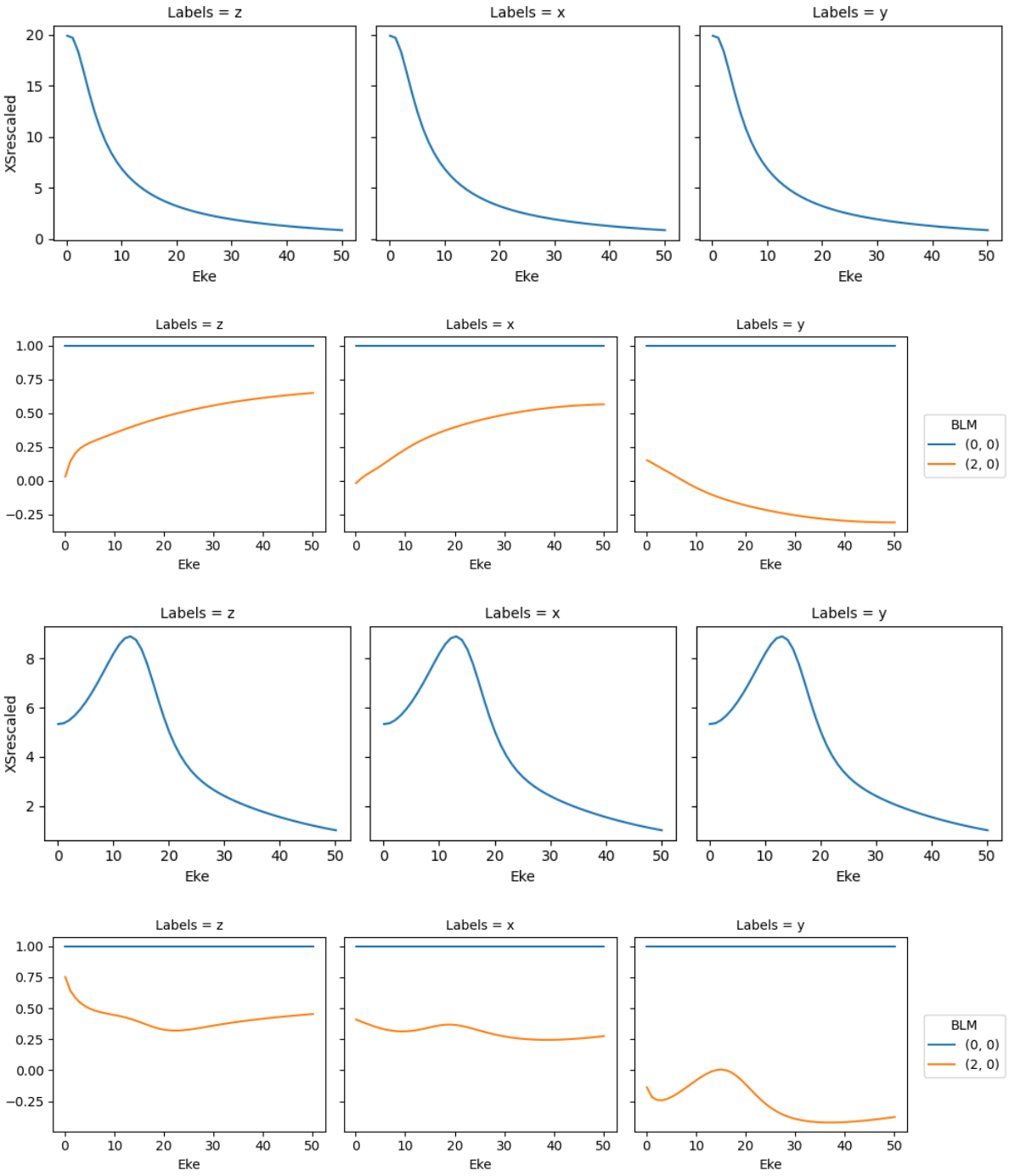

1.0
0.8
0.6
0.4
0.2
1piu-1
Plot type = abs, threshold = 0.01
Inc. cross-section False, log10 False
l,(m,mu)
0
0
2
Categories
x
z
y
0
Labels,l,m,t
Labels
l
m
t
0.1
2.1
4.1
6.1
8.1
10.1
12.1
14.1
16.1
18.1
20.1
22.1
24.1
26.1
28.1
30.1
32.1
34.1
36.1
38.1
40.1
42.1
44.1
46.1
48.1
50.1
Eke



Line-plots are available with the `BLMplot` method, although this currently only supports the Matplotlib backend, and may have problems with dims in some cases (work in progress!).

```
data.BLMplot(dataType='AFBLM', thres = 1e-2)  # Passing a threshold value here will␣
↪remove any spurious BLM parameters.
```

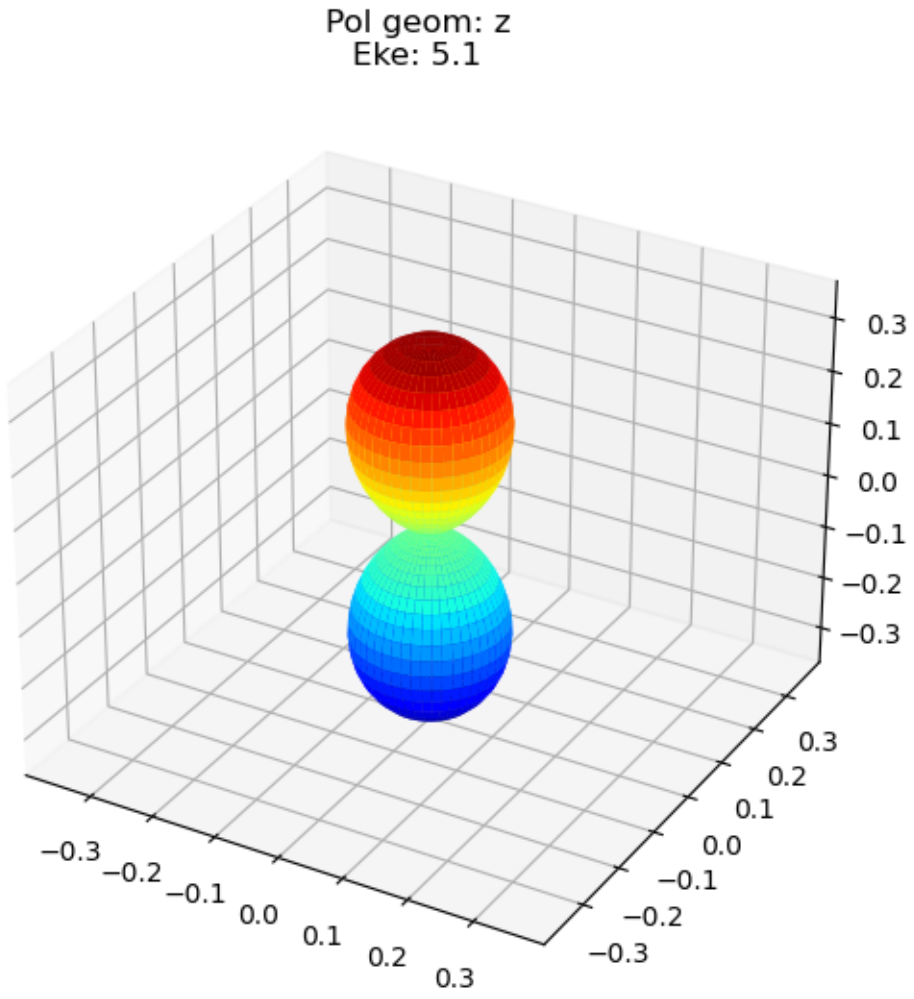
```
Dataset: orb6, 1piu-1, XS
Dataset: orb6, 1piu-1, AFBLM
Dataset: orb5, 3sg-1, XS
Dataset: orb5, 3sg-1, AFBLM
```

