
Quantum Metrology with Photoelectrons Vol. 3 *Analysis methodologies*

Paul Hockett with Varun Makhija

Jul 31, 2023

CONTENTS

I Frontmatter	5
1 About the authors	7
2 Abstract	9
3 A note on book versions, formats and conventions	11
3.1 Versions	11
3.2 Conventions	11
3.3 Formatting	12
3.4 Numerics	12
II Part I - Theory & software	13
4 Introduction	15
4.1 Topical introduction: from quantum metrology to a generalised bootstrapping protocol	15
4.2 Context & aims for Vol. 3	19
5 Quantum metrology software platform/ecosystem overview	23
5.1 Analysis components	23
5.2 Additional tools	25
5.3 Python ecosystem (backends, libraries and packages)	25
5.4 Installation and environment set-up	27
5.5 General platform discussion	28
6 Theory	29
6.1 Photoionization dynamics	29
6.2 Symmetry in photoionization	31
6.3 Tensor formulation of photoionization	34
6.4 Density matrix representation	54
6.5 Molecular alignment	64
6.6 Observables: photoelectron flux in the LF and MF	68
6.7 Information content & sensitivity	78
7 Numerical methodologies for extracting matrix elements	89
7.1 Fitting methodologies	92
7.2 Fitting strategies	94

III Part II - Extracting matrix elements - numerical methods & case studies	97
8 Extracting matrix elements overview	99
8.1 General notes on the case studies	100
9 Basis sets for fitting	101
9.1 Symmetry-defined basis sets	101
9.2 Computationally-defined basis sets	101
9.3 Basis creation worked examples	102
9.4 Comparison with symmetry-defined and computational matrix elements	114
10 General fit setup and numerics	117
10.1 Init and pulling data	117
10.2 Setup with options	118
10.3 Compute AF- β_{LM} and simulate data	123
10.4 Fitting the data: configuration	124
10.5 Fitting the data: Running fits	127
11 Case study: Generalised bootstrapping for a homonuclear diatomic scattering system, N_2 ($D_{\infty h}$)	131
11.1 General setup	131
11.2 Load existing fit data or run fits	132
11.3 Post-processing and data overview	134
11.4 Data exploration	136
11.5 Classify candidate sets	137
11.6 Explore candidate result sets	139
11.7 Parameter estimation & fidelity	143
11.8 Using the reconstructed matrix elements	145
12 Case study: Generalised bootstrapping for a linear heteronuclear scattering system, OCS ($C_{\infty v}$)	151
12.1 General setup	151
12.2 Load existing fit data or run fits	152
12.3 Post-processing and data overview	155
12.4 Data exploration	158
12.5 Classify candidate sets	158
12.6 Explore candidate result sets	160
12.7 Parameter estimation & fidelity	169
12.8 Using the reconstructed matrix elements	170
13 Case study: Generalised bootstrapping for a general asymmetric top scattering system, C_2H_4 (D_{2h})	177
13.1 General setup	177
13.2 Load existing fit data or run fits	178
13.3 Post-processing and data overview	181
13.4 Data exploration	184
13.5 Classify candidate sets	185
13.6 Explore candidate result sets	186
13.7 Parameter estimation & fidelity	190
13.8 Using the reconstructed matrix elements	192
14 Case studies: summaries, conclusions and outlook	199
14.1 General notes on fitting methodologies for the case studies	199
14.2 Retrieval fidelity	201
14.3 MF-PAD retrieval	202
14.4 Conclusions and outlook	203

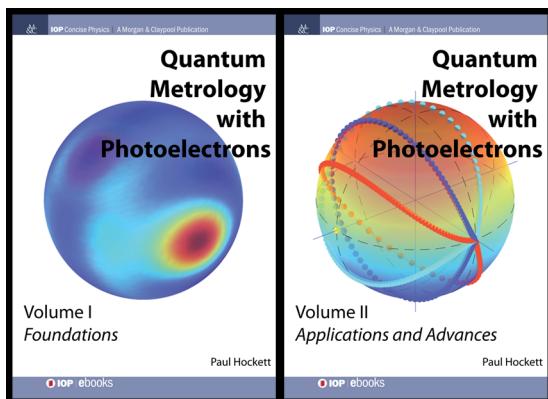
IV Backmatter	205
15 Bibliography	207
16 Glossary	209
V Test pages	213
17 Build versions and config tests	215
17.1 Versions	215
17.2 Docker build env	216
17.3 Book versions	216
17.4 Github pkg versions	217
17.5 Full conda env	218
Bibliography	229
Index	239

Quantum Metrology with Photoelectrons Volume 3: *Analysis methodologies*, an open source executable book by Paul Hockett with Varun Makhija. 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](#).

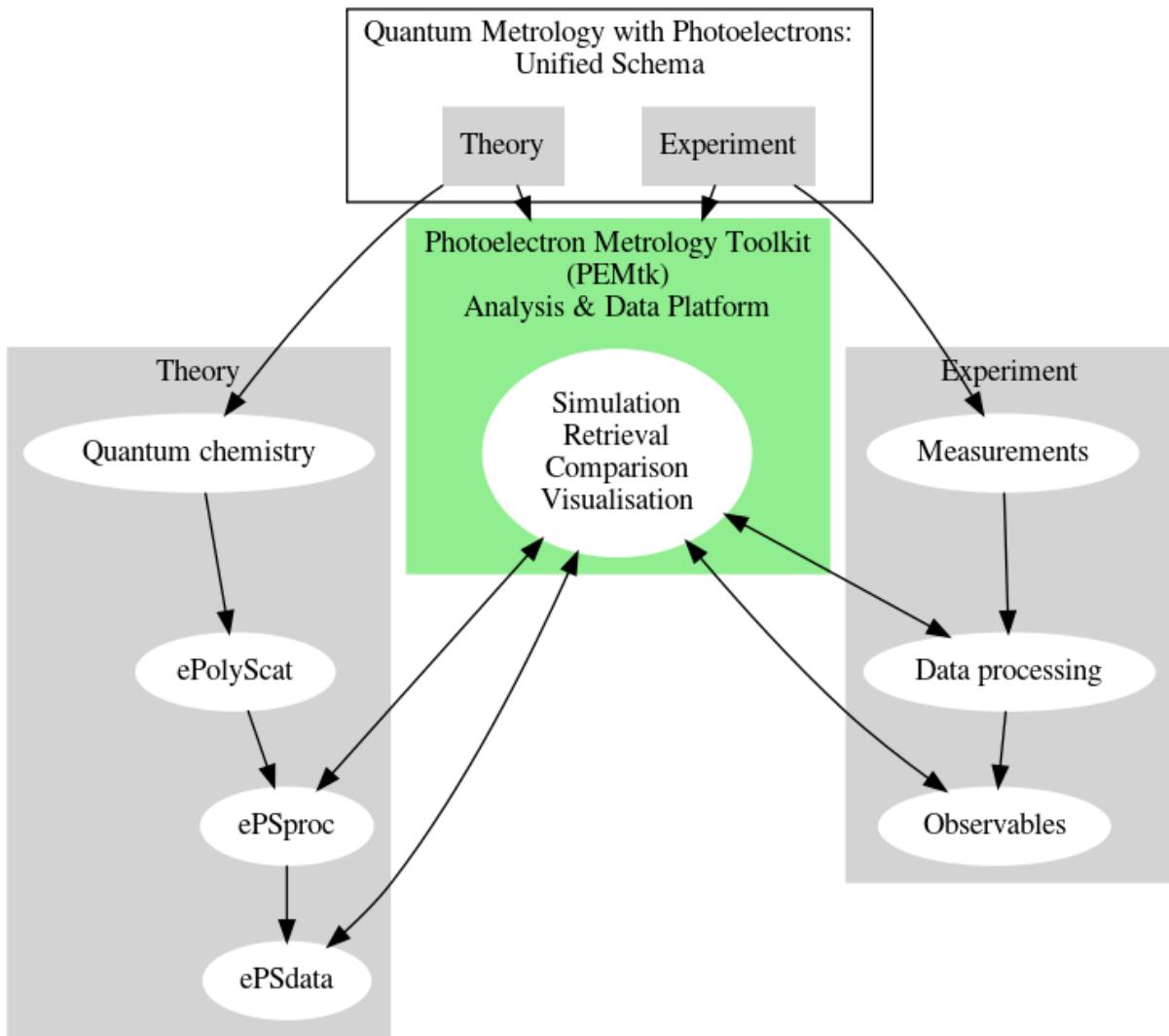
Technical details

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**

ABOUT THE AUTHORS

Paul Hockett and Varun Makhija's interests span a gamut of topics in atomic, molecular and optical (AMO) and quantum physics, at the intersection of spectroscopy, photoionization, (ultrafast) optics, molecular control and dynamics, and metrology. Paul's recent work has focussed on photoelectron metrology, with the main aims of matrix element retrieval, as outlined in this series (*Quantum metrology with photoelectron*) of books. Varun's work has focussed primarily on rotational wavepacket methodologies and reconstruction, and application of these techniques to molecular systems, including full excited state density matrix reconstruction and applications in photoelectron metrology.

Paul is a research scientist at the National Research Council of Canada, Ottawa, Canada,

<https://orcid.org/0000-0001-9561-8433>

Varun is a faculty member at the University of Mary Washington, Virginia, USA,

<https://orcid.org/0000-0002-4975-4888>

CHAPTER**TWO**

ABSTRACT

The overall aim of Quantum Metrology with Photoelectrons Vol. 3 is to expand, explore, and illustrate, new computational developments in quantum metrology with photoelectrons: specifically, the application of new python-based tools to tackle problems in photoionization matrix element retrieval. Part I details the topic, theory and computational methods; Part II provides further numerical details and case-studies.

The book itself is fully open-source, and written as a set of Jupyter Notebooks. All the material herein is available directly to readers via a Github repository [Quantum Metrology Vol. 3 \(Github repo\)](#). Any example script, page, chapter - up to and including the full book - can be executed and modified by readers to further explore the topic interactively, and provide a foundation which can be adapted to apply the methodology to new problems. Further details can be found in Sect. 4.2.

A NOTE ON BOOK VERSIONS, FORMATS AND CONVENTIONS

3.1 Versions

This book exists in multiple formats, which are not all equal:

1. Jupyter notebooks. The original form, interactive computational notebooks includes text, executable code and full outputs. Source notebooks are available via *Quantum Metrology Vol. 3* (Github repo).
2. HTML pages. Compiled from the notebooks, include interactive figures and most computational outputs. The HTML version is available at *Quantum Metrology Vol. 3 (HTML version)*.
3. PDF and hard-copy. Standard static outputs, compiled from the notebooks. In this form some computational outputs are truncated or omitted for brevity and readability. Since some formats may not support hyperlinks, URLs to external references are also usually included in the bibliography - note that these may not always be the *full* URLs linked in the main text, and may only list the main index page of a given site in some cases. Some figures may also be omitted.

3.2 Conventions

Note: In many cases where there is significant truncation of the presentation in the PDF, a note like this may be included.

E.g. *Full tabulations of the parameters available in HTML or notebook formats only.*

Code (Python) appears in formatted cells, with comments, and outputs below the cell:

```
# Example comment in code
value = 3*3
print(f'This is a code cell, value={value}')
```

```
This is a code cell, value=9
```

In HTML and PDF formats some code cells that appear in the source notebooks may be hidden or removed, or have outputs hidden or removed. This is usually for brevity - e.g. to remove additional code-only examples that are only useful when working directly on the code, or repeated code - or to hide additional formatting commands required only for Jupyter Book builds. All code cells are annotated to indicate their contents.

Code-related terms in the text, e.g. the names of functions, packages etc., usually appear as in-line blocks, e.g. Numpy, and may additionally be linked to relevant web resources, e.g. [Numpy](#).

For more details on the aims, tools and build-chain, see [Sect. 4.2.2](#).

3.3 Formatting

In some cases additional formatting is required for defining Jupyter Notebook to HTML and PDF outputs (via the Jupyter Book build-chain, see [Sect. 4.2.2](#)), in particular the `glue` command is used for formatting figure outputs with captions. In general use these are not required, but will transparently display figures when executed in the Jupyter Lab environment. Note that glued tables from Pandas DataFrames are not nicely rendered in the HTML format, but interactive HTML output is usually included too, although this may be hidden in the cell above the glued table.

3.4 Numerics

At the time of writing the main code-bases used in this work (see [Sect. 5](#)) are still in active development, bugs, inconsistencies and errors cannot, therefore, be ruled out in the numerical examples. However, the case for 1D alignment and reconstruction has been well-tested in the past (e.g. Refs. [1, 2, 3]), so is expected to be accurate; cases with 3D alignment are presented in a provisional context, with caveats as above, although the general methodology as demonstrated is robust.

Part II

Part I - Theory & software

INTRODUCTION

The overall aim of *Quantum Metrology with Photoelectrons Vol. 3* is to expand, explore, and illustrate, new computational developments in 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 interactively, and provide a foundation which can be adapted to apply the methodology to new problems.

Whilst this volume aims to provide a self-contained text, and focuses on computational examples which may be used without extensive background knowledge, a brief contextual introduction is presented here (Sect. 4.1 below), and the necessary core physics, as well as some recent extensions, is also presented herein (Chapter 6). The unfamiliar reader is referred to *Quantum Metrology Vol. 1* [4] for a more detailed introduction to the physics, and as a more general gateway to the literature. Following the topical introduction, the remainder of Part I introduces the main computational and software tools (Chapter 5), recent theory developments (Chapter 6), and concludes with a general overview for approaching matrix element retrieval numerically (Chapter 7).

Part II details the application of these tools to a few specific cases, including a general guide to setting up and running the *Photoelectron Metrology Toolkit* [5] fitting routines (see Chapter 8 for an outline), then proceeding with a (relatively) simple homonuclear diatomic example (Chapter 11), and escalating in complexity to a the most general polyatomic asymmetric top case.

4.1 Topical introduction: from quantum metrology to a generalised bootstrapping protocol

There are two core topics at the heart of this work, specifically photoelectron spectroscopy (and associated experimental, theoretical and analysis methodologies) and quantum metrology in general. To briefly (re)introduce these topics, and contextually frame the work discussed herein, some brief comments from *Quantum Metrology Vol. 1* [4] are reproduced below; the reader is referred to the introductory chapters of *Quantum Metrology Vol. 1* [4] for a lengthier treatment, and an introductory video to *Phase-sensitive Photoelectron Metrology* can be found online (see also Refs. [6, 7, 8] for further introductory presentation videos, materials and resources around the topic).

4.1.1 Quantum metrology with photoelectrons

To set the general context, consider quantum metrology in general...

Quantum metrology can be loosely defined as any class of experiment which provides detailed information on quantum mechanical properties (phases, coherences, entanglement etc.) of a system. To stay with the spirit of modern metrology, this definition can further be refined to measurements which provide high-resolution quantum information; a clear contemporary example is therefore experimental methodologies which provide full quantum state reconstruction (e.g. quantum tomography), and/or make use of quantum mechanical properties as a tool for measurement (e.g. atom interferometry). Traditional high-resolution spectroscopies

may also fit within this definition in some cases, although in the majority of cases high-resolution spectroscopic measurements provide transition line-strengths and energies, but lack sufficient information for a full determination or reconstruction of the underlying quantum state.

[...]

...at what point does a measurement of a quantum mechanical system become quantum metrology? A pragmatic view on this is that the complete quantum state of the system must be capable of *unique definition from the experimental measurement(s)*. This is pragmatic in the sense that it leaves the door open for both *inferred* and *direct* reconstruction techniques. In the former case, the experimental data informs the theory and analysis, but is not directly ‘analysed’ or ‘inverted’ to provide or reconstruct the full quantum information; in the latter case one obtains the desired quantum mechanical information from the measurement in a more ‘direct’ fashion (which may, admittedly, still remain as a rather convoluted process, depending on the level of theoretical input required). Traditional spectroscopies again provide a touchstone here - high-resolution spectroscopy measurements can be compared with models or ab initio computations to provide quantum mechanical details of a system, but typically do not directly provide this information from a set of measurements alone. In this sense they fit a pragmatic definition of quantum metrology, but not a more specific definition of quantum metrology as a (somewhat) direct empirical technique.

[...]

In summary, while quantum metrology can come in many flavours, at heart it might be considered as any set of measurements (and associated analysis methodologies) which provide detailed (quantitative) quantum mechanical information on a given system of interest - ideally with little or no restriction on the complexity of the system - and it is discussed in this spirit herein.