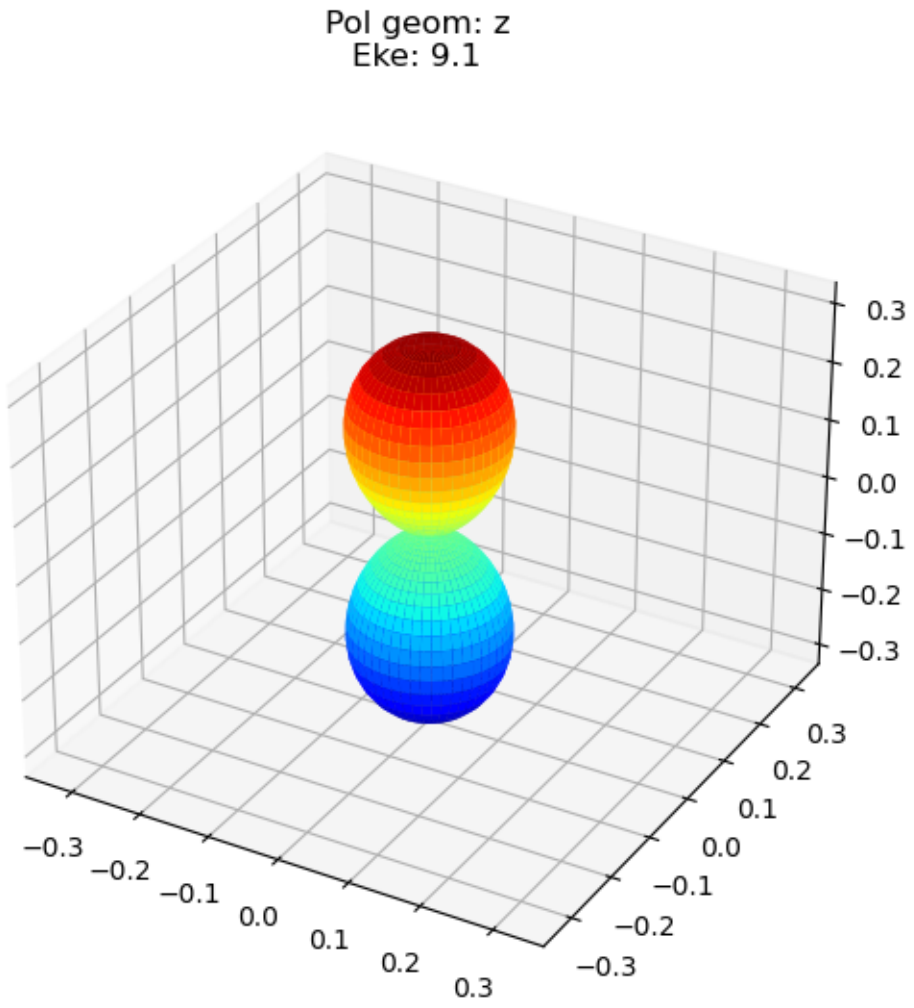


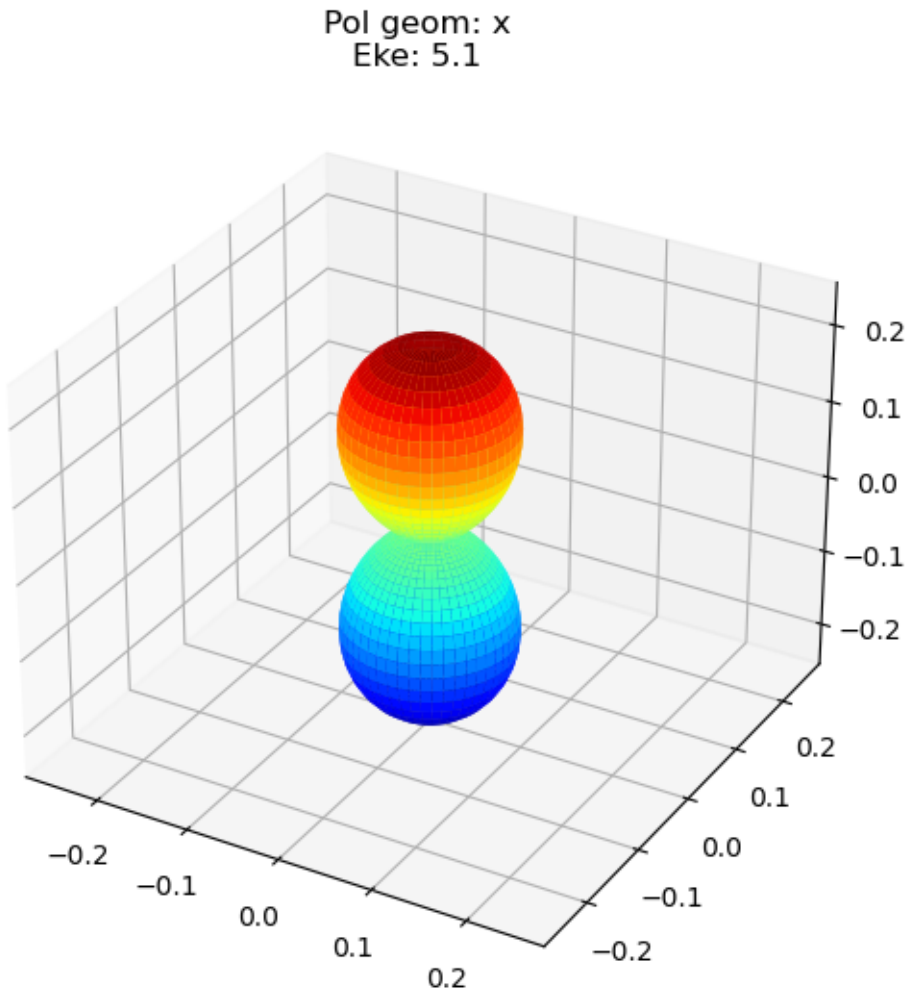
Polar plots are available for these distributions using the `padPlot()` method.

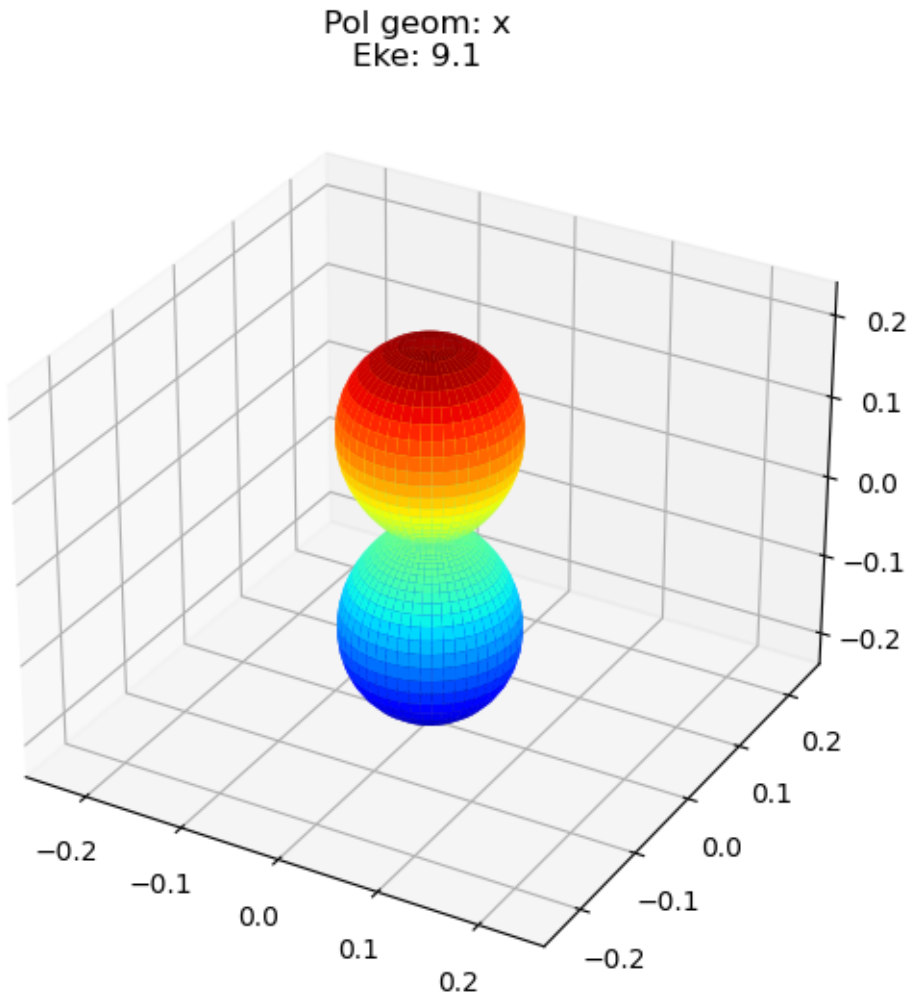
```
data.padPlot(keys = 'orb5', Erange = [5, 10, 4], dataType='AFBLM')

# NOTE - seem to have an inconsistency with (x,y) pol geometries here - should check␣
↪source code & fix. Likely due to mix-up in frame defns., i.e. probably mixing LF␣
↪and MF pol geom defn. - TBC.
```

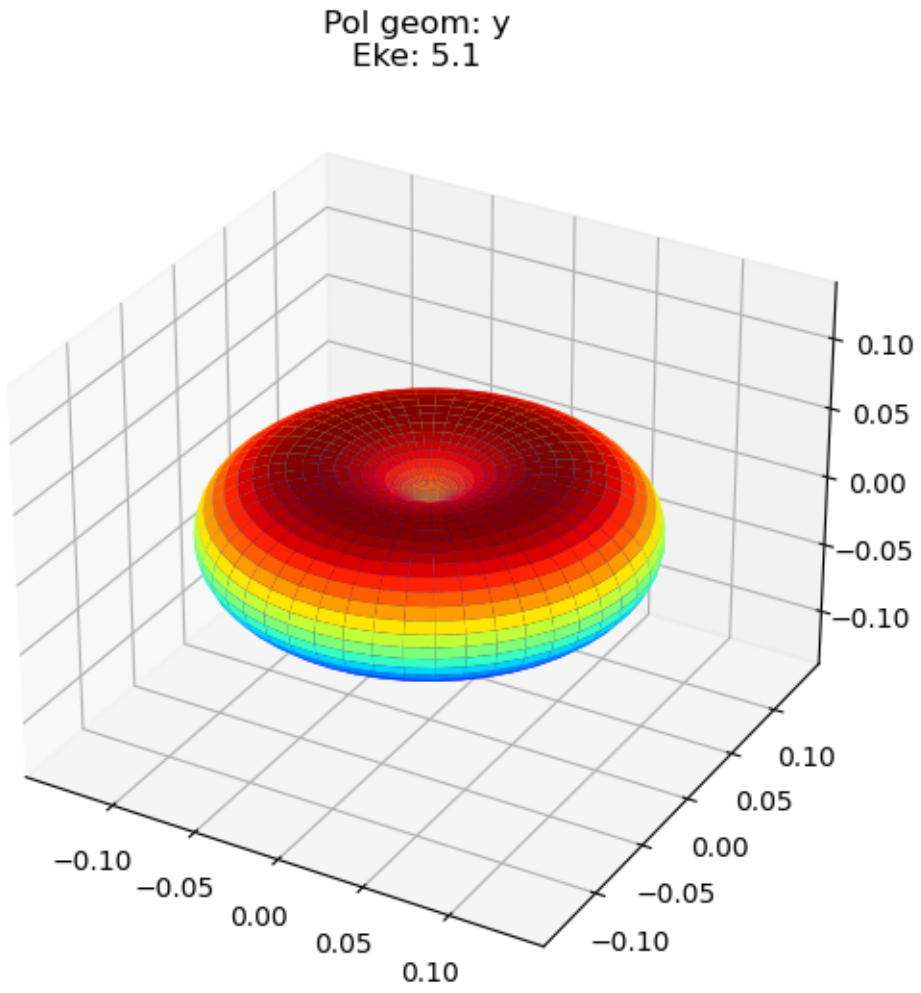
```
Found dims ('Labels', 't', 'Eke', 'BLM'), summing to reduce for plot. Pass selDims␣
↪to avoid.
Using default sph betas.
Sph plots: Pol geom: z
Plotting with mpl
Data dims: ('t', 'Eke', 'Phi', 'Theta'), subplots on Eke
Sph plots: Pol geom: x
Plotting with mpl
Data dims: ('t', 'Eke', 'Phi', 'Theta'), subplots on Eke
Sph plots: Pol geom: y
Plotting with mpl
Data dims: ('t', 'Eke', 'Phi', 'Theta'), subplots on Eke
```

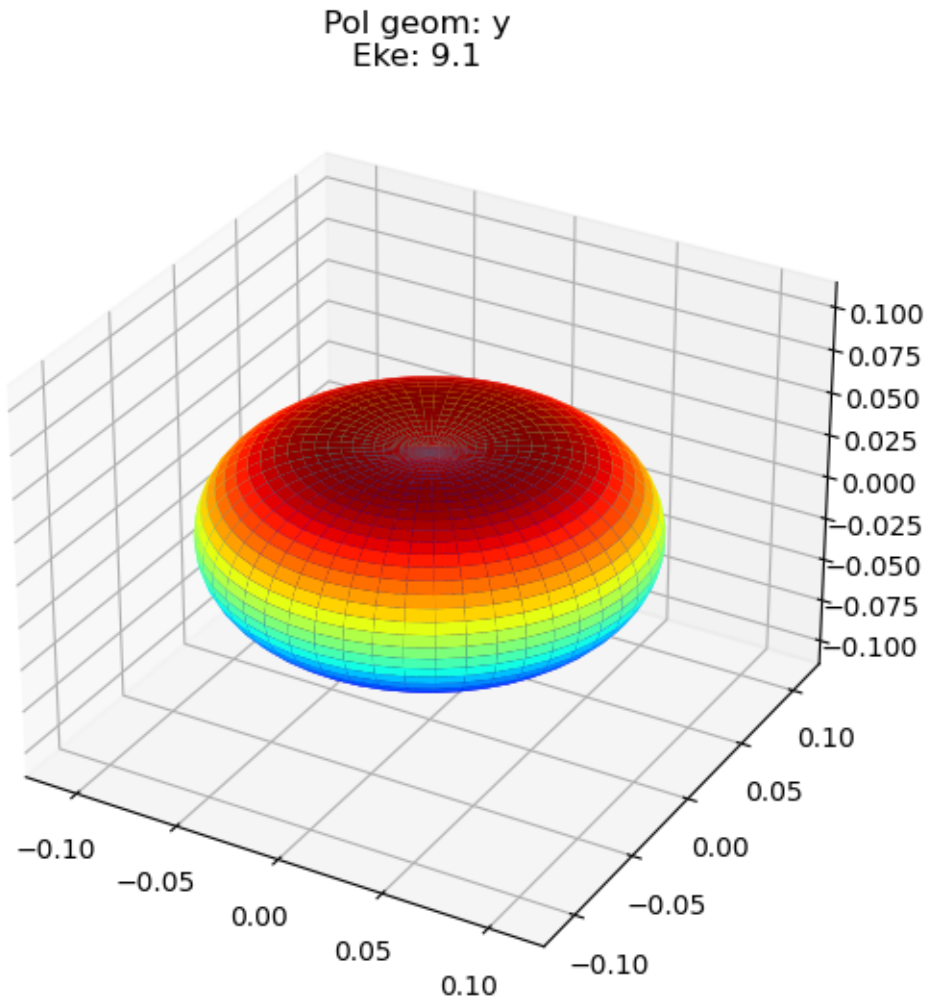



Pol geom: z
Eke: 9.1
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3


Pol geom: x
Eke: 5.1
0.2
0.1
0.0
−0.1
−0.2
−0.2
−0.1
0.0
0.1
0.2
0.2
0.1
0.0
−0.1
−0.2


Pol geom: x
Eke: 9.1
0.2
0.1
0.0
−0.1
−0.2
−0.2
−0.1
0.0
0.1
0.2
0.2
0.1
0.0
−0.1
−0.2


Pol geom: y
Eke: 5.1
0.10
0.05
0.00
−0.05
−0.10
−0.10
−0.05
0.00
0.05
0.10
0.10
0.05
0.00
−0.05
−0.10



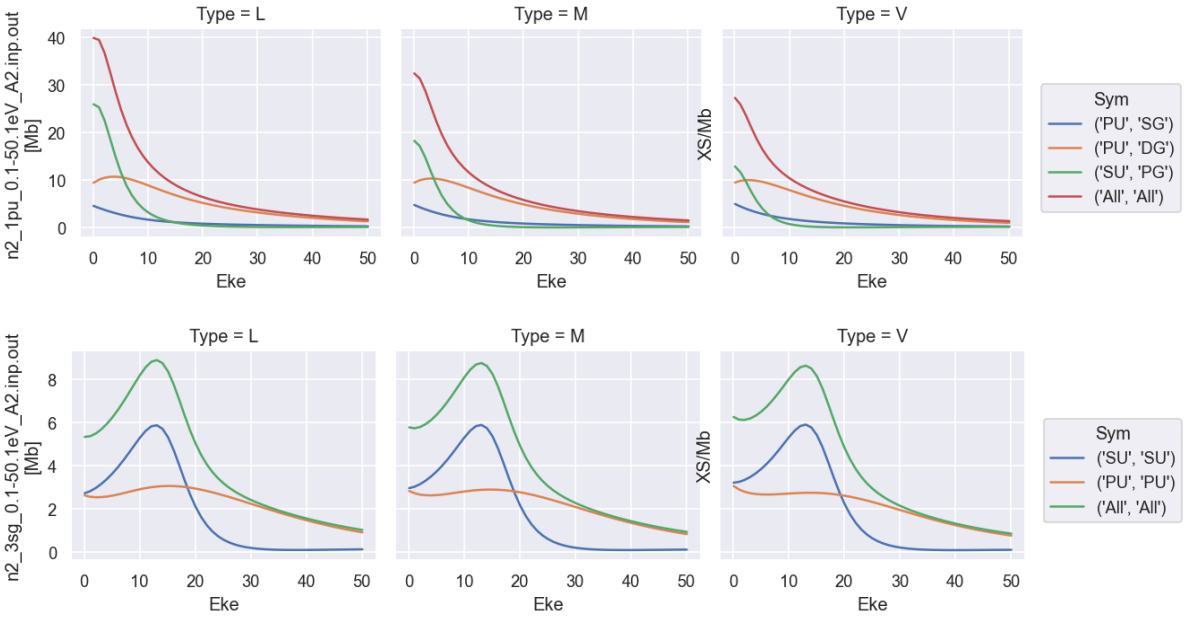
6.3 Additions

6.3.1 Plot styles for line-plots

To set to Seaborn plotting style, use `ep.hvPlotters.setPlotters()` (note this will be set for all plots after loading, unless overriden). Seaborn must be installed for this to function.