---*Quantum Metrology* Vol. 1 [4], Chpt. 1

And, for the specific case of photoionization...

... both *ab initio* methods and *high-dimensionality measurements* (combined with detailed *analysis methodologies*) can nonetheless provide detailed information on the photoionization dynamics. Although the simple analogy with Young’s double-slit [i.e. basic two-path interferometry] fails, the resulting photoelectron flux, measured spatially, remains, in essence, a self-referencing angular interferogram of the continuum wavefunction. In a more abstract sense, the basic interferometer paradigm can be extended to the general ‘photoionization interferometer’, one just has to keep in mind that there are now potentially many, many channels. In the most basic sense, the energy and angle resolved interferograms - the photoelectron flux as a function of energy and angle $I(E, \theta, \phi)$ - which may be measured, are nothing more than an interferometric measurement sensitive to the relative phases of the different angular momentum components.

[...]

In the photoionization community, the angular interferograms (which will usually be considered at a single energy E) are photoelectron angular distributions (*PADs*), and have long been used as a means to learn about the process of photoionization. In this context, *PADs* measured for a range of experimental parameters can provide a dataset with sufficient information content to determine the magnitudes and phases of the photoelectron wavefunction, hence the photoionization dynamics may be reconstructed from the measurements in favourable cases. This class of measurement is traditionally termed a *complete photoionization experiment*, although the exact nature of the completeness may vary. The phase-sensitivity of photoelectron interferograms have also been used in complementary fashions in other contexts, including as a means to probe the phase-shift induced by a specific prepared pathway, and control in multipath schemes, and in many other regimes.

[...]

... the combination of a phase-sensitive quantum mechanical observable - photoelectron interferograms - with modern experimental and computational techniques provides the tools required for a full quantum metrology based on this class of measurement. Following the above discussion and definitions, a full metrology technique is one which allows both the *intrinsic* and *extrinsic/dynamic* quantum mechanical properties

of the system under study to be obtained/reconstructed from a measurement, or set of measurements. In the simplest case, one might seek to understand just the intrinsic photoionization dynamics of a scattering system (e.g. the magnitudes and phases of the various pathways [...]), while in more complex cases the intrinsic properties are part of a probe process for additional properties or dynamics of the system [...]. In all cases, the key is measurement (and possibly control) with a high information-content technique, and a detailed understanding of the processes involved.

---*Quantum Metrology Vol. 1 [4], Chpt. 1*

In summary, the focus of the work presented herein is the quantitative analysis of phase-sensitive observables from photoionization, specifically photoelectron angular distributions (*PADs*) as a function of energy, angle and time, which will be denoted $I(\epsilon, t, \theta, \phi)$ in general herein. These *photoelectron interferograms* are introduced in more detail (including examples) in Sect. 6.6.

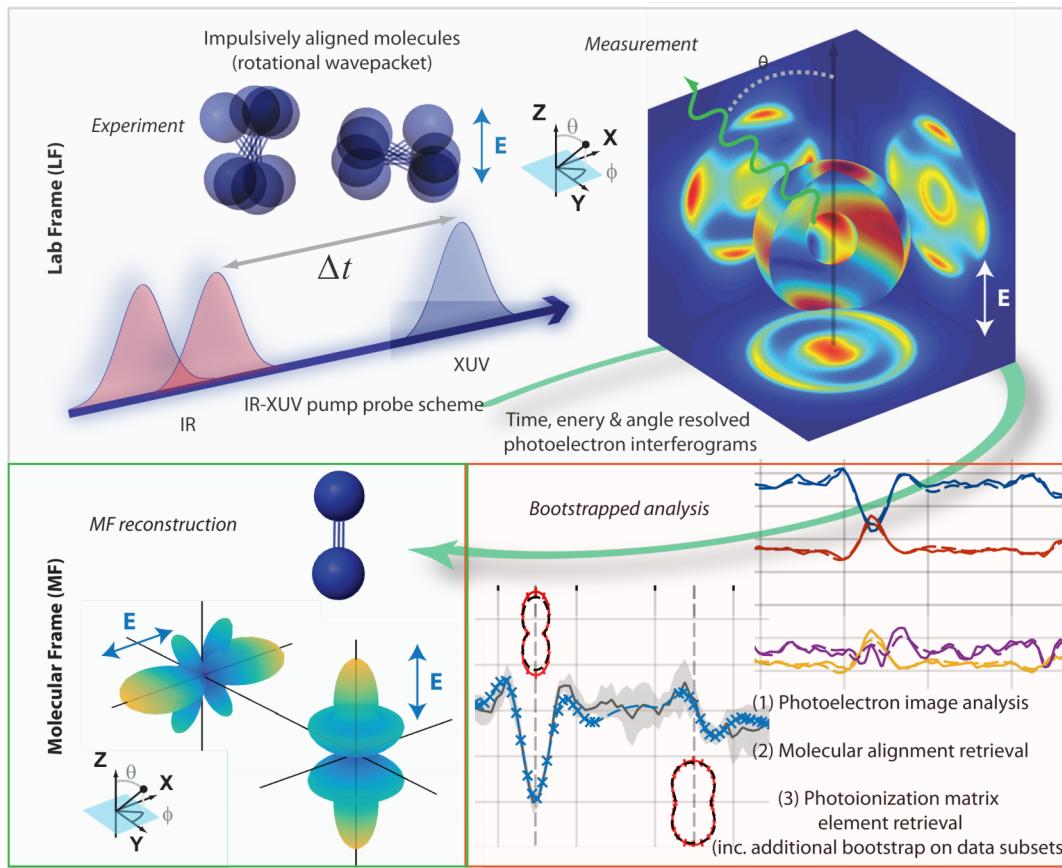


Fig. 4.1: Conceptual outline for the generalised *bootstrap retrieval protocol* for *radial matrix elements*, including retrieval of *MF* properties, via a set of time-resolved measurements and suitable post-processing scheme. In the *LF/AF*, a set of laser pulses creates and probes an aligned distribution of molecules, and photoelectron images are measured (as a function of time, hence molecular alignment). The experimental data is analyzed through a multi-step “bootstrap” protocol to obtain matrix elements, which constitute a complete description of the photoionization event. These can further be used to obtain *MF* observables, for any polarization geometry. Note the coordinates shown will be used throughout the book, with the *LF* z-axis defined relative to the laser field (E) polarization, and the *MF* defined with the molecular axis aligned along the z-axis (and polarization geometries for the laser field (E) referenced to this axis) - see also Fig. 6.1 for a more detailed frame definition.

4.1.2 Generalised geometric metrology protocols

In order to develop a quantitative form of photoelectron spectroscopy, hence analyse photoelectron interferograms in the context of quantum metrology in general, a number of techniques have previously been investigated (see *Quantum Metrology* Vols. 1 & 2 [4, 9]). In general, any applicable technique involves the manipulation or control of parameters which affect the observables in analytically-defined (or otherwise well-characterised) ways; measurements over a set of suitable experimental or control parameters then provide the high information-content dataset required for a full characterisation of the system at hand. Typically, “geometric” (angular-momentum) properties of the system provide a suitable set of control parameters, and a number of experimental methodologies with different flavours of these parameters have been demonstrated (see *Quantum Metrology* Vol. 2 [9]). The main, outstanding, issue with previous techniques was the system-specific nature of many of the applications: ideally, one would like to make use of a generalised protocol, which is independent of the particulars of the system under study, hence does not require, for example, specific spectroscopic properties to be known and/or be experimentally accessible.

The main aim of the work in the current volume is the further development, deployment and demonstrations of, such a scheme. The focus is on one specific *high information-content* technique: the *generalised bootstrap retrieval protocol*, which makes use of experiments using rotational wavepackets (*RWP*) as a (geometric) control dimension, and time-resolved photoelectron measurements as a high-dimensionality, phase-sensitive observable; the combination of these measurements with a quantitative analysis methodology provides a (relatively) general route to a full quantum metrology with photoelectrons (a.k.a. complete photoionization experiments, a.k.a. quantum state reconstruction/quantum tomography). A brief introduction to the technique is given below, with theoretical and numerical techniques and demonstrations forming the remainder of this book; interested readers can find a longer topical introduction in *Quantum Metrology* Vol. 2 [9] (in particular Chpt. 11), and see also Ref. [1] for an experimental demonstration, and Refs. [2, 3] for a recent review in the context of molecular frame reconstruction.

As defined in *Quantum Metrology* Vol. 2 [9]:

For the analysis of the data [time-resolved photoelectron images from a rotationally-excited system], a ‘bootstrapping’ fitting approach was developed. This methodology [...] is illustrated conceptually in Fig. 4.1, and outlined in more detail in Fig. 7.1 [...] is comprised of two stages (potentially split into multiple steps) which allow for the separation of the two sets of unknowns (rotational and ionization dynamics), and provides a way to gradually bootstrap to the complete *MF* results via stages of analysis of increasing complexity. The nature of the fitting at each stage also provides a flexible methodology which can be used to carefully sample the solution hyperspace in order to ensure unique results, and fit with variable information content (experimental measurements) based on computational time and desired precision, based on a similar Monte-Carlo sampling manner to the methodologies already discussed [...]. In all cases, the underlying physics provides stringent limits on the form of the fitting functions, hence the fitting procedure at each stage is expected to be somewhat reliable by construction. Further analysis of the results, including comparison with experimental parameters, additional data not used in the analysis, and *ab initio* calculations all provide additional means of cross-checking and verifying the extracted physical parameters.

In terms of information content, the bootstrapping procedure gradually increases both the experimental information content - the number of geometric configurations of the photoionization interferometer - and the level of physical information included (hence fitted/extracted) in the analysis. In the first step, *ADMs* [i.e. molecular alignment properties] are determined without the need for accurate treatment of the ionization probe [10]; in the second step this information is used as part of the calculation to determine the ionization dynamics. In the sub-steps to determine the ionization dynamics, the experimental information content included in the analysis is gradually increased: the initial coarse steps in this procedure provide a base-line high information content, without the necessity for many temporal points, via the selection of highly distinct molecular axis distributions, while latter sub-steps allow for fine-tuning of the data by gradually coupling additional time-steps [or other constraints] into the analysis.

---*Quantum Metrology* Vol. 2 [9], Chpt. 11

The protocol as presented relies on certain steps to be experimentally realisable, and theoretically calculable:

1. Molecular alignment. Experimentally, this can be induced in any system with a strong (typically $> 10^{12} \text{~Wcm}^{-2}$),

short (few hundred femtosecond timescale or shorter) infra-red laser pulse, which (impulsively) creates a rotational wavepacket in the system. The exact nature of the wavepacket is laser pulse(s) and system dependent, but the technique is general.

2. Time-resolved photoelectron measurements. Experimentally, this requires - at minimum - a pump-probe type configuration, with the alignment pulse as the pump, and a time-delayed ionization pulse. This is a typical experimental configuration in many ultrafast laser labs, with pulses typically in the atto- or femto-second regime. Measurements may be made by any angle-resolved technique; photoelectron imaging (via velocity-map imaging, *VMI*) is currently the most accessible and widespread method.
3. Data analysis. This provides the bridge from high information-content measurements to a full quantum metrology (system characterisation). For the generalised bootstrapping approach this requires:
 - In order to characterise the rotational wavepacket created, alignment calculations of the system must be possible - such computations are increasingly tractable, if not already (somewhat) routine for a number of groups, although quite challenging and computational expensive for asymmetric top systems. These calculations are required in order to determine the rotational wavepacket (*RWP*) quantitatively, and in order to determine the corresponding *ADMs* from/for the experiment. *RWP* computation is beyond the scope of the current work, but their use in the bootstrapping protocol is discussed in [Chapter 7](#).
 - To characterise the *intrinsic* photoionization dynamics, a set of appropriate geometric basis functions must be computed, and combined with a sufficiently large dataset to enable extraction of the photoionization matrix elements via a fitting procedure. This is the main focus of Part II herein.
 - (Optional) In cases with *extrinsic* dynamics, these may further be analysed once the *intrinsic* dynamics have been characterised (or as part of that characterisation); this may, however, remain qualitative or semi-quantitative, depending on the system dynamics and complexity. This aspect of photoelectron metrology is beyond the scope of the current work, see discussion in *Quantum Metrology* Vol. 2 [9] for more discussion; recent examples of such work may be found in Refs. [11, 12], which investigated coherent electronic dynamics and complete quantum tomography of such a case.
4. (Optional) *Ab initio* computations may also be performed to compare with any or all of the previous steps; comparison with step 3 is particularly powerful, since one can compare fundamental quantum mechanical properties, as opposed to comparisons between measured and simulated observables, which may be integrated over many degrees of freedom of the system.

As detailed in the following section ([Sect. 4.2](#)), the main aims herein are the development of the methodology and toolkit to address the data analysis requirement (step 3), and to test this methodology for a range of example cases.

4.2 Context & aims for Vol. 3

4.2.1 Scientific aims

The work in the current volume primarily addresses recent developments towards a generalised bootstrapping protocol (i.e. the analysis of the data obtained by time-resolved photoelectron imaging measurements - or similar - from a rotationally-excited system), as previously outlined in *Quantum Metrology* Vol. 2 [9] Sect. 12.3; in particular the new *Photoelectron Metrology Toolkit* [5] has been built with the aim of making the protocol easy to use and apply to any given problem (as distinct from a bespoke/per-experiment analysis methodology and/or non-open-source codebase).

Part I herein includes a full precis of the new codebase ([Chapter 5](#)), along with the theory ([Chapter 6](#)) and numerics ([Chapter 7](#)) implemented towards this end; [Part II](#) provides multiple demonstrations of the new code-base, including the use of the toolkit to investigate more complex systems beyond the simple homonuclear diatomic case demonstrated to date.

Although the analysis herein focuses on the *RWP* case, the techniques and codebase developed are equally applicable to *any methodology or protocol making use of geometric properties as a variable*, and are built with all such problems in mind

- although minor modifications or extensions may be required for specific cases. Examples include other cases discussed in *Quantum Metrology* Vols. 1 & 2 [4, 9], e.g. the use of shaped laser pulses or the use of narrow-band, state-selected rotational excitation; in all cases the fitting/retrieval of matrix elements is carried out in the same manner, and the only changes required to the methodology are the choice of control variable and the corresponding input experimental or theoretical parameters - this is discussed further in Sect. 7.2.

4.2.2 Technical context and notes

As noted previously, Vol. 3 is somewhat distinct from the previous volumes in the series; although involving computational elements, *Quantum Metrology* Vols. 1 & 2 [4, 9] 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, [Photoelectron Metrology Toolkit](#) [5], introduced in more detail in Chapter 5. In order to fulfill this aim, Vol. 3 is an open source computational/computable document, with code directly available to readers to facilitate code transparency and reuse. This can be broken down as follows:

1. The book itself is written as a set of [Jupyter Notebooks](#) [13].¹
 - These are .ipynb files, usually running a [Python kernel](#) [14], each of which is designed such that it can be modified and used independently.
 - The full book is compiled from these sections using the [Jupyter Book](#) [15, 16] project platform,² which includes build tools and specifications for the specific flavour of [Markdown \(MyST\)](#) [17] used for the written text, and uses [Sphinx](#) [18] to build HTML and Latex/PDF flavours of the book.
 - The book source code is available via a Github repository, [Quantum Metrology Vol. 3 \(Github repo\)](#), which includes all the notebooks (in the doc-source directory), as well as installation and build notes for building the book itself.
 - An HTML version is also available at [Quantum Metrology Vol. 3 \(HTML version\)](#), which includes interactive figures.
2. The code examples *within* the book make use the new [Photoelectron Metrology Toolkit](#) [5].
 - In order to run code examples, a specific python environment (with various additional python packages) is required.
 - A full introduction to the relevant software tool-chain, including installation instructions for the codes used *within* the book, can be found in Chapter 5: [Quantum metrology software platform/ecosystem overview](#).
 - For a quick and easy installation, including all requirements, a Docker build of the platform can also be used, see Sect. 5.4.2: [Docker deployments](#) (see also the [Open Photoionization Docker Stacks](#) [19] for more related tools).
 - Once configured, any code examples from the book can be executed locally by the user/reader, and modified as desired. Each notebook is designed to be run as a stand-alone computational document.
3. The book can be regarded as, essentially, a manual and introduction to the [Photoelectron Metrology Toolkit](#) [5], as well as a foundation for those wishing to use (and potentially extend) the platform.
 - Part I covers all required background material, including details of the theory and numerical methods implemented.
 - Part II contains various examples of usage for a range of problems, and possible extensions.
 - Since no specific knowledge of the underlying physics should be required to use the software tools, they will hopefully also provide a suitable platform for new researchers wishing to learn about photoionization in general.