For more on Seaborn styles, see the Seaborn docs.

```
from epsproc.plot import hvPlotters
hvPlotters.setPlotters()

data.plotGetCro()
```

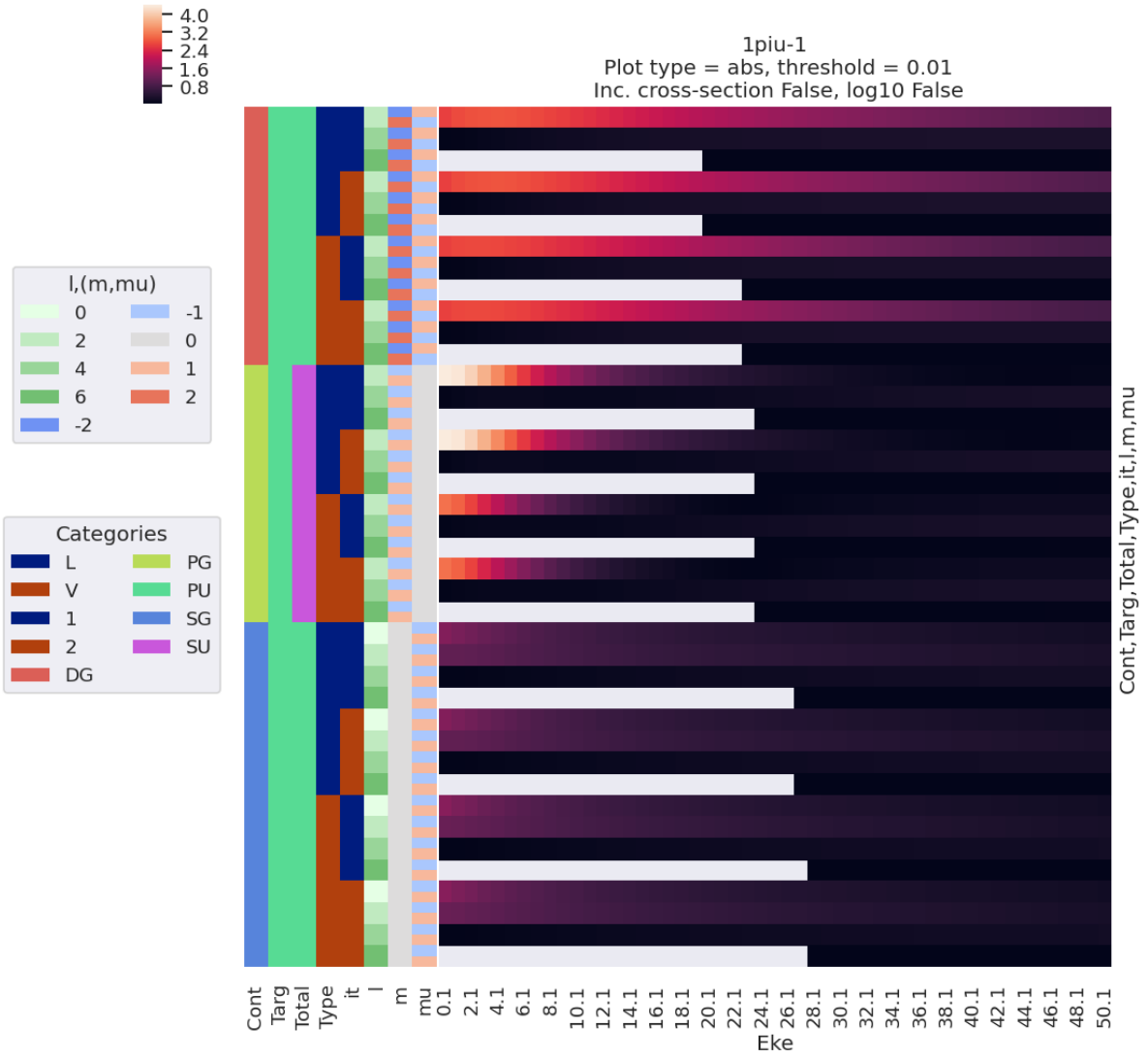


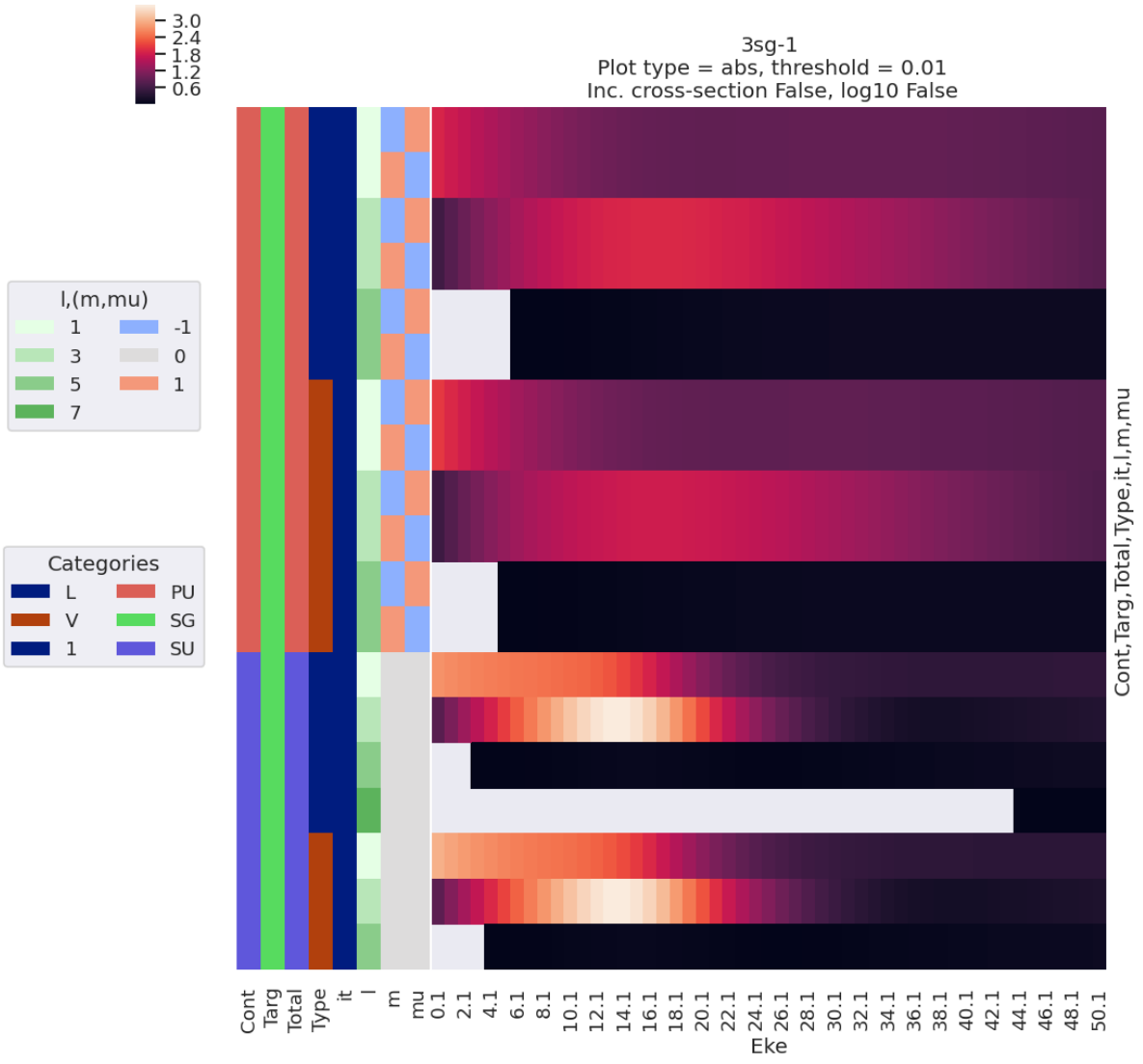
6.3.2 Matrix element plotting

For a full view of the computational results, use the `lmPlot()` method with the default, which correspond to `dataType=matE`.

```
data.lmPlot()
```

```
Plotting data n2_1pu_0.1-50.1eV_A2.inp.out, pType=a, thres=0.01, with Seaborn
Plotting data n2_3sg_0.1-50.1eV_A2.inp.out, pType=a, thres=0.01, with Seaborn
```

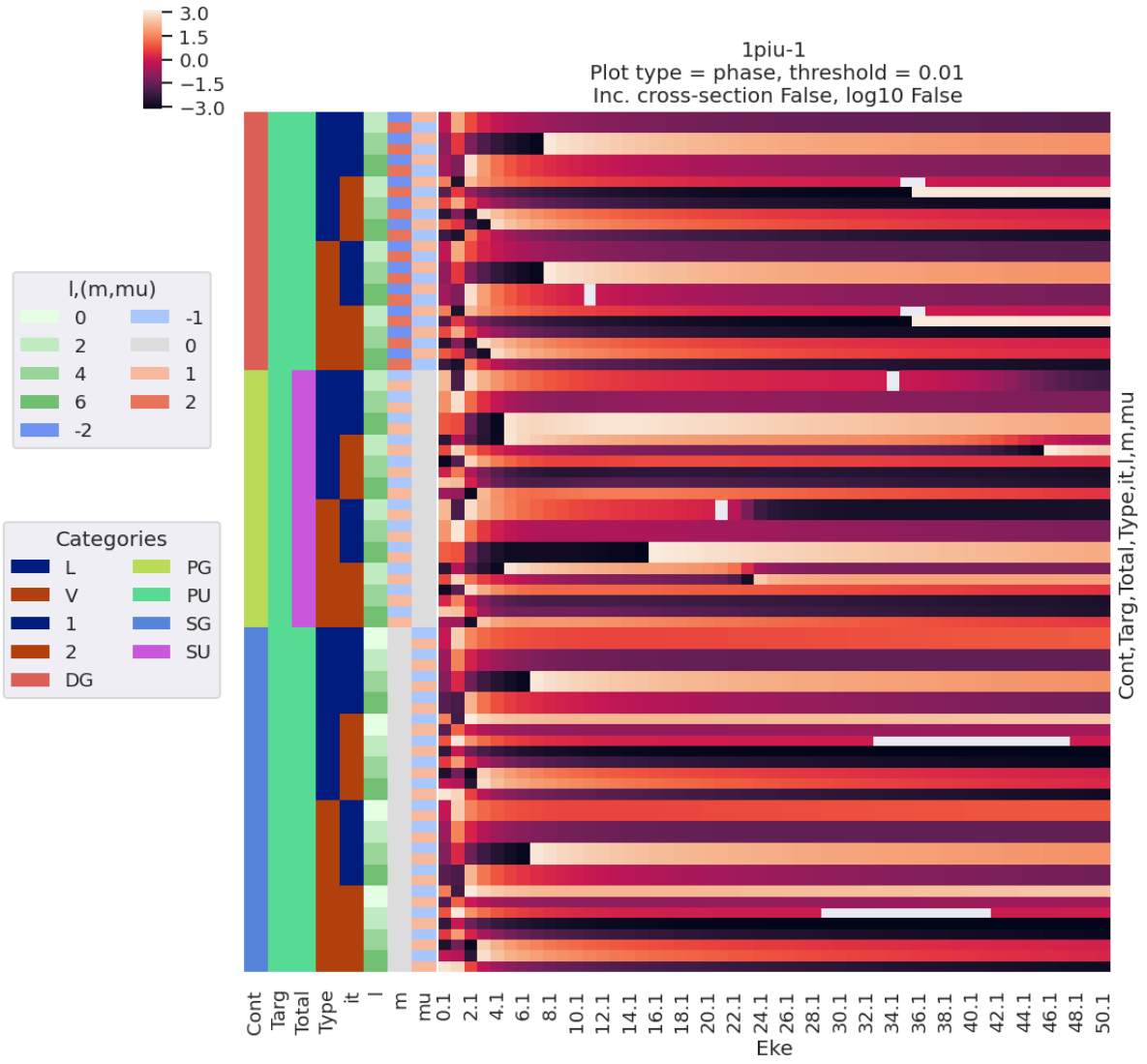

4.0
3.2
2.4
1.6
0.8
1piu-1
Plot type = abs, threshold = 0.01
Inc. cross-section False, log10 False
l,(m,mu)
0
2
4
6
-2
-1
0
1
2
Categories
L
V
1
2
DG
PG
PU
SG
SU
Cont,Targ,Total,Type,it,l,m,mu
Cont
Targ
Total
Type
it
l
m
mu
0.1
2.1
4.1
6.1
8.1
10.1
12.1
14.1
16.1
18.1
20.1
22.1
24.1
26.1
28.1
30.1
32.1
34.1
36.1
38.1
40.1
42.1
44.1
46.1
48.1
50.1
Eke

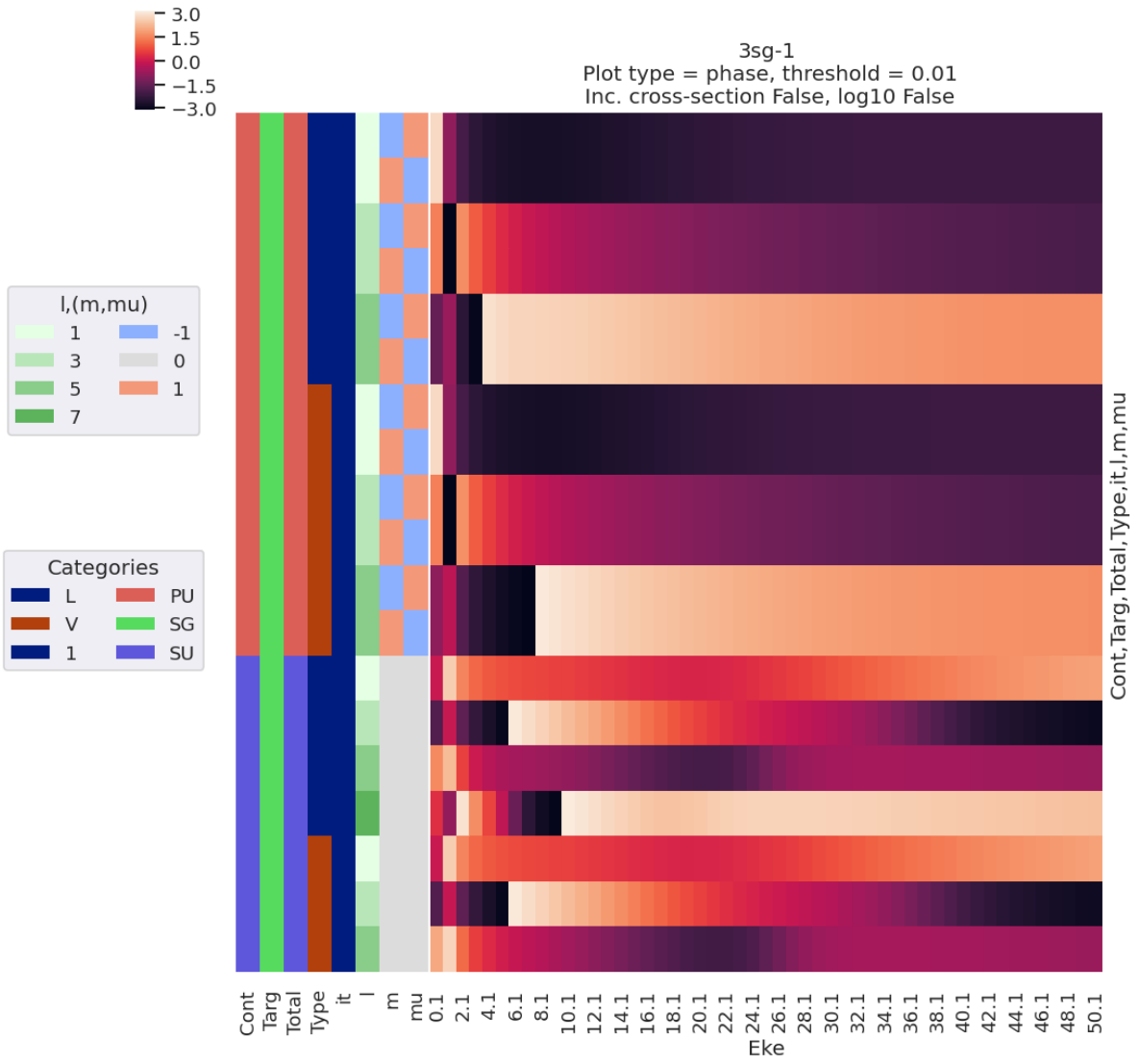


The default here plots abs values, but the same routine can be set for phase plotting.

```
data.lmPlot(pType='phase')
```

```
Plotting data n2_1pu_0.1-50.1eV_A2.inp.out, pType=phase, thres=0.01, with Seaborn
Plotting data n2_3sg_0.1-50.1eV_A2.inp.out, pType=phase, thres=0.01, with Seaborn
```