¹ For more information on the [Jupyter Project](#) and ecosystem, see [jupyter.org](#) and Refs. [13, 31, 32].)

² For more information see [jupyterbook.org](#) and Refs. [15, 16].

- It is, of course, also hoped that established researchers in the field will find the tools useful, and readily adaptable, to related problems of interest.
- Further documentation for the software tools (including the full PEMtk API) can be found online in the [PEMtk documentation](#) [20].

Finally, it is of note that whilst readers unfamiliar with the Jupyter and Python ecosystem may find that there is somewhat of a barrier to entry for making use of the platform, it is one that may be worth surmounting given the ubiquity of these tools, and general usefulness in modern scientific/data-science workflows; readers already making use of these tools in their work should have no difficulty, and the platform adheres to standard practice wherever possible. For an introduction to Python for data science, the [Python Data Science Handbook](#) provides a solid introduction, and is itself [an open source textbook available via Github](#) [21, 22].

4.2.3 A brief note on open science, open source software and reproducibility

A large part of the motivation for creating new tools, making them open source, and standardized, is down to the nature of the modern scientific endeavour, and the difficulty of reproducibility. In short, many projects now involve a substantial element of analysis making use of in-house codes, which are often inaccessible to other researchers; the same may apply to the raw datasets used. Whilst this may be justified in some cases, in general it leads to a lack of transparency and portability for the computational and/or data component(s) of research. The Open Science movement, in part, aims to challenge these issues - see, for further discussion, Refs. [23, 24, 25, 26, 27, 28, 29], or the [Wikipeadia Open Science page](#) for a brief summary [30].

As noted above, this book is fully open-source, including the full book source code, the computational libraries used and the datasets illustrated herein, and available via [Quantum Metrology Vol. 3 \(Github repo\)](#); this is detailed further in [Sect. 5](#). In order to aid portability and reproducibility, Docker builds are also available: these provide a means to define a full computational platform/stack, from the OS level and up, including all necessary dependencies and version; further details can be found in [Sect. 5.4.2](#).

In general, it is hoped that making such tools more accessible, usable, and interconnected - as well as making computational data generally available - will lower the barrier to entry to the field and create a useful foundation for interested researchers to work from.

QUANTUM METROLOGY SOFTWARE PLATFORM/ECOSYSTEM OVERVIEW

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 layer, and analysis routines, for photoelectron metrology, including new methods and tools, as well as a unifying bridge between these and existing tools. Fig. 5.1 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 \[5\]](#) library - implement general data handling for theory and experimental datasets (although not a full experimental analysis toolchain), along with 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.

5.1 Analysis components

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

- The [Photoelectron Metrology Toolkit \[5\]](#) (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 \[20\]](#).
- The [ePSproc codebase \[33, 34, 35\]](#) aims to provide methods for post-processing with *ab initio* radial dipole matrix elements from [ePolyScat \(ePS\) \[36, 37, 38, 39\]](#), or equivalent matrix elements from other sources (dedicated support for R-matrix results from [the RMT suite \[40, 41\]](#) is in development, for an overview of *ab initio* methods/packages see Ref. [42]). 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 \[35\]](#).

Warning: As [noted elsewhere](#), many components of the toolkit are still in active development, and some numerical details may change.

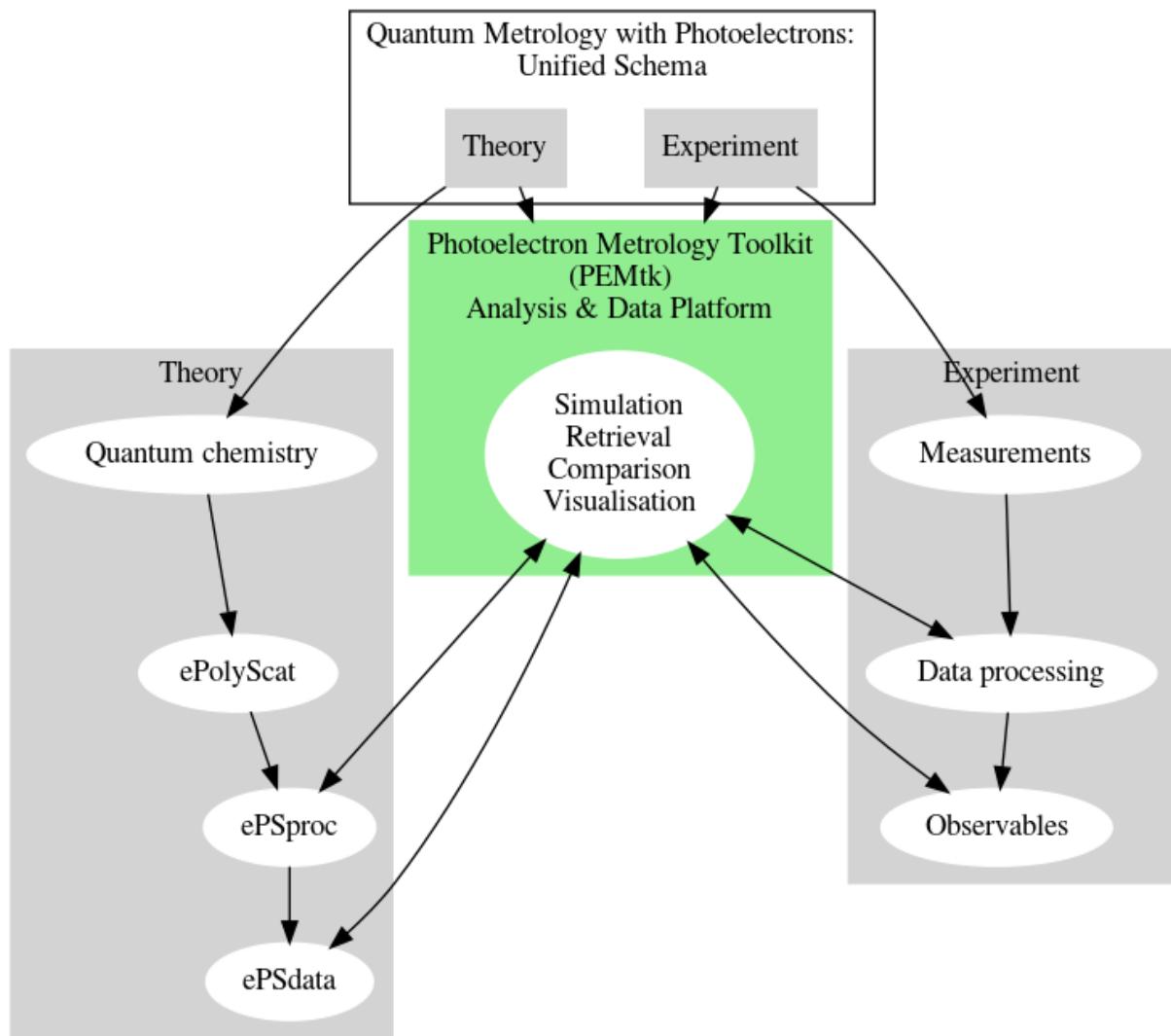


Fig. 5.1: Quantum metrology with photoelectrons ecosystem overview.

5.2 Additional tools

Other tools listed in Fig. 5.1 include:

- Quantum chemistry layer. The starting point for *ab initio* computations. Many tools are available, but for the examples herein, all computations made use of [Gamess \(“The General Atomic and Molecular Electronic Structure System”\)](#) [43, 44] for electronic structure computations, and inputs to ePolyScat.
 - For a python-based approach, various packages are available, e.g. [PySCF](#), [PyQuante](#), [Psi](#) can be used for electronic structure calculation, although note that some ePSproc [34] routines currently require Gamess files (specifically for visualisation of orbitals).
 - A range of other python tools are available, including [cclib](#) for file handling and conversion, [Chemlab](#) for molecule wavefunction visualisations, see further notes below.
- ePolyScat (ePS) [36, 37, 38, 39] 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](#) [39] and Refs. [36, 37, 38].
 - A Docker build is available (via the [Open Photoionization Docker Stacks](#) [19] project).
 - ePS (along with a range of other computational AMO tools) is also available online via the [AMOS gateway](#)¹ [45, 46, 47].
- ePSdata [48] is an open-data/open-science collection of ePS + ePSproc results.
 - 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 [ePSdata](#) [48] [Github project repository](#), and notebooks + datasets via [ePSdata Zenodo](#) [49]. 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 [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 \(CC BY-NC-SA 4.0\)](#) license, and are part of an ongoing [Open Science initiative](#).

5.3 Python ecosystem (backends, libraries and packages)

The core analysis tools, which constitute the [Photoelectron Metrology Toolkit](#) [5] platform, are themselves built with the aid of a range of open-source python packages/libraries which handle various backend functionality. Notably, they make use of the following key packages:

- General functionality makes use of the usual “Scientific Python” stack, in particular:
 - [Numpy](#) [50] for general numerical methods and data types.
 - [pandas](#) [51] for statistical methods, and various tabulation and sorting tasks.
 - [Scipy](#) [52] for some special functions and computational routines, particularly spherical harmonics and fitting routines (see below).
- General ND-array and tensor handling and manipulation makes use of the [Xarray library](#) [53, 54].
- Angular momentum functions

¹ Formerly known as the AMP gateway.

- Wigner D and 3js are currently implemented directly, or via the [Spherical Functions library](#) [55, 56], and have been tested for consistency with the definitions in Zare (for details see [the ePSproc docs](#) [35]). The Spherical Functions library also uses [quaternion](#) [57, 58] which implements a quaternion datatype in Numpy.
- Spherical harmonics are defined with the usual physics conventions: orthonormalised, and including the Condon-Shortley phase. Numerically they are implemented directly or via SciPy’s `sph_harm` function (see [the SciPy docs for details](#) [59]). Further manipulation and conversion between different normalisations can be readily implemented with the [pySHtools](#) library [60, 61, 62, 63]. See [Sect. 6.6.1](#) for examples.
- Symmetry functionality, specifically computing symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta, \phi)$ (see Eq. (6.37)), makes use of [libmsym](#) [64, 65] (symmetry coefficients) and [pySHtools](#) [60, 61, 62, 63] (general spherical harmonic handling and conversion). See [Sect. 6.6.2](#) for examples.
- Non-linear optimization (fitting), as used for the [bootstrap retrieval protocol](#) (to determine *radial matrix elements*):
 - Fitting is handled via the [lmfit](#) library [66, 67], which implements and/or wraps a range of non-linear fitting routines in Python, including classes for handling fitting parameters and outputs. In this work only the Levenberg-Marquardt least-squares minimization method has been used, which wraps [Scipy’s least_squares](#) functionality [59], hence this is the core numerical minimization routine for the demonstration cases herein. (See [Chapter 7](#) for further discussion of fitting methods.)
 - Basic parallelization for fitting routines is implemented using the [xyzpy](#) library [68], see [Chapter 10](#) for further details.
- For plotting a range of tools can be used, some of which are implemented/wrapped in the [Photoelectron Metrology Toolkit](#) [5], or can be used directly with [Xarray](#) data structures, including:
 - [Matplotlib](#) [69]: basic plotting, including [Xarray](#) direct plotters.
 - [Holoviews](#) [70]: used for data handling and interactive plotting, Holoviews is a general plotting tool which wraps various backends; [hvplot](#) [71] can also be used to provide additional Pandas and [Xarray](#) integration for Holoviews. Most of the plots herein use Holoviews.
 - [Bokeh](#) [72]: used for interactive plots, implemented in the [Photoelectron Metrology Toolkit](#) [5] via Holoviews wrappers/methods.
 - [Plotly](#) [73]: used in the the [Photoelectron Metrology Toolkit](#) [5] and the [ePSproc codebase](#) [33, 34, 35] for spherical polar plotting routines.
 - [Seaborn](#) [74, 75]: used for statistical methods and some specialist plots and styles, particularly the `lmPlot` routine in [ePSproc codebase](#) [33, 34, 35].
- Some specialist (optional) tools also make use of additional libraries, although these are not required for basic use; in particular:
 - For 3D orbital visualizations with [ePSproc](#) [34]: [pvista](#) for 3D plotting (which itself is built on VTK), [cclib](#) for electronic structure file handling and conversion, and methods based on [Chemlab](#) for molecule wavefunction (orbital) computation from electronic structure files are all used on the backend.
 - [Numba](#) [76] is used for numerical acceleration in some routines, although remains mainly experimental in [ePSproc](#) at the time of writing (an exception to this is the Spherical Functions library, which does make full use of Numba acceleration).

Code examples and further comments can be found as and when numerical examples are introduced in the text, particularly in [Chapter 6](#) and [Chapter 7](#).

5.4 Installation and environment set-up

5.4.1 Quick-start installation

For a basic installation, up-to-date version of Photoelectron Metrology Toolkit [5] and ePSproc codebase [33, 34, 35] can be installed directly from Github source using pip:

```
pip install git+https://github.com/phockett/ePSproc.git
pip install git+https://github.com/phockett/PEMtk.git
```

This should also install the required dependencies, although not all of the optional packages. (Note that `pip install ePSproc` will also work, and install the latest release from [the Pypi repository](#), but this may not be fully up-to-date compared to the Github source; `PEMtk` is not yet available via Pypi.)

For more details and other installation options, see the [ePSproc extended installation notes online](#), which includes directions for virtual environments (Anaconda, Venv).

5.4.2 Docker deployments

Docker [77] provides a useful mechanism for distribution of software as stand-alone containers (essentially minimal virtual machines), including definitions and versioning for everything from the operating system layer and up. Docker containers are both portable and reproducible, hence excellent tools for open science (see Sect. 4.2.3).

A Docker-based distribution of various codes for tackling photoionization problems is available from the [Open Photoionization Docker Stacks](#) [19] project, which aims to make a range of these tools more accessible to interested researchers, and fully cross-platform/portable. The project currently includes Docker builds for `ePSproc` and `PEMtk` (as well as `ePS` and other useful tools). These are based on the [Jupyter Docker Stacks project](#) [78], which includes Jupyter Lab, and also add all the required tools for the work illustrated herein.

A Docker container for this book is also available from the [Quantum Metrology Vol. 3 \(Github repo\)](#), which builds on the `ePSproc` and `PEMtk` container, and additionally includes the source notebooks and build tools (specifically [Jupyter Book](#) [15, 16] and related tools) as discussed in Sect. 4.2.2. It is suggested that readers interested in making use of this work start here as the easiest - and most comprehensive - methodology for getting the tools up and running.

5.4.3 Running examples

Any of the source notebooks can be run individually in a correctly configured Python/Jupyter environment (readers unfamiliar with Jupyter can find introductory materials online at [jupyter.org](#) [13]). Note that the majority of the imports are handled by a setup script, executed at the top of each notebook, for brevity and to ensure a standardized build:

```
%run '../scripts/setup_notebook.py'
```

For additional customization this script can be modified as desired. Depending on the build environment the full path to the script may also need to be set (the current code assumes the script will be located in the `qm3-repo/doc-source/scripts` directory, and notebooks run from their source dirs, e.g. `qm3-repo/doc-source/part1`).

5.5 General platform discussion

Note that, at the time of writing:

- Rotational wavepacket simulation is not yet implemented in the [Photoelectron Metrology Toolkit \[5\]](#), and these must be obtained via other codes. An initial build of the [limapack suite \[79\]](#) for rotational wavepacket simulations is currently part of the [Open Photoionization Docker Stacks \[19\]](#), but has yet to be used in this work.
- Fitting:
 - Fitting methods have not yet been carefully optimized, with only a general non-linear least squares method implemented. However, other methods should be easy to implement, either via the [lmfit library \[66, 67\]](#) or with other Python libraries or custom codes; optimization making use of Numba should also be possible.
 - Only the [bootstrap retrieval protocol](#) is currently implemented in [Photoelectron Metrology Toolkit \[5\]](#), along with associated analysis routines. However, the routines were written to be general and modular, so modification of the routines to other retrieval schemes should be fairly easy, and usually requires only (a) a function which computes the required basis set (e.g. [channel functions](#)) and (b) observables for the problem at hand. Examples are given in [Part II](#) for the [generalised bootstrap retrieval protocol](#), and [MF-PADs](#) based retrieval is also implemented in the codebase. For further details see the [PEMtk documentation \[20\]](#), particularly the fitting model backends and fitting MF and other datasets pages.
- The [Photoelectron Metrology Toolkit \[5\]](#) codebase is currently still under heavy development, so readers may wish to consult the ongoing [PEMtk documentation \[20\]](#) in future for changes and updates.
- For specific guides to various aspects of both codebases, see the relevant docs, which include full API guides. Some particular materials of introductory interest include:
 - A general quick-start demo can be found in the [ePSproc documentation \[35\]](#), specifically the [ePSproc class intro page](#).
 - For more details of the data structures used, see the [ePSproc documentation \[35\]](#), specifically the [data structures page](#).