3.0
1.5
0.0
−1.5
−3.0
1piu-1
Plot type = phase, threshold = 0.01
Inc. cross-section False, log10 False
l,(m,mu)
0
2
4
6
-2
-1
0
1
2
Categories
L
V
1
2
DG
PG
PU
SG
SU
Cont
Targ
Total
Type
it
l
m
mu
0.1
2.1
4.1
6.1
8.1
10.1
12.1
14.1
16.1
18.1
20.1
22.1
24.1
26.1
28.1
30.1
32.1
34.1
36.1
38.1
40.1
42.1
44.1
46.1
48.1
50.1
Eke
Cont,Targ,Total,Type,it,l,m,mu



6.4 Versions

```
import scooby
scooby.Report(additional=['epsproc', 'xarray', 'jupyter'])
```

```
--------------------------------------------------------------------------------
  Date: Tue Oct 20 15:57:33 2020 Eastern Daylight Time

                OS : Windows
            CPU(s) : 32
           Machine : AMD64
      Architecture : 64bit
               RAM : 63.9 GB
       Environment : Jupyter
```

(continues on next page)

(continued from previous page)

```
  Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]

           epsproc : 1.3.0-dev
            xarray : 0.15.0
           jupyter : Version unknown
             numpy : 1.19.2
             scipy : 1.3.0
           IPython : 7.12.0
         matplotlib : 3.3.1
             scooby : 0.5.6

  Intel(R) Math Kernel Library Version 2020.0.0 Product Build 20191125 for
  Intel(R) 64 architecture applications
--------------------------------------------------------------------------------
```

CHAPTER

SEVEN

EPSPROC MATLAB DEMO

As noted in the main readme/intro doc, ePSproc originally started as a set of Matlab routines, before migration to Python in 2019. Although no longer maintained/updated since ~2018, the Matlab routines are still available as part of the ePSproc distribution. This notebook demos the routines, following the demo script `ePSproc_NO2_MFPADs_demo.m` (to run this code in a Jupyter environment, a Matlab kernel is required, e.g. Calysto's Matlab kernel).

Functionality:

- **Read raw photoionization matrix elements from ePS output files with "dumpIdy" segments**
- **Calculate MF-PADs from the matrix elements (ePSproc_MFPAD.m, see also ePSproc_NO2_MFPADs_demo.m)**
- **Plot MF-PADs**
- **Plot X-sects**
- **(Beta testing): Calculate MF-BLMs from matrix elements, see ePSproc_MFBLM.m**
- **(Under development): Calculate AF-BLMs from matrix elements.**

Source:

- **/matlab: stable matlab code (as per `release v1.0.1 <https://github.com/phockett/ePSproc/releases>__`).**
  - **a set of functions for processing (ePSproc*.m files)**
  - **a script showing demo calculations, `ePSproc_NO2_MFPADs_demo.m`**
- **/docs/additional contains:**
  - **the benchmark results from these calculations, `ePSproc_NO2_testing_summary_250915.pdf`**
  - **additional notes on ePS photoionization matrix elements, `ePSproc_scattering_theory_ePS_notes_011015.pdf`.**

See `ePSproc: Post-processing suite for ePolyScat electron-molecule scattering calculations <https://www.authorea.com/users/71114/articles/122402/_show_article>_` for more details.

7.1 Setup

```
cd('~/github/ePSproc')  % Change to ePSproc root dir.
```

```
ans =

    '9.4.0.813654 (R2018a)'
```

```
%% *** SETTINGS
%  Set up basic environment

% Name & path to ePS output file. In this version set full file name here, and␣
↪working directory below.
% fileName='no2_demo_ePS.out'  % OK for MFPAD testing, but only single E point
fileName='n2_3sg_0.1-50.1eV_A2.inp.out'

% Set paths for Linux or Win boxes (optional!)
if isunix
    dirSlash='/';
else
    dirSlash='\';
end

filePath=[pwd dirSlash 'data' dirSlash 'photoionization'];                    %␣
↪Root to working directory, here set as current dir/data/photoionization
fileBase=[filePath dirSlash fileName];    % Full path to ePS results file, here set as␣
↪current working direcory

scriptPath=[pwd dirSlash 'matlab' dirSlash];  % Add path to ePSproc scrips to Matlab␣
↪path list, here set as current dir/matlab
path(path,[scriptPath]);
```

```
fileName =

    'n2_3sg_0.1-50.1eV_A2.inp.out'
```

```
%% *** Read data
%  Variables:
%       rlAll contains matrix elements (from DumpIdy segments)
%       params contains various calculation parameters
%       getCro contains cross-section (from GetCro segments), if present

[rlAll, params, getCro]=ePSproc_read(fileBase);

params.fileBase=fileBase;
params.fileName=fileName;
```

```
*** Reading ePS output file
/home/paul/github/ePSproc/data/photoionization/n2_3sg_0.1-50.1eV_A2.inp.out
Found 102 sets of matrix elements
Read 102 sets of matrix elements (0 blank records)
Found 2 symmetries
    'SU'    'PU'
```

(continues on next page)

(continued from previous page)

```
Found 51 energies
Found 2 atoms
Found 18 data records
Found 3 sets of cross sections
```

```
% Matrix elements are stored in a structure
rlAll
```

```
rlAll =

  2x51 struct array with fields:

    eKE
    PE
    symm
    symmSet
    MbNorm
    rawIdyHead
    rawIdy1
    rlnlHead
    rlnl1
    rawIdy2
    rlnl2
    pWaveAll
```

```
% GetCro outputs (total cross-secions, LF betas)
getCro
```

```
getCro =

  1x3 struct array with fields:

    GetCro
```

```
% General calculation params
params
```

```
params =

  struct with fields:

        symmList: {'SU'  'PU'  'All'}
             eKE: [1x51 double]
         symmAll: {1x102 cell}
            eAll: [1x102 double]
        nRecords: 102
       nEnergies: 51
          nSymms: 2
           gLmax: 11
        blankRec: 0
        missingE: [1x0 double]
```

(continues on next page)

(continued from previous page)

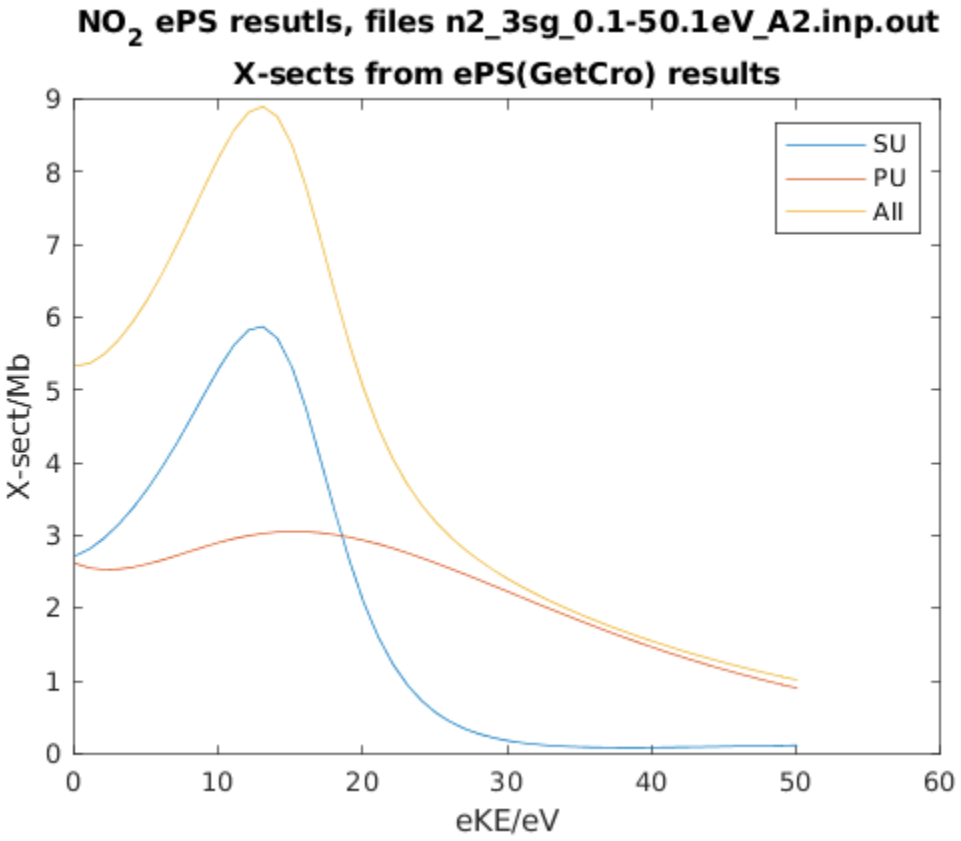
```
        pWaveAll: [144x51x3 double]
      pWaveAllMb: [144x51x3 double]
        LMallInd: [144x2 double]
          coords: {1x5 cell}
     dataRecords: {18x2 cell}
              IP: 15.5800
   GetCroHeader: {{1x1 cell}}
        fileBase: '/home/paul/github/ePSproc/data/photoionization/n2_3sg_0.1-50.
↪1eV_A2.inp.out'
        fileName: 'n2_3sg_0.1-50.1eV_A2.inp.out'
```