Nonetheless, although both the codebase and methodologies are still under development, a range of numerical methods have been successfully trialled (as illustrated in [Part II](#) herein), and are now available to other researchers to make use of and build on.

CHAPTER
SIX

THEORY

In this chapter a number of fundamentals are outlined. Only a brief introduction to the necessary physics (which already has a rich literature) is presented, and the emphasis is instead on code and numerical examples. These are intended both to give readers an insight into the physics, and also illustrate aspects of the [Photoelectron Metrology Toolkit \[5\]](#) and [ePSproc codebase \[33, 34, 35\]](#) that can be used for these problems. These methods will form the basis for the numerical reconstruction work presented in [Part II](#).

Readers only interested in fitting problems from an experimental perspective may wish to skip most of this section; [Sect. 6.6:](#) and [Sect. 6.7:](#) should provide sufficient background for pure reconstruction problems.

6.1 Photoionization dynamics

The core physics of photoionization has been covered extensively in the literature, and only a very brief overview is provided here with sufficient detail to introduce the metrology/reconstruction/retrieval problem; the reader is referred to *Quantum Metrology Vol. 1* [4] (and references therein) for further details and general discussion.

Photoionization can be described by the coupling of an initial state of the system to a particular final state (photoion(s) plus free photoelectron(s)), coupled by an electric field/photon. Very generically, this can be written as a matrix element $\langle \Psi_f | \hat{\Gamma}(\mathbf{E}) | \Psi_i \rangle$, where $\hat{\Gamma}(\mathbf{E})$ defines the light-matter coupling operator (depending on the electric field \mathbf{E}), and Ψ_i , Ψ_f the total wavefunctions of the initial and final states respectively.

There are many flavours of this fundamental light-matter interaction, depending on system and coupling. For metrology, the focus is currently on the simplest case of single-photon absorption, in the weak field (or perturbative), dipolar regime, resulting in a single photoelectron. (For more discussion of various approximations in photoionization, see Refs. [80, 81].) In this case the core physics is well defined, and tractable (albeit non-trivial), via the separation of matrix elements into radial (energy) and angular-momentum (geometric) terms pertaining to couplings between various elements of the problem; the retrieval of such matrix elements is then a well-defined problem in general, achieved by making use of the analytic geometric terms in combination with fitting methodologies for the unknown radial matrix elements, and this is the approach explored herein. Again, more extensive background and discussion can be found in *Quantum Metrology Vol. 1* [4], and references therein. Note, however, that whilst the general approach taken here is sound, many outstanding questions remain regarding the information content required for such an approach to be successful, and if other limitations prevent a unique set of solutions to be found in a given case (e.g. symmetry restrictions, phase ambiguities) - such questions are explored further herein, particularly in the case studies presented [Part II](#).

The basic case also provides a strong foundation for extension into more complex light-matter interactions, in particular cases with shaped laser-fields (i.e. a time-dependent coupling $\hat{\Gamma}(\mathbf{E}, t)$) and multi-photon processes (which require multiple matrix elements, and/or different approximations). Note, however, that non-perturbative (strong field) light-matter interactions are, typically, not amenable to description in a separable picture in this manner. In such cases the laser field, molecular and continuum properties are strongly coupled, and are typically treated numerically in a fully time-dependent manner (although some separation of terms may work in some cases, depending on the system and interaction(s) at hand).

Underlying the photoelectron observables is the photoelectron continuum state $|\mathbf{k}\rangle$, prepared via photoionization. The photoelectron momentum vector is denoted generally by $\mathbf{k} = k\hat{\mathbf{k}}$, in the molecular frame ([MF](#)). The ionization matrix

elements associated with this transition provide the set of quantum amplitudes completely defining the final continuum scattering state,

$$|\Psi_f\rangle = \sum \int |\Psi_+; \mathbf{k}\rangle \langle \Psi_+; \mathbf{k}| \Psi_f \rangle d\mathbf{k}, \quad (6.1)$$

where the sum is over states of the molecular ion $|\Psi_+\rangle$. The number of ionic states accessed depends on the nature of the ionizing pulse and interaction. For the dipolar case,

$$\hat{\Gamma}(\mathbf{E}) = \hat{\mu} \cdot \mathbf{E} \quad (6.2)$$

Hence,

$$\langle \Psi_+; \mathbf{k} | \Psi_f \rangle = \langle \Psi_+; \mathbf{k} | \hat{\mu} \cdot \mathbf{E} | \Psi_i \rangle \quad (6.3)$$

Where the notation implies a perturbative photoionization event from an initial state i to a particular ion plus electron state following absorption of a photon $h\nu$, $|\Psi_i\rangle + h\nu \rightarrow |\Psi_+; \mathbf{k}\rangle$, and $\hat{\mu} \cdot \mathbf{E}$ is the usual dipole interaction term [82], which includes a sum over all electrons s defined in position space as \mathbf{r}_s :

$$\hat{\mu} = -e \sum_s \mathbf{r}_s \quad (6.4)$$

The position space photoelectron wavefunction is typically expressed as a *partial-wave expansion*, expanded as (asymptotic) continuum eigenstates of orbital angular momentum, with angular momentum components (l, m) ,

$$\Psi_{\mathbf{k}}(r) \equiv \langle r | \mathbf{k} \rangle = \sum_{lm} Y_{lm}(\hat{\mathbf{k}}) \psi_{lm}(r, k) \quad (6.5)$$

where r are MF electronic coordinates and $Y_{lm}(\hat{\mathbf{k}})$ are the spherical harmonics. Note the lower-case notation for the partial wave angular-momentum components, distinct from upper-case for the similar terms (L, M) in the observables (Sect. 6.6).

Similarly, the ionization dipole matrix elements can be separated generally into radial (energy-dependent or ‘dynamical’ terms) and geometric (angular momentum) parts (this separation is essentially the Wigner-Eckart Theorem, see Ref. [83] for general discussion), and written generally as (using notation similar to [84]):

$$\langle \Psi_+; \mathbf{k} | \hat{\mu} \cdot \mathbf{E} | \Psi_i \rangle = \sum_{lm} \gamma_{l,m} \mathbf{r}_{k,l,m} \quad (6.6)$$

Provided that the *geometric coupling parameters* (the geometric part of the matrix elements) $\gamma_{l,m}$ - which includes the geometric rotations into the *LF* arising from the dot product in Eq. (6.6) and other angular-momentum coupling terms - are known, knowledge of the so-called *radial matrix elements* elements, at a given k thus equates to a full description of the system dynamics (and, hence, the observables). Determination of these *radial matrix elements* - which are complex quantities with magnitudes and phases - is the aim of the reconstruction methodologies discussed herein (see Sect. 7).

For the simplest treatment, the radial matrix element can be approximated as a 1-electron integral involving the initial electronic state (orbital), and final continuum photoelectron wavefunction:

$$\mathbf{r}_{k,l,m} = \int \psi_{lm}^*(r, k) r \Psi_i(r) dr \quad (6.7)$$

As noted above, the geometric terms $\gamma_{l,m}$ are analytical functions which can be computed for a given case - minimally requiring knowledge of the molecular symmetry and polarization geometry, although other factors may also play a role (see Sect. 6.3.2 for details).

The photoelectron angular distribution (*PADs*) at a given (ϵ, t) can then be determined by the squared projection of $|\Psi_f\rangle$ onto a specific state $|\Psi_+; \mathbf{k}\rangle$; very generally this can be written in terms of the energy and angle-resolved observable, which arises as the coherent square:

$$I(\epsilon, \theta, \phi) = \langle \Psi_f | \Psi_+; \mathbf{k} \rangle \langle \Psi_+; \mathbf{k} | \Psi_f \rangle \quad (6.8)$$

Expansion in terms of the components of the matrix elements as detailed above then yields a separation into radial and angular components (see *Quantum Metrology* Vol. 1 [4], Sect. 2.1 for a full derivation), which can be written (at a single energy) as (following Eq. 2.45 of *Quantum Metrology* Vol. 1 [4]):

$$I(\theta, \phi; k) = \sum_{ll'} \sum_{\lambda\lambda'} \sum_{mm'} \gamma_{\alpha\alpha_+ l\lambda m l' \lambda' m'} r_{kl\lambda} r_{k'l'\lambda'} e^{i(\eta_{l\lambda}(k) - \eta_{l'\lambda'}(k))} Y_{lm}(\hat{k}) Y_{l'm'}^*(\hat{k}) \quad (6.9)$$

In this form α denotes all other quantum numbers required to define the initial state, and α_+ the final state of the molecular ion. The *radial matrix elements* $r_{kl\lambda}$, denote an integral over the radial part of the wavefunctions, in this case labelled by the *MF* quantum numbers, and the associated scattering phase is given by $\eta_{l\lambda}(k)$ (i.e. the matrix elements are written in magnitude-phase form, rather than complex form). The γ term denotes a general set of *geometric coupling parameters* arising from the coherent square. A tensor form is also given herein, see [Sect. 6.3.2](#), and includes a full breakdown of these terms and details of numerical implementations.

Comparison of Eq. (6.9) with Eq. (6.37) indicates that the amplitudes in Eq. (6.6) also determine the observable *anisotropy parameters* $\beta_{L,M}(\epsilon, t)$ (Eq. (6.37)), which basically collect all the terms in Eq. (6.9) and the product over spherical harmonics, into a resultant set of (L, M) parameters which describe the observable *PADs*. (Note that the photoelectron energy ϵ and momentum k are used somewhat interchangeably herein, with the former usually preferred in reference to observables.) Further discussion of the observables, including computation and form of these observables, can be found in [Sect. 6.6](#):

The radial matrix elements - hence observables - are a sensitive function of molecular geometry and electronic configuration in general, as they depend on the overlap of the initial and final state wavefunctions, which includes the ionization continuum (scattering wavefunction) of the photoelectron. Hence, they may be considered to be responsive to molecular dynamics as well as photoionization dynamics, although they are formally time-independent in a Born-Oppenheimer basis. For further general discussion and examples see Ref. [85] and *Quantum Metrology* Vol. 1 [4]; discussions of more complex cases with electronic and nuclear dynamics can be found in Refs. [81, 86, 87, 88].

Note, also, that in the treatment above there is no time-dependence incorporated in the notation; however, a time-dependent treatment readily follows, and may be incorporated either as explicit time-dependent modulations in the expansion of the wavefunctions for a given case, or implicitly in the radial matrix elements. Examples of the former include, e.g. a rotational or vibrational wavepacket, or a time-dependent laser field. The rotational wavepacket case is discussed herein (see [Sect. 6.3.2](#)).

Typically, for reconstruction experiments, a given measurement will be selected to simplify this as much as possible by, e.g., populating only a single ionic state (or states for which the corresponding observables are experimentally energetically-resolvable), and with a bandwidth $d\mathbf{k}$ which is small enough such that the *radial matrix elements* can be assumed constant over the observation window. Importantly, the angle-resolved observables are sensitive to the magnitudes and (relative) phases of these *radial matrix elements* - as emphasised in the magnitude-phase form of Eq. (6.9) - and can be considered as angular interferograms. It is the interferometric nature of the *PADs* which enables a phase-sensitive reconstruction protocol to be pursued.

6.2 Symmetry in photoionization

Symmetry in photoionization has briefly been suggested by the introduction of symmetrized harmonics ([Sect. 6.6.2](#)), and is discussed in detail in *Quantum Metrology* Vol. 1 [4] (Sect. 2.2.3.3). Herein a brief review is given, with a focus on using symmetry in matrix element retrieval problems. For further details, see *Quantum Metrology* Vol. 1 [4]; for a more general discussion of symmetry in molecular spectroscopy see the textbook by Bunker and Jensen [89], and the specific case of photoionization is expanded on in the work of Signorelli and Merkt [90].

In general, for the dipole matrix element to be non-zero the direct product of the initial state, final state and dipole operator symmetries must contain the totally symmetric representation of the molecular symmetry (*MS*) group, which is isomorphic to the point group (*PG*) in rigid molecules. This general case can be written as:

$$\Gamma_{rve}^f \otimes \Gamma_{dipole} \otimes \Gamma_{rve}^i \supset \Gamma^s \quad (6.10)$$

Where Γ_{rve} is the rovibronic symmetry of the system (i.e. total symmetry excluding spin), with the i/f superscript denoting initial and final states respectively. Γ^s is the totally symmetric representation in the appropriate molecular symmetry group, and Γ_{dipole} is the symmetry of the dipole operator.

For the specific case of photoionization the final state is split into the symmetry species of the ion and the photoelectron [90]:

$$\Gamma^e \otimes \Gamma_{rve}^+ \otimes \Gamma_{dipole} \otimes \Gamma_{rve}^i \supset \Gamma^s \quad (6.11)$$

This is, essentially, a statement of the limiting case of Eq. (6.3) (see also alternative forms of Eqs. (6.6), (6.7)), which defines the symmetry requirements for the overlap integral to be non-zero (although does not indicate that it will be non-zero for a given system).

In the reconstruction experiments discussed herein, this general form can be often be further simplified. In particular, assuming a full Born-Oppenheimer separation of dynamics, the problem can be treated within the static *PG* of the system, and only the electronic state symmetries need to be taken into account. In practice, this treatment is appropriate for cases with separable rotational wavepackets, and may also be a reasonable approximation for cases with vibronic wavepackets in cases where the nuclear excursions are relatively small and/or can be treated as linear combinations over a set of symmetrized basis functions. Within this approximation the general symmetry requirements can be written as:

$$\Gamma^{e(X)} \otimes \Gamma_e^+ \otimes \Gamma_{dipole} \otimes \Gamma_e^i \supset \Gamma^s \quad (6.12)$$

And $\Gamma^{e(X)}$ indicates that the continuum symmetries are expressed in a basis of symmetrized harmonics (Sect. 6.6.2). From Eq. (6.12), the set of allowed matrix elements for a given ionization event can be expressed, in terms of the allowed set of symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta, \phi)$, or (equivalently) the usual partial wave basis expressed in spherical harmonics $Y_{l,\lambda}(\theta, \phi)$, and a set of associated symmetrization coefficients $b_{hl\lambda}^{\Gamma\mu}$.

A brief numerical example is given below, and a more detailed treatment for a range of photoionization cases forms the second half of the book, see Chapter 8 for details.

```
# Example following symmetrized harmonics demo

# Import class
from pemtk.sym.symHarm import symHarm

# Compute hamronics for Td, lmax=4
sym = 'D2h'
lmax=4

symObj = symHarm(sym, lmax)

# Allowed terms and mappings are given in 'dipoleSyms'
symObj.dipole['dipoleSyms']
```

```
*** Mapping coeffs to ePSproc dataType = mate
Remapped dims: {'C': 'Cont', 'mu': 'it'}
Added dim Eke
Added dim Targ
Added dim Total
Added dim mu
Added dim Type
Found dipole symmetries:
{'B1u': {'m': [0], 'pol': ['z']}, 'B2u': {'m': [-1, 1], 'pol': ['y']}, 'B3u': {'m': [-1, 1], 'pol': ['x']}}
```

```
{'B1u': {'m': [0], 'pol': ['z']},
 'B2u': {'m': [-1, 1], 'pol': ['y']},
 'B3u': {'m': [-1, 1], 'pol': ['x']}}}
```

```
# Setting the symmetry for the neutral and ion allows direct products to be computed,
# and allowed terms to be determined.

sNeutral = 'A1g'
sIon = 'B2u'

symObj.directProductContinuum([sNeutral, sIon])

# Results are pushed to self.continuum, in dictionary and Pandas DataFrame formats,
# and can be manipulated using standard functionality.
# The subset of allowed values are also set to a separate DataFrame and list.
symObj.continuum['allowed']['PD']
```

```
<pandas.io.formats.style.Styler at 0x7f2a001d7760>
```

Dipole	Target	allowed	m	pol	result	terms
B1u	B3g	True	[0]	[z]	[A1g]	[A1g, B2u]
B2u	A1g	True	[-1, 1]	[y]	[A1g]	[A1g, B2u]
B3u	B1g	True	[-1, 1]	[x]	[A1g]	[A1g, B2u]

```
# Ylm basis table with the Character values limited to those defined
# in self.continuum['allowed']['PD'] Target column
symObj.displayYlm(symFilter = True)
```

Character (\$\Gamma\$)	SALC (h)	PFIX (\$\mu\$)	m	b			
				1	0	1	2
A1g	0	0	0	1.0			
	1	0	0		1.0		
	2	0	2			1.0	
	3	0	0				1.0
	4	0	2				1.0
	5	0	4				1.0
B1g	0	0	-2		1.0		
	1	0	-4			1.0	
	2	0	-2				1.0
B3g	0	0	-1		1.0		
	1	0	-3			1.0	
	2	0	-1				1.0

6.3 Tensor formulation of photoionization

A number of authors have treated *PADs* and related problems in the context of photoionization theory and matrix element reconstruction (see *Quantum Metrology* Vol. 2 [9] Chpt. 8 for examples and discussion, a range of review articles can also be found in the literature, e.g. Refs. [91, 92, 93, 94]); herein, a geometric tensor based formalism is developed, which is close in spirit to the treatments given by Underwood and co-workers [88, 95, 96], but further separates various sets of physical parameters into dedicated tensors; this allows for a unified theoretical and numerical treatment, where the latter computes properties as tensor variables which can be further manipulated and investigated to give detailed insights into various aspects of photoionization for the system at hand, and implications/effects for matrix element retrieval in a given case. Furthermore, the tensors can readily be converted to a density matrix representation [83, 97], which is more natural for some quantities, and also emphasizes the link to quantum state tomography and other quantum information techniques. Much of the theoretical background, as well as application to aspects of the current problem, can be found in the textbooks of Blum [97] and Zare [83].

Within this treatment, the observables can be defined in a series of simplified forms, emphasizing the quantities of interest for a given problem. The most general, and simplest, form is given in Sect. 6.3.1, in terms of *channel functions*, and the remainder of this section (Sect. 6.3.2 - Section 6.3.9) gives a detailed breakdown of the various components of the *channel functions*, and numerical examples.

6.3.1 Channel functions

A simple form of the equations¹, amenable to fitting and numerical implementation, is to write the observables in terms of *channel functions*, which define the ionization continuum for a given case and set of parameters u (e.g. defined for the *MF*, or defined for a specific experimental configuration),

$$\beta_{L,M}^u = \sum_{\zeta, \zeta'} \Upsilon_{L,M}^{u,\zeta\zeta'} \|\zeta\zeta' \quad (6.13)$$

Where ζ, ζ' collect all the required quantum numbers, and define all (coherent) pairs of components. The term $\|\zeta\zeta'$ denotes the coherent square of the ionization matrix elements:

$$\|\zeta\zeta' = I^\zeta(\epsilon) I^{\zeta'*}(\epsilon) \quad (6.14)$$

Eq. (6.13) is effectively a convolution equation (cf. Refs. [95, 98]) with channel functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$, for a given “experiment” u , summed over all terms ζ, ζ' . Aside from the change in notation (which is here chosen to match the formalism of Refs. [36, 37, 38]), these matrix elements are essentially identical to the simplified *radial matrix elements* $\mathbf{r}_{k,l,m}$ defined in Eq. (6.6), in the case where $\zeta = \{k, l, m\}$. Similarly, the *channel functions* are essentially nothing but a slightly different form of the *geometric coupling parameters* of Eqs. (6.6), (6.9), incorporating all required geometric parameters. Note, also, that the *radial matrix elements* used herein are usually assumed to be symmetrized (unless explicitly stated), i.e. expanded in *symmetrized harmonics* per Eq. (6.37), but with any additional symmetry parameters $b_{hl\lambda}^{\Gamma_\mu}$ incorporated into the value of the *radial matrix elements*.

These complex matrix elements can also be equivalently defined in a magnitude, phase form:

$$I^\zeta(\epsilon) \equiv \mathbf{r}_\zeta \equiv r_\zeta e^{i\phi_\zeta} \quad (6.15)$$

This tensorial form is numerically implemented in the *ePSproc* [34] codebase, and is in contradistinction to standard numerical routines in which the requisite terms are usually computed from vectorial and/or nested summations. The *Photoelectron Metrology Toolkit* [5] codebase implements *radial matrix elements* retrieval based on the tensor formalism, with pre-computation of all the geometric tensor components (*channel functions*) prior to a fitting protocol for matrix element analysis, essentially a fit to Eqn. (6.13), with terms $I^\zeta(\epsilon)$ as the unknowns (in magnitude, phase form per (6.15)). The main computational cost of a tensor-based approach is that more RAM is required to store the full set of tensor

¹ Cf. the general form of Eq. (6.9). See also *Quantum Metrology* Vol. 2 [9] Chpt. 12 for early discussion and motivation for this formalism.

variables; however, the method is computationally efficient since it is inherently parallel (as compared to a traditional, serial loop-based solution), hence may lead to significantly faster evaluation of observables. Furthermore, the method allows for the computational routines to match the formalism quite closely, and for the investigation of the properties of the *channel functions* for a given problem in general terms, as well as for specific experimental cases including examination of specific couplings/effects. (Again, this is in contrast to standard nested-loop routines, which can be somewhat opaque to detailed interpretation, and typically implement the full computation of the observables in one monolithic computational routine; they do, however, have significantly lower RAM requirements since the full multi-dimensional basis tensors are not required to be stored.) Sect. 6.3.2 provides details of the tensor components of the *channel functions*, and the remainder of this section breaks these down further, including numerical examples, and discussion of their significance for fitting problems in specific cases.

6.3.2 Full tensor expansion

In more detail, the *channel functions* $\Upsilon_{L,M}^{u,\zeta\zeta'}$ can be given as a set of tensors, defining each aspect of the problem. The following equations illustrate this for the *MF* and *LF/AF* cases, fully expanding the general form of Eq. (6.13) in terms of the relevant tensors. Further details and numerical examples are given in the following sub-sections.

For the *MF*:

$$\begin{aligned} \beta_{L,-M}^{\mu_i, \mu_f}(\epsilon) = & (-1)^M \sum_{P,R',R} [P]^{\frac{1}{2}} E_{P-R}(\hat{e}; \mu_0) \\ & \times \sum_{l,m,\mu} \sum_{l',m',\mu'} (-1)^{(\mu'-\mu_0)} \Lambda_{R',R}(R_{\hat{n}}; \mu, P, R, R') B_{L,-M}(l, l', m, m') \\ & \times I_{l,m,\mu}^{p_i \mu_i, p_f \mu_f}(\epsilon) I_{l',m',\mu'}^{p_i \mu_i, p_f \mu_f*}(\epsilon) \end{aligned} \quad (6.16)$$

And the *LF/AF* as:

$$\begin{aligned} \bar{\beta}_{L,-M}^{\mu_i, \mu_f}(\epsilon, t) = & (-1)^M \sum_{P,R',R} [P]^{\frac{1}{2}} E_{P-R}(\hat{e}; \mu_0) \\ & \times \sum_{l,m,\mu} \sum_{l',m',\mu'} (-1)^{(\mu'-\mu_0)} \bar{\Lambda}_{R'}(\mu, P, R') B_{L,S-R'}(l, l', m, m') \\ & \times I_{l,m,\mu}^{p_i \mu_i, p_f \mu_f}(\epsilon) I_{l',m',\mu'}^{p_i \mu_i, p_f \mu_f*}(\epsilon) \sum_{K,Q,S} \Delta_{L,M}(K, Q, S) A_{Q,S}^K(t) \end{aligned} \quad (6.17)$$

In both cases a set of geometric tensor terms are required,

these terms provide details of:

- $E_{P-R}(\hat{e}; \mu_0)$: polarization geometry & coupling with the electric field.
- $B_{L,M}(l, l', m, m')$: geometric coupling of the partial waves into the $\beta_{L,M}$ terms (spherical tensors). Note for the *AF* case the terms may be reindexed by $M = S - R'$, which allows for the projection dependence on the *ADMs* (see below).
- $\Lambda_{R',R}(R_{\hat{n}}; \mu, P, R, R')$, $\bar{\Lambda}_{R'}(\mu, P, R')$: frame couplings and rotations (note slightly different terms for *MF* and *AF*).
- $\Delta_{L,M}(K, Q, S)$: alignment frame coupling (*LF/AF* only).
- $A_{Q,S}^K(t)$: ensemble alignment described as a set of *axis distribution moments* (*ADMs*, *LF/AF* only). Note for a one-photon ionization case - the traditional *LF* experiment - there will only be a single term, $K = Q = S = 0$, with no time-dependence, which describes an isotropic molecular ensemble. In general only the *AF* is discussed explicitly herein, but it is of note that this is identical to the traditional *LF* definition for this limiting case of an isotropic ensemble.
- Square-brackets are short-hand for degeneracy terms, e.g. $[P]^{\frac{1}{2}} = (2P + 1)^{\frac{1}{2}}$.

And $I_{l,m,\mu}^{p_i\mu_i,p_f\mu_f}(\epsilon)$ are the *radial matrix elements*, as a function of energy ϵ . As noted above, these *radial matrix elements* are essentially identical to the simplified forms $r_{k,l,m}$ defined in Eqn. (6.6), except now with additional indices to label symmetry and polarization components defined by a set of *partial-waves* $\{l, m\}$, for polarization component μ (denoting the photon angular momentum components) and channels (symmetries) labelled by initial and final state indexes $(p_i\mu_i, p_f\mu_f)$. The notation here follows that used by ePolyScat (ePS) [36, 37, 38, 39], and these matrix elements again represent the quantities to be obtained numerically from data analysis, or from an ePolyScat (or similar) calculation.

Following the tensor components detailed above, the full form of the *channel functions* of Eq. (6.13) for the *AF* and *MF* can be written as:

$$\begin{aligned} \mathcal{T}_{L,M}^{u,\zeta\zeta'} = & (-1)^M [P]^{\frac{1}{2}} E_{P-R}(\hat{e}; \mu_0) (-1)^{(\mu' - \mu_0)} \Lambda_{R',R}(R_{\hat{n}}; \mu, P, R, R') \\ & \times B_{L,-M}(l, l', m, m') \end{aligned} \quad (6.18)$$

$$\begin{aligned} \bar{\mathcal{T}}_{L,M}^{u,\zeta\zeta'} = & (-1)^M [P]^{\frac{1}{2}} E_{P-R}(\hat{e}; \mu_0) (-1)^{(\mu' - \mu_0)} \bar{\Lambda}_{R'}(\mu, P, R') \\ & \times B_{L,S-R'}(l, l', m, m') \Delta_{L,M}(K, Q, S) A_{Q,S}^K(t) \end{aligned} \quad (6.19)$$

Note that, in this case as given, time-dependence arises purely from the $A_{Q,S}^K(t)$ terms in the AF case, and the electric field term currently describes only the photon angular momentum coupling, although can in principle also describe time-dependent/shaped fields. Similarly, a time-dependent initial state (e.g. a vibrational wavepacket) could also describe a time-dependent MF case.

It should be emphasized, however, that the underlying physical quantities are essentially identical in all the theoretical approaches, with a set of coupled angular-momenta defining the *geometric coupling parameters* part of the photoionization problem, despite these differences in the details of the theory and notation.

The various tensors defined above are implemented as functions in the ePSproc codebase [33, 34, 35], and further wrapped for fitting cases in the *Photoelectron Metrology Toolkit* [5]. In the remainder of this section, numerical examples using these codes are illustrated and explored. Full computational details can be found in the ePSproc documentation [35], including extended discussion of each tensor and complete function references in the geomCalc submodule documentation.

6.3.3 Frame definitions

A conceptual overview of the *LF/AF* and relation to the *MF*, in the context of the bootstrap reconstruction protocol, can be found in Fig. 4.1. A more detailed definition is given in Fig. 6.1, as pertains to the use of angular momentum notation and projections. The figure shows the general use of angular momenta and associated projection terms in molecular spectroscopy as an aid to visualising the discussion in the following sections. Note, however, that some alternative notations are used in this volume, in particular specific projection terms may be used for certain physical quantities.

In simple cases, the frame definition for the *AF* is identical to that of the *LF*, since it is usually defined by the laser polarization, with the distinction that an aligned molecular ensemble is additionally present. For the limiting case of an isotropic distribution, the *AF* and (traditional) *LF* are identical. However, in cases with non-linear laser polarization, and/or multiple pulses with different polarization vectors, the situation may be more complicated, and additional *frame rotation(s)* may be required (see Fig. 6.1 inset). In such cases the reference frame may be chosen as the final ionizing laser pulse polarization, or as a symmetry axis in the *AF*. For high degrees of (3D) alignment the AF may approach the *MF* in the ideal case, although will usually be limited by the symmetry of the system.

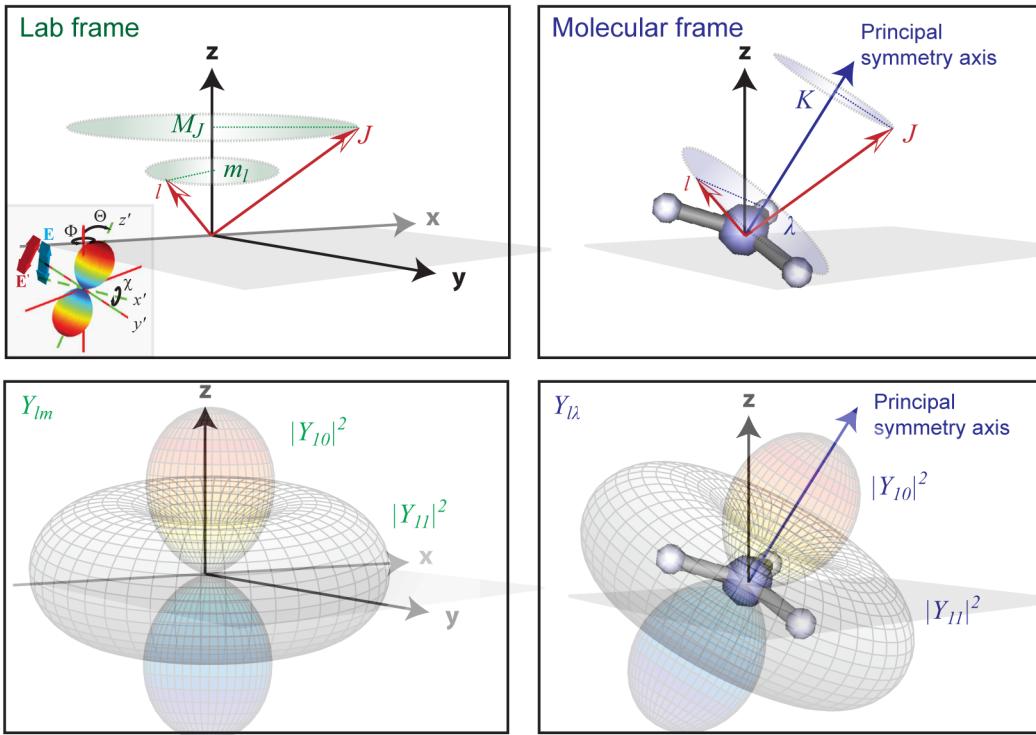


Fig. 6.1: Reference frame and angular momentum definitions for the Laboratory frame (*LF*) and Molecular frame (*MF*), using a general notation from molecular spectroscopy. In this case the *LF* shows an angular momentum vectors J and l ; J is usually used to define rotational (or sometimes total) angular momentum of the system, and l the electronic component. Projection terms onto the *LF* z -axis, M_J and m_l are also indicated. In the *MF* equivalent angular momentum terms are shown, with projections K and λ onto the molecular symmetry axis. The insert shows a *frame rotation* $(x, y, z) \leftarrow (x', y', z')$, defined by a set of *Euler angles* $R_{\hat{n}} = \{\chi, \Theta, \Phi\}$, and illustrating the rotation of the z -axis (defined by the electric field vector E), and a spherical harmonic function in the (x', y', z') frame. See also Fig. 4.1. Figure reproduced from *Quantum Metrology* Vol. 1 [4], Fig. 2.3, and shows - note that some alternative notations are used in this volume.