7.2 Plot cross-sections and betas

These are taken from the `GetCro` segments in the ePS output files, and correspond to results for an isotropic ensemble of molecules, i.e. observables in the lab frame (LF) for 1-photon ionization (see the ePS tutorial for more details).

```
%% Plot GetCro results for each symm & total

col=2;  % Select column from getCro output (see params.GetCroHeader)

figure('color',[1 1 1],'name','GetCro outputs');

for n=1:length(getCro)
    plot(getCro(n).GetCro(:,1)-params.IP,getCro(n).GetCro(:,col));
    hold on;
end

title({['NO_2 ePS resutls, files ' strrep(fileName,'_','\_')]; 'X-sects from␣
↪ePS(GetCro) results'});
xlabel('eKE/eV');
ylabel('X-sect/Mb');

legend([params.symmList 'Sum']);
```

```
Warning: MATLAB has disabled some advanced graphics rendering features by␣
↪switching to software OpenGL. For more information, click <a href="matlab:opengl(
↪'problems')">here</a>.
Warning: Ignoring extra legend entries.
> In legend>set_children_and_strings (line 646)
  In legend>make_legend (line 316)
  In legend (line 259)
```



7.3 MFPADs

These are calculated numerically from the matrix elements, for a given polarization geometry and symmetry (the method is the same as the python version of the routine).

```
%% *** Calculate MFPADs - single polarization geometry, all energies and symmetries
%  Calculate for specified Euler angles (polarization geometry) & energies

% Set resolution for calculated I(theta,phi) surfaces
res=100;

% ip components to use from ePS output (1=length gauge, 2=velocity gauge)
ipComponents=1;

% it components to use from ePS output (for degenerate cases), set an array here for␣
 ↪as many components as required, e.g. it=1, it=[1 2] etc.
it=1;

% Set light polarization and axis rotations LF -> MF
p=0;                % p=0 for linearly pol. light, +/-1 for L/R circ. pol.
eAngs=[0 0 0];      % Eugler angles for rotation of LF->MF, set as [0 0 0] for z-pol,␣
 ↪[0 pi/2 0] for x-pol, [pi/2 pi/2 0] for y-pol
polLabel='z';

% Run calculation - outputs are D, full set of MFPADs (summed over symmetries); Xsect,
 ↪ calculated X-sects; calcsAll, structure with results for all symmetries.
```

(continues on next page)

(continued from previous page)

```
[Xsect, calcsAll, pWaves]=ePSproc_MFPAD(rlAll,p,eAngs,it,ipComponents,res);

% Add pol labels - currently expected in plotting routine, but not set in MFPAD␣
↪routine
for n=1:size(calcsAll,2)
    for symmIn=1:size(calcsAll,1)
        calcsAll(symmInd,n).polLabel=polLabel;
    end
end
```

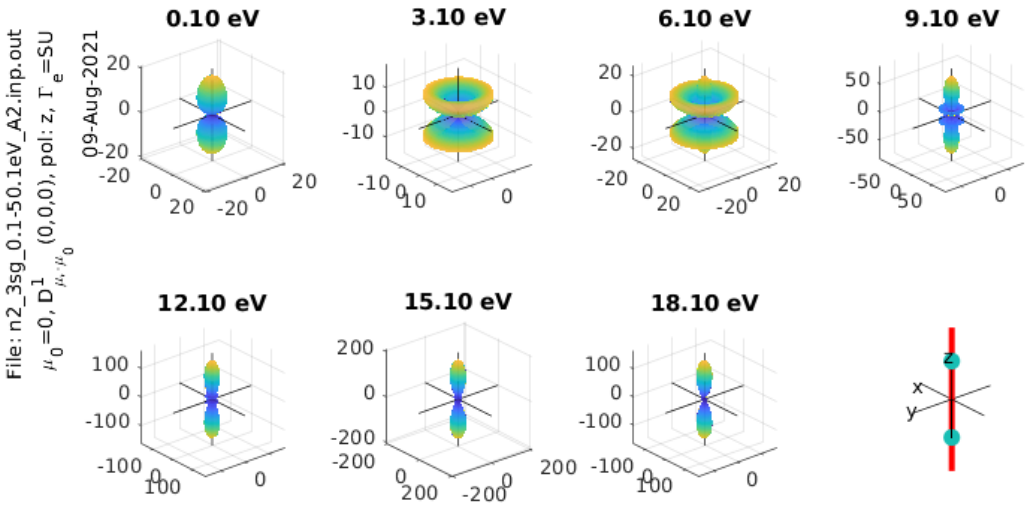
```
% Results are output as a structure, dims (symmetries, energies).
calcsAll
```

```
calcsAll =

  3x51 struct array with fields:

    D
    C
    Cthres
    eKE
    symm
    euler
    Xsect
    XsectD
    Rlf
    p
    Cind
```

```
%plot -s 800,400
```

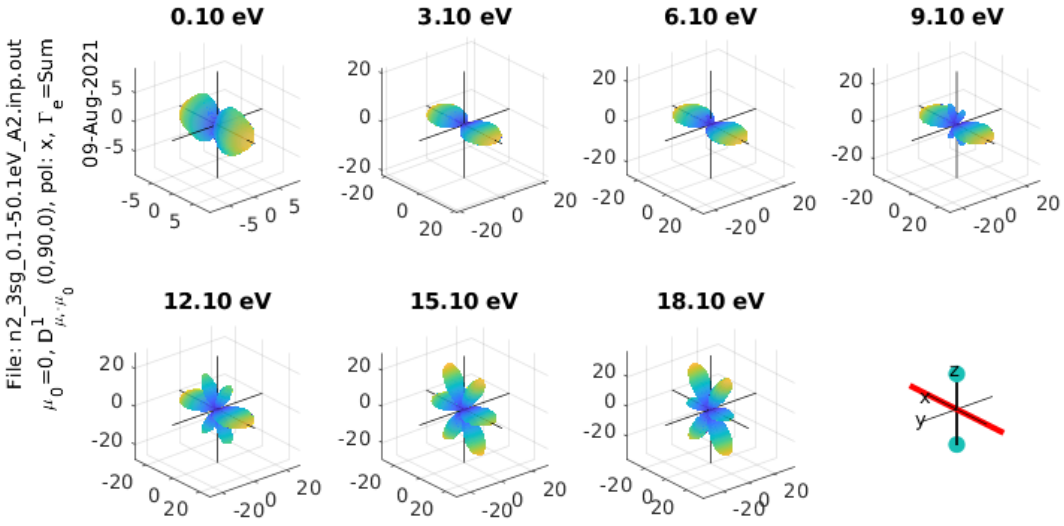
(Above is a line-magic for setting displayed plot size with the Calysto kernel - see the demo notebook for more.)

```
%% Plotting - MFPAD panel plots

% Set plot ranges
symmInd=1;      % Select symmetry (by index into calcsAll rows). Final symmetry state␣
↪is set as sum over all symmetries
% eRange=1;       % Select energies (by index into calcsAll cols)
eRange=1:3:20;

% Additional options (optional)
sPlotSet=[2 4];             % Set [rows cols] for subplot panels. The final panel␣
↪will be replaced with a diagram of the geometry
% titlePrefix='NO2 testing';  % Set a title prefix for the figure
titlePrefix='';

ePSproc_MFPAD_plot(calcsAll,eRange,symmInd,params,sPlotSet,titlePrefix);
% ePSproc_MFPAD_plot(calcsAll,eRange,symmInd,params,sPlotSet,'','n','off');
% ePSproc_MFPAD_plot(calcsAll,eRange,symmInd,params,[2 4],'','n','off');
```