6.3.4 Numerical aside: symmetry-defined channel functions

In the following sub-sections, each component is defined in detail, including numerical examples. For illustration purposes, the numerical example uses a minimal set of assumptions, and is defined initially purely by symmetry, although further terms may be required for computation of some of the geometric terms and are discussed where required. A fuller discussion of symmetry considerations in photoionization can be found in Sect. 6.2, and discussion of *symmetrized harmonics* in Sect. 6.6.2.

For this example, the D_{2h} point group is used, representing a fairly general case of a planar asymmetric top system, e.g. ethylene (C_2H_4). Note that, in this case, the symmetrization coefficients ($b_{hl\lambda}^{\Gamma\mu}$, see (6.37)) have the property that $\mu = 0$ only, and the h index is redundant, since it maps uniquely to l - see Fig. 6.2 - so these indexes can be dropped. Note, also, the unfortunate convention that the label μ is used for multiple indexes; to avoid ambiguity this term is remapped to μ_X in the numerics below. However, in this case, since μ can be dropped from the symmetrization coefficients, there is actually no ambiguity in later usage.

Note: Full tabulations of the parameters available in HTML or notebook formats only.

```
*** Setup symmetry-defined matrix elements using PEMtk

# Import class
from pemtk.sym.symHarm import symHarm

*** Compute hamronics for Td, lmax=4
sym = 'D2h'
lmax=4

lmaxPlot = 2 # Set lmaxPlot for subselection on plots later.
```

```
# Create symHarm object with given settings,
# this will also compute the symmetrized harmonics
symObj = symHarm(sym, lmax)
```

```
# Display results (real harmonics)
symObj.displayXlm(setCols='h') #, dropLevels='mu')

# Glue version for JupyterBook output
# As above, but with PD object return and glue.
glue("D2hXlm", symObj.displayXlm(setCols='h', returnPD=True), display=False)
```

To compute basis tensors from these symmetry coefficients, they can be converted to the standard *radial matrix elements* format used in the *Photoelectron Metrology Toolkit* [5] (see Sect. 9.3.4 for further discussion of the numerical implementation), and used with the standard routines. These are demonstrated below, for two flavours - the base *ePSproc* [34] routine for computation of *AF-PADs*, and the *Photoelectron Metrology Toolkit* [5] fitting routine which wraps this functionality. Note that all tensors are stored as *Xarray* [53, 54] objects, which allow for easy numerical manipulation, subselection etc.

```
*** Compute basis functions for given matrix elements using PEMtk fit class
# This illustration uses the symmetrized matrix elements set above

*** To use ePSproc/PEMtk classes,
```

(continues on next page)

Character (\$\Gamma\$)	PFIX (\$\mu\$)	l	h	b					
				0	1	2	3	4	5
A1g	0	0	0	1.0					
		2	0		1.0				
			2			1.0			
		4	0				1.0		
			2					1.0	
A1u	0		4						1.0
		3	-2	1.0					
B1g	0	2	-2	1.0					
		4	-4		1.0				
			-2			1.0			
B1u	0	1	0	1.0					
		3	0		1.0				
			2			1.0			
B2g	0	2	1	1.0					
		4	1		1.0				
			3			1.0			
B2u	0	1	-1	1.0					
		3	-3		1.0				
			-1			1.0			
B3g	0	2	-1	1.0					
		4	-3		1.0				
			-1			1.0			
B3u	0	1	1	1.0					
		3	1		1.0				
			3			1.0			

Fig. 6.2: Symmetrized harmonics coefficients ($b_{hl}^{\Gamma\mu}$, see (6.37)) for D2h symmetry ($l_{max}=4$) generated with the **Photo-electron Metrology Toolkit** [5] wrapper for `libmsym` [64, 65]. Note that, in this case, the coefficients have the property that $\mu = 0$ only, and the h index is redundant (maps uniquely to l).

(continued from previous page)

```

# these values can be converted to ePSproc BLM data type...
# Run conversion - the default is to set the coeffs to the 'BLM' data type,
# additional dim mappings can also be set.
# Outputs are set to symObj.coeffs[dataType]
dimMap = {'C':'Cont','mu':'muX'}
symObj.toePSproc(dimMap=dimMap)

# Run conversion with a different dimMap & dataType
# Outputs are set to symObj.coeffs[dataType]
dataType = 'matE'
symObj.toePSproc(dimMap = dimMap, dataType=dataType)

**** Setup class object
data = pemtkFit()

# Set to new key in data class
dataKey = sym
data.data[dataKey] = {}

# Set data.data[dataKey][dataType] from cases set above
# This pushes the symmetrized coeffs computed above to the PEMtk fit class
# object for general use with PEMtk methods.
# Here set 'matE' for use as matrix elements, and 'BLM' for pad plotting routines.
for dataType in ['matE','BLM']:
    # Select expansion in complex harmonics, and sum redundant dims
    data.data[dataKey][dataType] = symObj.coeffs[dataType]['b (comp)'].sum(['h','muX'])
    # Propagate attrs
    data.data[dataKey][dataType].attrs = symObj.coeffs[dataType].attrs

# Set data by key
# data.subKey is the default location used by the PEMtk routines
data.subKey = dataKey

**** Compute basis function - two flavours
# Using PEMtk `afblmMatEfit` method
# - this only returns the product basis set as used for fitting
# See the docs for more details, https://pemtk.readthedocs.io
phaseConvention='S' # For consistency in the method, explicitly set the
                    # phase convention used here ('S' = standard, 'E' = ePS).
BetaNormX, basisProduct = data.afblmMatEfit(selDims={},
                                             sqThres=False,
                                             phaseConvention=phaseConvention)

# Using ePSproc directly - this includes full basis return if specified
# See the docs for more details, https://epsproc.readthedocs.io
BetaNormX2, basisFull = ep.geomFunc.afblmXprod(data.data[data.subKey]['matE'],
                                                 basisReturn = 'Full',
                                                 thres=None, selDims={},
                                                 sqThres=False,
                                                 phaseConvention=phaseConvention)

# The basis dictionary contains various numerical parameters, these are investigated
# below.

```

(continues on next page)

(continued from previous page)

```
# See also the ePSproc docs at
# https://epsproc.readthedocs.io/en/latest/methods/geometric_method_dev_260220_090420_
# tidy.html
print(f"Product basis elements: {basisProduct.keys()}")
print(f"Full basis elements: {basisFull.keys()}")

# Use full basis for following sections
basis = basisFull
```

```
*** Mapping coeffs to ePSproc dataType = BLM
Remapped dims: {'C': 'Cont', 'mu': 'muX'}
Added dim Eke
Added dim P
Added dim T
Added dim C
*** Mapping coeffs to ePSproc dataType = mate
Remapped dims: {'C': 'Cont', 'mu': 'muX'}
Added dim Eke
Added dim Targ
Added dim Total
Added dim mu
Added dim it
Added dim Type
Product basis elements: dict_keys(['BLMtableResort', 'polProd', 'phaseConvention',
    'BLMRenorm'])
Full basis elements: dict_keys(['ONs', 'EPRX', 'lambdaTerm', 'BLMtable',
    'BLMtableResort', 'AFterm', 'AKQS', 'polProd', 'phaseConvention', 'BLMRenorm',
    'matEmult'])
```

6.3.5 Matrix element geometric coupling term $B_{L,M}$

The coupling of the *partial-waves* as coherent pairs, $|l, m\rangle$ and $|l', m'\rangle$, into the observable set of $\{L, M\}$ is defined by a tensor contraction with two 3j terms:

$$B_{L,M} = (-1)^m \left(\frac{(2l+1)(2l'+1)(2L+1)}{4\pi} \right)^{1/2} \begin{pmatrix} l & l' & L \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} l & l' & L \\ -m & m' & M \end{pmatrix} \quad (6.20)$$

Note that this term is equivalent, effectively, to a triple integral over spherical harmonics (e.g. Eq. 3.119 in Zare [83]):

$$\int_0^{2\pi} \int_0^\pi Y_{J_3 M_3}(\theta, \phi) Y_{J_2 M_2}(\theta, \phi) Y_{J_1 M_1}(\theta, \phi) \sin \theta d\theta d\phi = \left(\frac{(2J_1+1)(2J_2+1)(2J_3+1)}{4\pi} \right)^{1/2} \times \begin{pmatrix} J_1 & J_2 & J_3 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} J_1 & J_2 & J_3 \\ M_1 & M_2 & M_3 \end{pmatrix}$$

And a similar term appears in the contraction over a pair of harmonics into a resultant harmonic (e.g. Eqs. C.21, C.22 in Blum [97]) - this is how the term arises in the derivation of the observables.

$$Y_{J_1 M_1}(\theta, \phi) Y_{J_2 M_2}(\theta, \phi) = \sum_{J_3 M_3} \left(\frac{(2J_1+1)(2J_2+1)(2J_3+1)}{4\pi} \right)^{1/2} \times \begin{pmatrix} J_1 & J_2 & J_3 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} J_1 & J_2 & J_3 \\ M_1 & M_2 & M_3 \end{pmatrix} Y_{J_3 M_3}^*(\theta, \phi)$$

Note also some definitions use conjugate spherical harmonics, which can be converted as, e.g., Eq. C.21 in Blum [97]:

$$\beta_{L,M}^{\mu_i,\mu_f} Y_{LM}^*(\theta_{\hat{k}}, \phi_{\hat{k}}) = \beta_{L,-M}^{\mu_i,\mu_f} (-1)^M Y_{L,-M}(\theta_{\hat{k}}, \phi_{\hat{k}}) \quad (6.21)$$

In the current Photoelectron Metrology Toolkit [5] codebase, the relevant basis item can be inspected as below, in order to illustrate the sensitivity of different (L, M) terms to the matrix element products. In many typical cases, however, this term is restricted to only $M = 0$ components overall by other geometric factors (see below).

The code cells below illustrate this for the current example case, and Fig. 6.3 offers a general summary. In general, this is a convenient way to visualize the selection rules into the observable: for instance, only terms $l = l'$ and $m = -m'$ contribute to the overall photoionization cross-section term ($L = 0, M = 0$), and the maximum observable $L_{max} = 2l_{max}$. However, since these terms are fairly simply followed algebraically in this case, via the rules inherent in the $3j$ product (Eq. (6.20)), this is not particularly insightful (although useful pedagogically). These visualizations will become more useful when dealing with real sets of matrix elements, and specific polarization geometries, which will further modulate or restrict the $B_{L,M}$ terms.

Numerically, various standard functions may be used to quickly gain deeper insight, for example min/max, averages etc. Such considerations may provide a quick sanity-check for a given case, and may prove useful when planning experiments to investigate particular aspects or channels of a given system. Other properties of the basis functions may also be interrogated numerically; for instance, correlation maps provide an alternative way to check which terms are strongly correlated or coupled, or will dominate a given aspect of the observable.

```
#*** Tabulate basis

basisKey = 'BLMtableResort' # Key for BLM basis set

# Reformat basis for display (optional)
stackDims = {'LM': ['L', 'M']}
basisPlot = basis[basisKey].rename({'S-Rp': 'M'}).stack(stackDims)

# Convert to Pandas, use ep.multiDimXrToPD as a general multi-dimensional restacker
pd, _ = ep.multiDimXrToPD(basisPlot, colDims=stackDims)

# Summarise properties and tabulate via Pandas Describe
pd.describe().T
```

```
#*** Plot BLM terms for basis set - basic case
basisKey = 'BLMtableResort'

# Basic plot
ep.lmPlot(basisPlot, xDim=stackDims); # Basic plot with all terms
```

```
#*** Plot BLM terms for basis set - plot with some additional figure formatting
#options

# Formatting options
titleString=f'$B_{\{L,M\}}$ terms for {sym}, lmax={lmaxPlot}'
titleDetails=True
labelRound = 1
catLegend=False
labelCols = [1,1]

# cmap = None for default.
# cmap = 'vlag'

# lmPlot with various options
```

(continues on next page)

(continued from previous page)

```
*_, gFig = ep.lmPlot(basisPlot.where((basisPlot.l<=lmaxPlot)
    & (basisPlot.lp<=lmaxPlot)),
    xDim=stackDims, pType = 'r',
    cmap=cmap, labelRound = labelRound,
    catLegend=catLegend,
    titleString=titleString, titleDetails=titleDetails,
    labelCols = labelCols);

# Glue figure
glue("lmPlot_BLM_basis_D2h", gFig.fig, display=False)
```

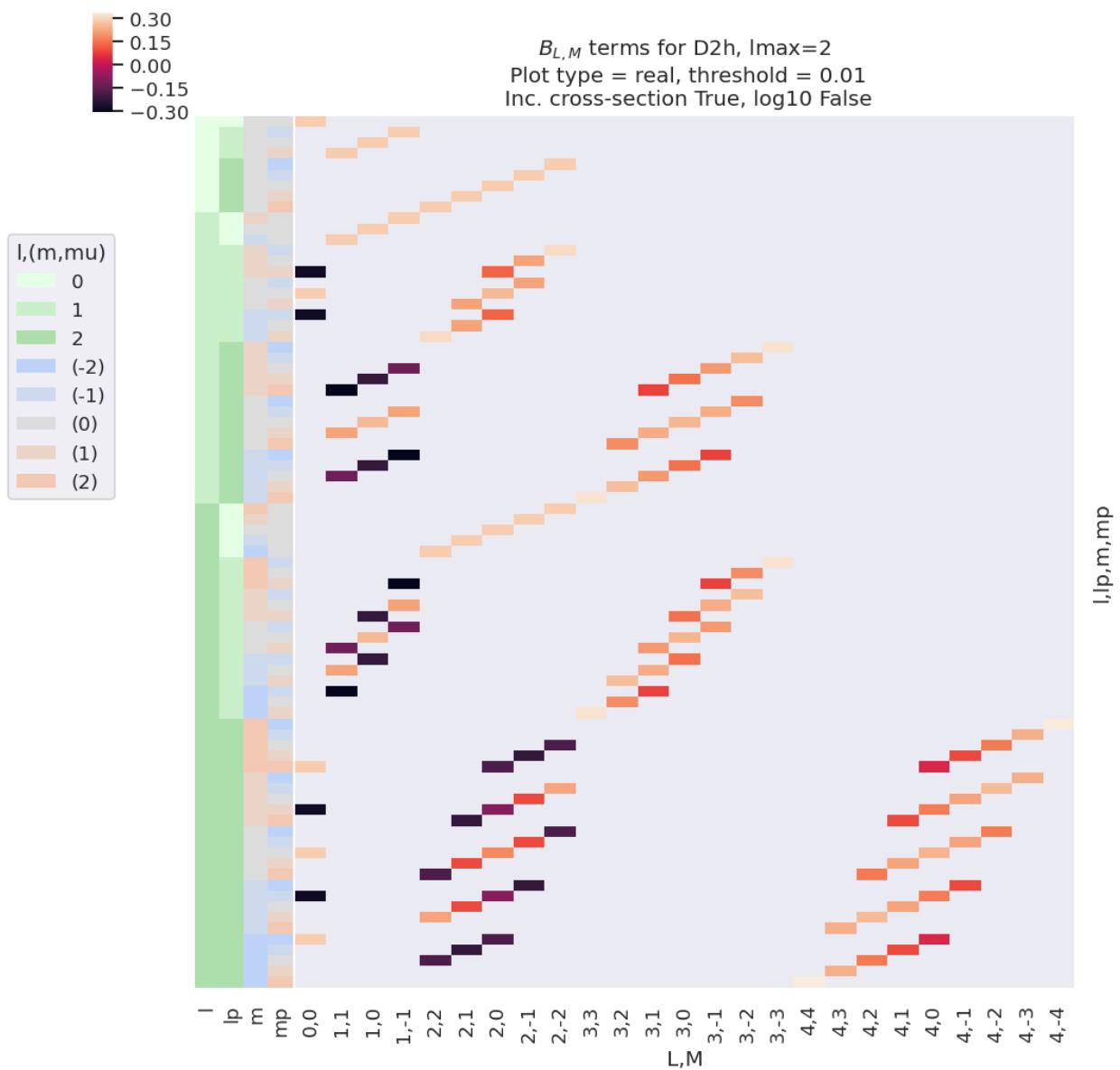


Fig. 6.3: Example $B_{L,M}$ basis functions for D_{2h} symmetry. Note figure is truncated to $l_{max} = l'_{max} = 2$ for clarity.