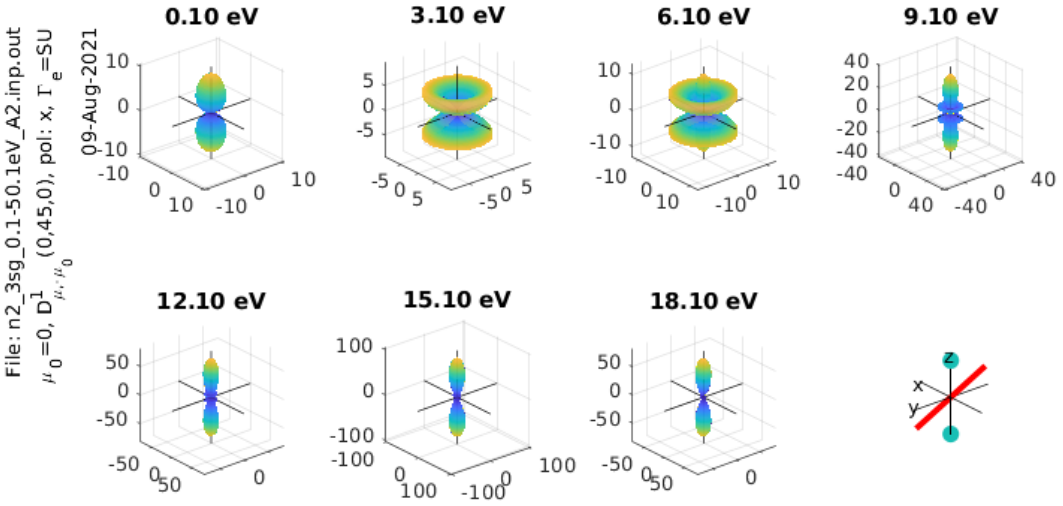


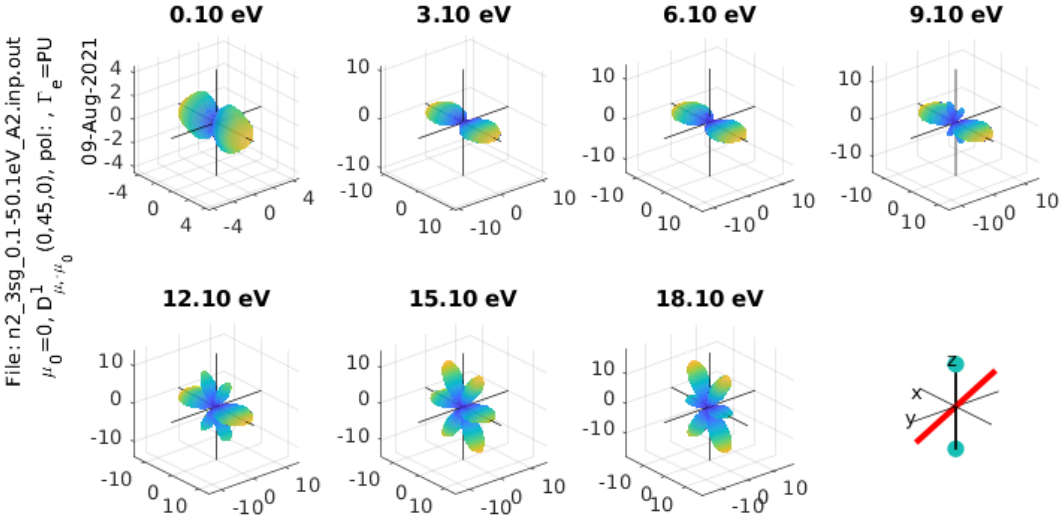

```
% Calculate & plot for a different polarization state
eAngs = [0 pi/2 0];  % x-pol case
polLabel = 'x';

[Xsect, calcsAll, pWaves]=ePSproc_MFPAD(rlAll,p,eAngs,it,ipComponents,res);

% Add pol labels - currently expected in plotting routine, but not set in MFPAD␣
↪routine
for n=1:size(calcsAll,2)
    for symmIn=1:size(calcsAll,1)
        calcsAll(symmInd,n).polLabel=polLabel;
    end
end

symmInd=3;
ePSproc_MFPAD_plot(calcsAll,eRange,symmInd,params,sPlotSet,titlePrefix);
```

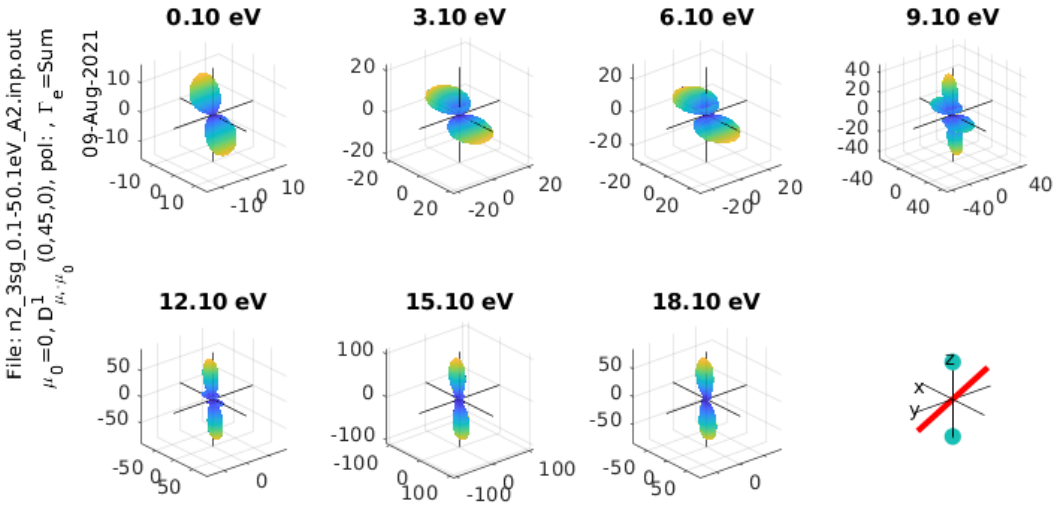
```
symmInd =

     3
```




```
% Calculate & plot for a different polarization state
eAngs = [0 pi/4 0];  % Diagonal pol case
polLabel = 'x';

[Xsect, calcsAll, pWaves]=ePSproc_MFPAD(rlAll,p,eAngs,it,ipComponents,res);

% Add pol labels - currently expected in plotting routine, but not set in MFPAD␣
↪routine
for n=1:size(calcsAll,2)
    for symmIn=1:size(calcsAll,1)
        calcsAll(symmInd,n).polLabel=polLabel;
    end
end

% Plot all symmetries
for symmInd = 1:3
    ePSproc_MFPAD_plot(calcsAll,eRange,symmInd,params,sPlotSet,titlePrefix);

end
```


File: n2_3sg_0.1-50.1eV_A2.inp.out
09-Aug-2021
μ_0=0, D^1_{μ,μ_0} (0,45,0), pol: x, Γ_e=SU
0.10 eV
3.10 eV
6.10 eV
9.10 eV
12.10 eV
15.10 eV
18.10 eV
z
x
y


File: n2_3sg_0.1-50.1eV_A2.inp.out
09-Aug-2021
μ_0=0, D^1_{μ,μ_0} (0,45,0), pol: , Γ_e=PU
0.10 eV
3.10 eV
6.10 eV
9.10 eV
12.10 eV
15.10 eV
18.10 eV
z
x
y



7.4 MF β_{LM}

```
%% *** Calculate MFPADs - single polarization geometry, all energies and symmetries
%  Calculate for specified Euler angles (polarization geometry) & energies

% Set resolution for calculated I(theta,phi) surfaces
res=100;

% ip components to use from ePS output (1=length gauge, 2=velocity gauge)
ipComponents=1;

% it components to use from ePS output (for degenerate cases), set an array here for as many components as required, e.g. it=1, it=[1 2] etc.
it=1;

% Set light polarization and axis rotations LF -> MF
p=0;                % p=0 for linearly pol. light, +/-1 for L/R circ. pol.
eAngs=[0 0 0];      % Eugler angles for rotation of LF->MF, set as [0 0 0] for z-pol, [0 pi/2 0] for x-pol, [pi/2 pi/2 0] for y-pol
polLabel='z';

% Run calculation - outputs are D, full set of MFPADs (summed over symmetries); Xsect, calculated X-sects; calcsAll, structure with results for all symmetries.
calcsAll=ePSproc_MFPAD(rlAll,p,eAngs,it,ipComponents,res);

% Add pol labels - currently expected in plotting routine, but not set in MFPAD routine
% for n=1:size(calcsAll,2)
%     for symmIn=1:size(calcsAll,1)
%         calcsAll(symmInd,n).polLabel=polLabel;
%     end
% end
```

```
plot(calcsAll.')
```