6.3.6 Electric field geometric coupling term $E_{P,R}(\hat{e}; \mu_0)$

The coupling of two 1-photon terms (which arises in the square of the ionization matrix element as per Eq. (6.9)) can be written as a tensor contraction:

$$E_{P,R}(\hat{e}) = [e \otimes e^*]_R^P = [P]^{\frac{1}{2}} \sum_p (-1)^R \begin{pmatrix} 1 & 1 & P \\ p & R-p & -R \end{pmatrix} e_p e_{R-p}^* \quad (6.22)$$

Where:

- e_p and e_{R-p} define the field strengths for the polarizations p and $R-p$, which are coupled into the spherical tensor E_{PR} ;
- square-brackets indicate degeneracy terms, e.g. $[P]^{\frac{1}{2}} = (2P+1)^{\frac{1}{2}}$;
- in the literature **LF/AF** field projection terms are usually denoted p (as above) or μ_0 (used as a more general angular-momentum projection notation). For the **MF** case photon projection terms are usually denoted by q or μ .
- The polarization vector or propagation direction is usually chosen to define the **LF/AF** z-axis for linear or non-linearly polarized light respectively (see Fig. 4.1 for the linear example), or defined relative to the molecular symmetry axis in the **MF**. (See Sect. 6.3.3 for further details.)
- For a given case the polarization geometry may define a single projection term, or there may be multiple terms allowed. For instance, linearly polarized light is defined by $\mu_0 = 0$ only, resulting in non-zero values for just the $P = 0, 2$ terms in Eq. (6.22), i.e. product terms $E_{0,0}, E_{2,0}$ are allowed. Non-linearly polarized light may contain all allowed components ($\mu_0 = -1, 0, 1$). For the **MF** case, allowed components may be defined directly in the **MF** or determined via a frame transformation from the **LF** - see Sect. 6.3.7.
- Note this notation implicitly describes only the time-independent photon angular momentum coupling, but time-dependent/shaped laser fields can be readily incorporated by allowing for time-dependent fields $e_p(t)$ (see, for instance, Ref. \cite{hockett2015CompletePhotoionizationExperiments}). (Support for this is planned in ePSproc [34], as of v1.3.2 this is in the development and testing phase.)

To derive this result, one can start from a general spherical tensor direct product, e.g., Eq. 5.36 in Zare [83]; for two first-rank tensors (e.g. electric field vectors) this contraction is given explicitly by Eq. 5.40 in Zare [83]:

$$[A^{(1)} \otimes B^{(1)}]_q^k = \sum_m \langle 1m, 1q-m | kq \rangle A(1, m) B(1, q-m) \quad (6.23)$$

Convert to $3j$ form:

$$[A^{(1)} \otimes B^{(1)}]_q^k = \sum_m (-1)^q [k]^{1/2} \begin{pmatrix} 1 & 1 & k \\ m & q-m & -q \end{pmatrix} A(1, m) B(1, q-m) \quad (6.24)$$

And substitute in appropriate terms.

As before, we can visualise these values with the Photoelectron Metrology Toolkit [5].

```
**** For illustration, recompute EPR term for default case.
EPRX = ep.geomCalc.EPR(form = 'xarray')

# Set parameters to restack the Xarray into (L,M) pairs
plotDimsRed = ['p', 'R-p']
xDim = {'PR': ['P', 'R']}

# Plot with ep.lmPlot(), real values
_, gFig = ep.lmPlot(EPRX, plotDims=plotDimsRed, xDim=xDim,
                     pType = 'r',
                     titleString=f'E_{\{P,R\}} terms (all cases).')
```

(continues on next page)

(continued from previous page)

```

labelCols = labelCols)

# Alternative version summed over l,l',m
# *_, gFig = ep.lmPlot(EPRX.unstack().sum(['l','lp','R-p']), xDim=xDim, pType = 'x')

# Glue figure
glue("lmPlot_EPR_basis", gFig.fig)

```

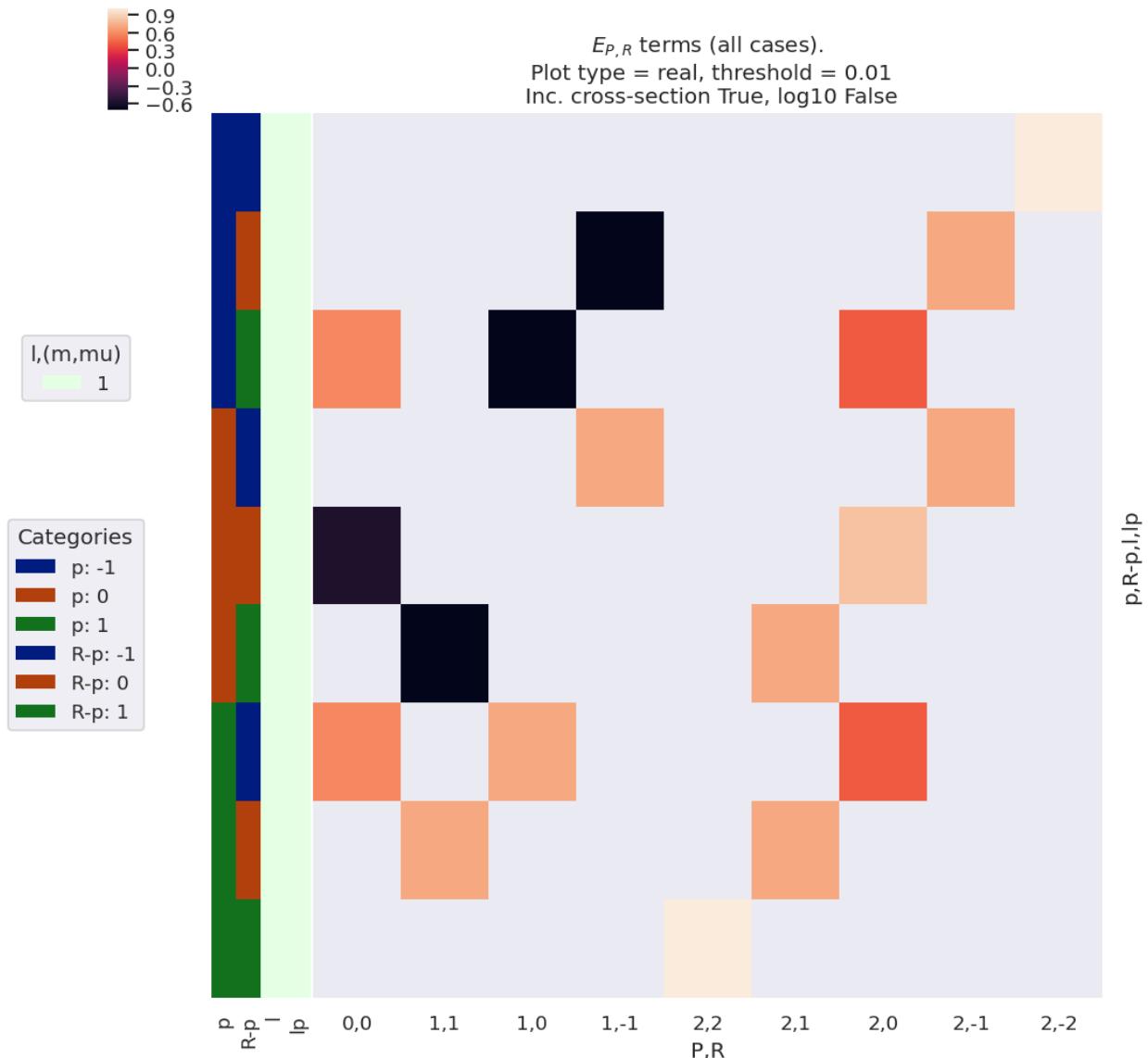


Fig. 6.4: Example $E_{P,R}$ basis functions. Note that for linearly polarised light $p = R - p = 0$ only, hence only the terms $E_{0,0}$ and $E_{2,0}$ are non-zero in this case. For non-linearly polarised cases many other terms are allowed.

6.3.7 Molecular frame projection term Λ

For the molecular frame case, the coupling between the LF and MF can be defined by a projection term, $\Lambda_{R',R}(R_{\hat{n}})$:

$$\Lambda_{R',R}(R_{\hat{n}}) = (-1)^{(R')} \begin{pmatrix} 1 & 1 & P \\ \mu & -\mu' & R' \end{pmatrix} D_{-R',-R}^P(R_{\hat{n}}) \quad (6.25)$$

This is similar to the $E_{P,R}$ term, and essentially rotates it into the MF , defining the projections of the polarization vector (photon angular momentum) μ into the MF for a given molecular orientation (*frame rotation*) defined by a set of rotations. The *frame rotation* is parameterized by a set of *Euler angles*, $R_{\hat{n}} = \{\chi, \Theta, \Phi\}$, with projections given by *Wigner rotation matrix elements* $D_{-R',-R}^P(R_{\hat{n}})$.

For the LF/AF case, the same term appears but in a simplified form:

$$\bar{\Lambda}_{R'} = (-1)^{(R')} \begin{pmatrix} 1 & 1 & P \\ \mu & -\mu' & R' \end{pmatrix} \equiv \Lambda_{R',R'}(R_{\hat{n}} = 0) \quad (6.26)$$

This form pertains since - in the LF/AF case - there is no specific frame transformation defined (i.e. there is no single molecular orientation defined in relation to the light polarization, rather a distribution as defined by the *ADMs*), but the total angular momentum coupling of the photon terms is still required in the equations.

Numerically, the function is calculated for a specified set of orientations, which default to the standard set of (x, y, z) MF polarization cases in the *Photoelectron Metrology Toolkit* [5] routines. For the LF/AF case, this term is still used, but restricted to $R_{\hat{n}} = (0, 0, 0) = z$, i.e. no frame rotation relative to the LF $E_{P,R}$ definition. In some cases additional frame transformations may be required here to define, e.g., the use of the propagation axis as the reference z -axis for circularly or elliptically polarized light. (See Sect. 6.3.3 for further discussion.)

6.3.8 Alignment tensor $\Delta_{L,M}(K, Q, S)$

Finally, for the LF/AF case, the alignment tensor couples the molecular axis ensemble (defined as a set of *ADMs*, see Sect. 6.5 for details) and the photoionization multipole terms into the final observable.

$$\Delta_{L,M}(K, Q, S) = (2K + 1)^{1/2}(-1)^{K+Q} \begin{pmatrix} P & K & L \\ R & -Q & -M \end{pmatrix} \begin{pmatrix} P & K & L \\ R' & -S & S - R' \end{pmatrix} \quad (6.27)$$

In the full equations for the observable, this term appears in a summation with the *ADMs*, as:

$$\tilde{\Delta}_{L,M}(t) = \sum_{K,Q,S} \Delta_{L,M}(K, Q, S) A_{Q,S}^K(t) \quad (6.28)$$

This summed alignment term can be considered, essentially, as a (coherent) geometric averaging of the MF observable weighted by the axis distribution in the AF (for more on the axis averaging as a convolution, see Refs. [4, 96]); equivalently, the averaging can be considered as a purely angular-momentum coupling effect, which accounts for all contributing moments of the various aspects of the system, and defines the allowed projections onto the final observables in the LF .

Mappings of these terms are investigated numerically below, for some exemplar cases.

Basic cases

Fig. 6.5 illustrates the alignment tensor $\Delta_{L,M}(K, Q, S)$ for some basic cases, and values are also tabulated in Table 6.6. Note that for illustration purposes the term is subselected with $K = 0, Q = 0, S = 0$ and $R' = 0$; $R \neq 0$ terms are included to illustrate the elliptically-polarized case, which can give rise to non-zero M terms.

For the simplest case of an unaligned ensemble, this term is restricted to $K = Q = S = 0$, i.e. $\Delta_{L,M}(0, 0, 0)$; for single-photon ionization with linearly-polarized light ($p = 0$, hence $P = 0, 2$ and $R = R' = 0$), this has non-zero values for $L = 0, 2$ and $M = 0$ only. Typically, this simplest case is synonymous with standard LF results, and maintains cylindrical and up-down symmetry in the observable.

For circularly polarized light ($p = \pm 1$, hence $P = 0, 1, 2$ and $R = R' = 0$), odd- L is allowed, signifying up/down symmetry breaking in the observable (where up/down pertains to the propagation direction of the light, conventionally the z -axis for non-linear polarizations, see Sect. 6.3.3). For elliptically polarized light, mixing of terms with different p allows for non-zero R terms (see Eq. (6.22)), hence non-zero M is allowed (see Eq. (6.27)), signifying breaking of cylindrical symmetry in the observable is allowed. Note, however, that non-zero values here do not indicate that such effects *will* be observed in any given case, only that they may be (or, at least, are not restricted by the alignment tensor).

```


#*** Set range of ADMs for test as time-dependent values (linear ramps)

# Set ADMs for increasing alignment...
# Note that delta term is independent of the absolute values of the ADMs(t),
# but does use this term to define limits on some quantum numbers.

tPoints = 10 # 10 t-points
inputADMs = [[0,0,0, *np.ones(tPoints)],
              [2,0,0, *np.linspace(0,1,tPoints)],
              [4,0,0, *np.linspace(0,0.5,tPoints)],
              [6,0,0, *np.linspace(0,0.3,tPoints)],
              [8,0,0, *np.linspace(0,0.2,tPoints)]]


# Set to AKQS parameters in Xarray
AKQS = ep.setADMs(ADMs = inputADMs)

# Use default EPR term - note this computes for all pol states, p=[-1,0,1]
EPR = ep.geomCalc.EPR(form='xarray')

#*** Compute alignment terms
AFterm, DeltaTerm = ep.geomCalc.deltaLMKQS(EPR, AKQS)

#*** Plot Delta term with subselections and formatting
xDim = {'LM': ['L', 'M']}
daPlot, daPlotpd, legendList, gFig = ep.lmPlot(
    DeltaTerm.sel(K=0, Q=0, S=0, Rp=0).sel({'S-Rp': 0}),
    xDim,
    pType = 'r', squeeze = False, thres=None,
    titleString="$\Delta(0,0,0)$ term ($R'=0$ only)",
    # Set dim mapping to use P,R with "l,m" colourmap
    dimMaps={'1Dims':['P'], 'mDims':'R'},
    labelCols=labelCols,
    catLegend=False)

# Glue versions for JupyterBook output
glue("deltaTerm000-lmPlot", gFig.fig, display=False)
# As above, but with PD object return and glue.
glue("deltaTerm000-tab", daPlotpd.fillna(''), display=False)


```

```


#*** Plot ADMs
_, ADMFig = ep.lmPlot(AKQS, xDim = 't', pType = 'r',
                      labelCols=labelCols, catLegend=False,
                      cmap='vlag',
                      titleString='ADMs (linear ramp example)'

#*** Plot AF term with subselection
_, AFFig = ep.lmPlot(AFterm.sel(R=0).sel(Rp=0).sel({'S-Rp': 0}),
                     xDim = 't', pType = 'r',


```

(continues on next page)

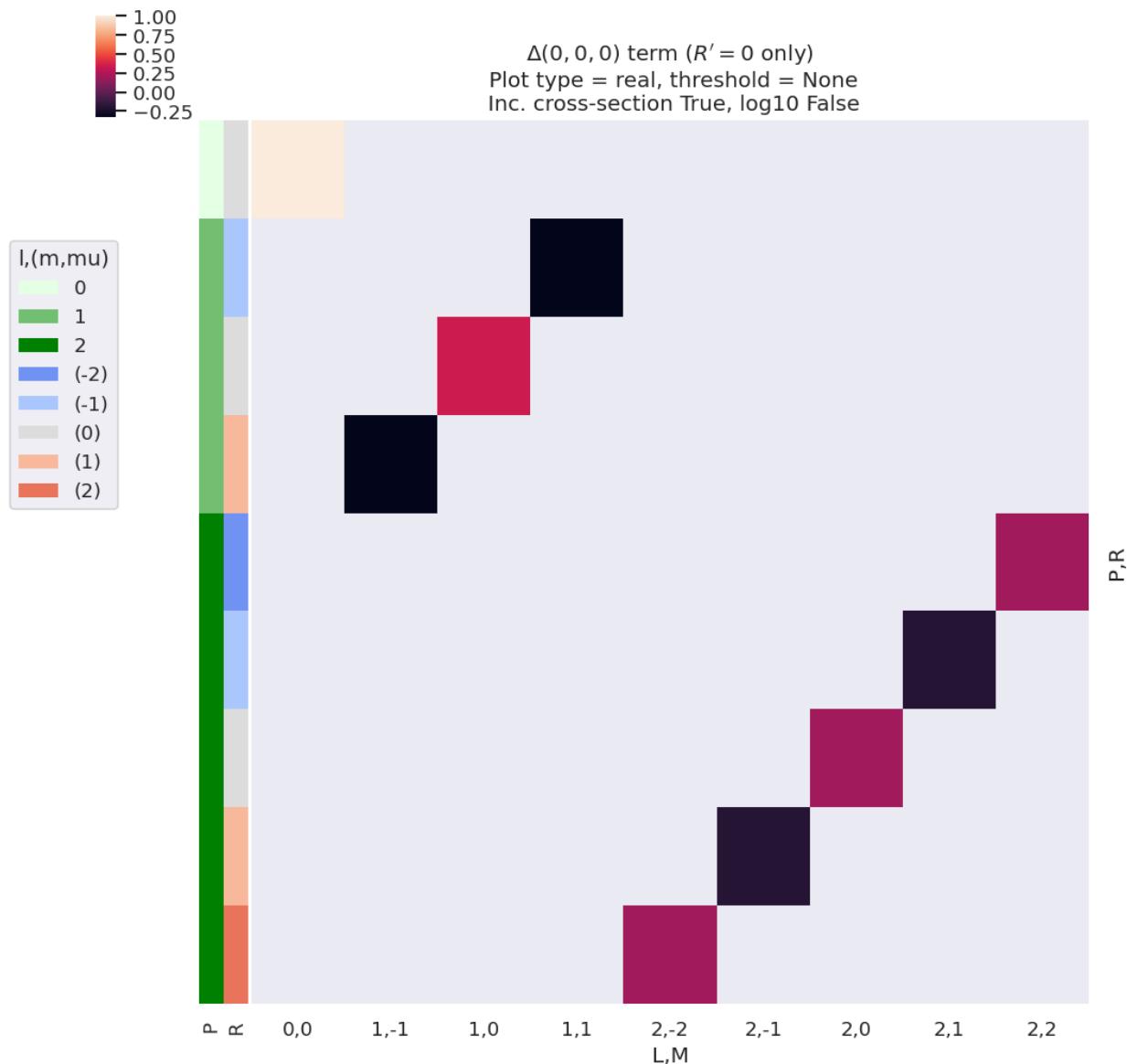


Fig. 6.5: Example $\Delta_{L,M}(0,0,0)$ basis functions (see also Fig. 6.6). For illustration purposes, the plot only shows terms for $R' = 0$. See main text for discussion.

	L	0	1		2			
M	0	-1	0	1	-2	-1	0	1
P	R							
0	0	1.0						
1	-1				-0.333			
	0			0.333				
	1		-0.333					
2	-2						0.2	
	-1					-0.2		
	0				0.2			
	1				-0.2			
	2				0.2			

Fig. 6.6: Example $\Delta_{L,M}(0,0,0)$ basis functions (see also Fig. 6.5). For illustration purposes, the table only shows terms for $R' = 0$. See main text for discussion.

(continued from previous page)

```
labelCols=labelCols,
cmap='vlag',
titleString='$\tilde{\Delta}_{L,M}(t)$ (linear ramp example)'

# Glue versions for JupyterBook output
glue("ADMs-linearRamp-lmPlot", ADMFfig.fig, display=False)
glue("AFterm-linearRamp-lmPlot", AFFig.fig, display=False)
```

For cases with aligned molecular ensembles, additional terms can similarly appear depending on the alignment as well as the properties of the ionizing radiation. Again, the types of terms follow some typical patterns dependent on the symmetry of the ensemble, as well as the order of the terms allowed. For instance, $L_{max} = P_{max} + K_{max} = 2 + K_{max}$, and K_{max} represents the overall degree of alignment of the ensemble; hence an aligned ensemble may be signified by higher-order terms in the observable (if allowed by other terms in the overall expansion) or, equivalently, aligning an ensemble prior to ionization can be used as a way to control which terms contribute to the alignment tensor.

Since this is a coherent averaging, additional interferences can also appear in the AF - or be restricted in the AF - depending on these geometric parameters and the contributing matrix elements. Additionally, any effects modulating these terms, for instance a time-dependent alignment (rotational wavepacket), vibronic dynamics (vibrational and/or electronic wavepacket), time-dependent laser field (control field) may be anticipated to lead to both changes in these terms and, potentially, interesting effects in the observable. Such effects have been discussed in more detail in *Quantum Metrology* Vol. 2 [9], and in the current case the focus is purely on rotational wavepackets.

Fig. 6.8 shows $\tilde{\Delta}_{L,M}(t)$ for various choices of alignment (as per the *ADMs* shown in Fig. 6.7), and illustrates some of the general features discussed. Note, for example:

- L_{max} varies with alignment; in the demonstration case $K_{max} = 8$ at later times, resulting in $L_{max} = 10$, whilst at $t = 0$ $K_{max} = 0$, thus restricting terms to $L_{max} = 2$.
- Odd- L values are correlated with $P = 1$ terms.
- Only $M = 0$ terms are allowed in this case ($Q = S = 0$).

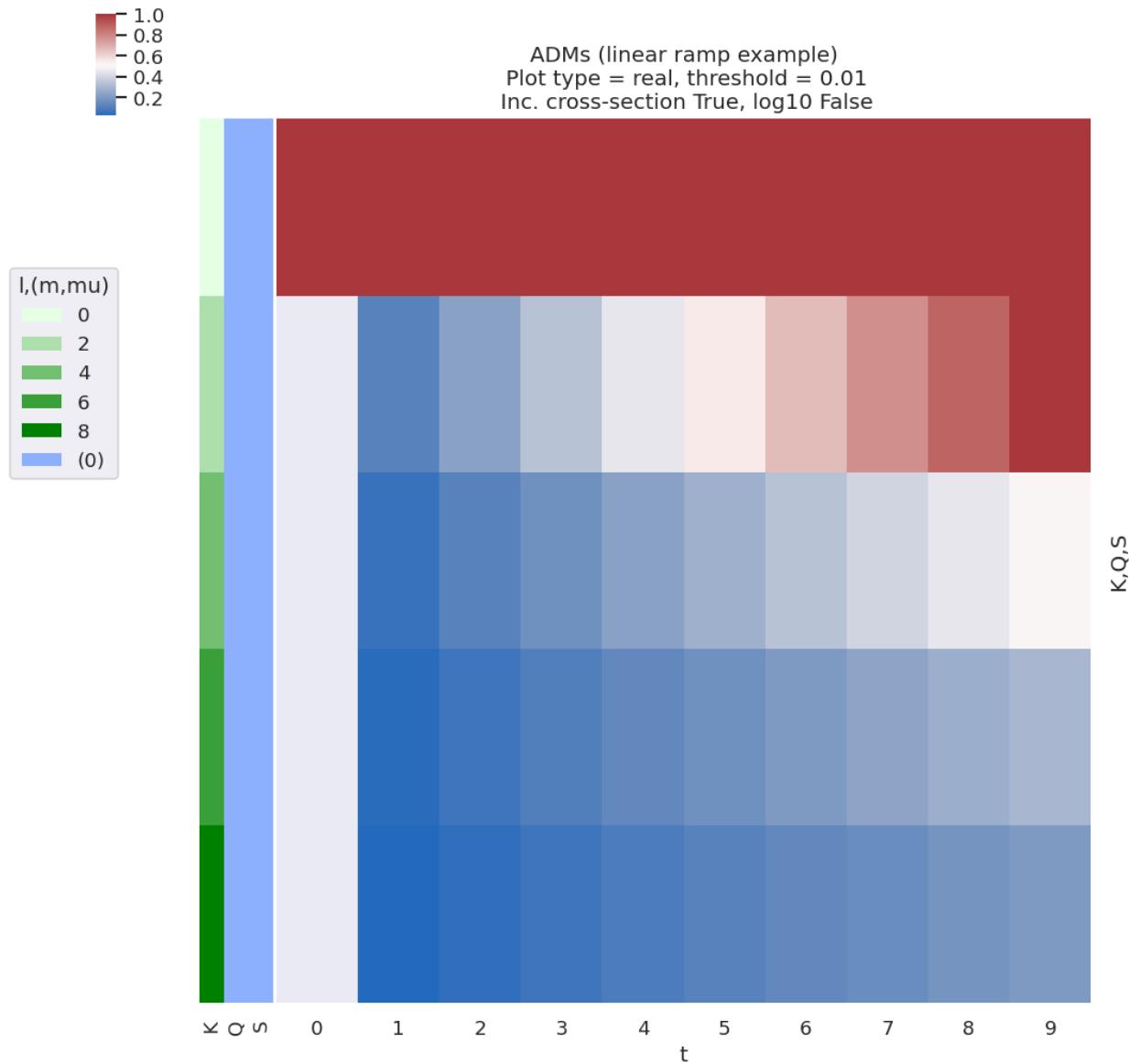


Fig. 6.7: Example ADMs used for AF basis function example (see Fig. 6.8). These ADMs essentially show an increasing degree of alignment with the t parameter, with high-order terms increasing at later t .

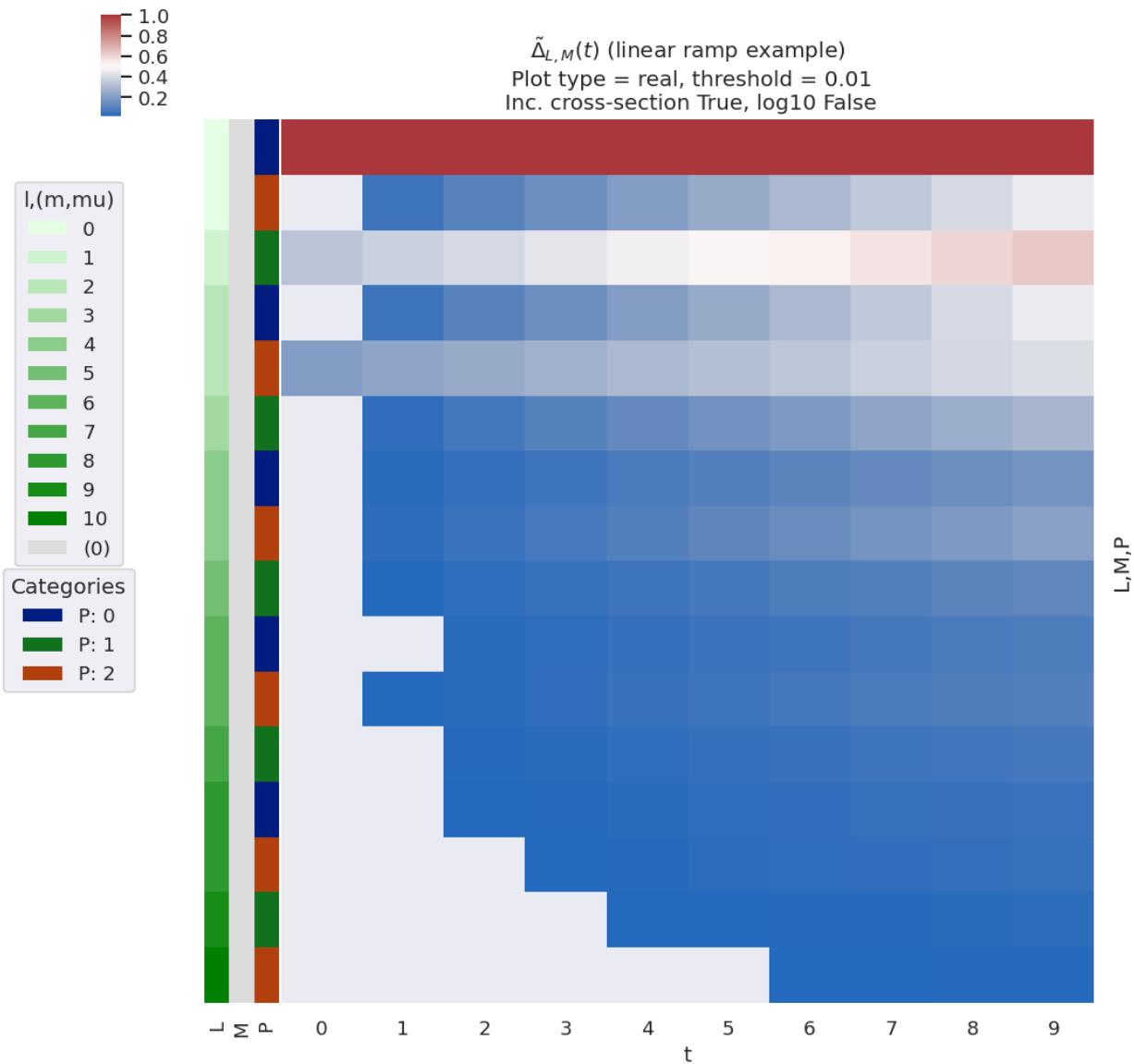


Fig. 6.8: Example of $\tilde{\Delta}_{L,M}(t)$ basis values for various choices of alignment (as per Fig. 6.7). The ADMs essentially show an increasing degree of alignment with the t parameter, with high-order terms increasing at later t , and this is reflected in the $\tilde{\Delta}_{L,M}(t)$ terms with higher-order L appearing at later t .

3D alignments and symmetry breaking

As discussed above, for the case where $Q \neq 0$ and/or $S \neq 0$ additional symmetry breaking can occur. It is simple to examine these effects numerically via changing the trial *ADMs* used to determine $\tilde{\Delta}_{L,M}(t)$ (Eq. (6.28)). (Realistic cases can be found in the case-studies presented in *Part II*.)

```
#*** Neater version
# Set ADMs for increasing alignment...
# Here add some (arb) terms for Q,S non-zero (indexed as [K,Q,S, ADMs(t)])
tPoints = 10
inputADMs3D = [[0,0,0, *np.ones(tPoints)],
                [2,0,0, *np.linspace(0,1,tPoints)],
                [2,0,2, *np.linspace(0,0.5,tPoints)],
                [2,2,0, *np.linspace(0,0.5,tPoints)],
                [2,2,2, *np.linspace(0,0.8,tPoints)]]

# Set to AKQS parameters in an Xarray
AKQS = ep.setADMs(ADMs = inputADMs3D)

# Compute alignment terms
AFterm, DeltaTerm = ep.geomCalc.deltaLMKQS(EPR, AKQS)

#*** Plot subsection, L<=2, and sum over Rp and S-Rp terms
titleString = ('$\\tilde{\\Delta}_{L,M}(t)$ (3D alignment ramp example).'
               '\n Summed over $R'$ and $S-R'$ terms.')
*, gFig = ep.lmPlot(AFterm.where(AFterm.L<=2).sum('Rp').sum('S-Rp'),
                     xDim = 't', pType = 'r',
                     cmap='vlag',
                     titleString=titleString)

# Glue versions for JupyterBook output
glue("ADMs-3DlinearRamp-lmPlot", gFig.fig, display=False)
```

For illustration purposes, Fig. 6.9 shows a subselection of the $\tilde{\Delta}_{L,M}(t)$ basis values, indicating some of the key features in the full 3D case, subselected for $L \leq 2$ and summed over R' and $S - R'$ terms. Note, in particular, the presence of $M \neq 0$ terms in general, and a complicated dependence of the allowed terms on the alignment, which may increase, decrease, or even change sign. As previously, these behaviours are generally useful for understanding specific cases or planning experiments for specific systems; this is explored further in *Part II* which focuses on the results for particular molecules (hence symmetries and sets of matrix elements).

6.3.9 Tensor product terms

Following the above, further resultant terms can also be examined, up to and including the full channel functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ (see (6.13)) for a given case. Numerically these are all implemented in the main ePSproc codebase [33, 34, 35], and can be returned by these functions for inspection - the full basis set already defined includes some of these products. Custom tensor product terms are also readily computed with the codebase, with tensor multiplications handled natively by the *Xarray* [53, 54] data objects (for more details of the data structures used, see the ePSproc documentation [35], specifically the *data structures* page).

Polarisation & ADM product term: the main product basis returned, labelled *polProd* in the output dictionary, contains the tensor product $\Lambda_R \otimes E_{PR}(\hat{e}) \otimes \Delta_{L,M}(K, Q, S) \otimes A_{Q,S}^K(t)$, expanded over all quantum numbers (see [full definition here](#)). This term, therefore, incorporates all of the dependence (or response) of the AF- β_{LMS} s on the polarisation state, and the axis distribution. Example results, making use of the linear-ramp *ADMs* of Sect. 6.3.8 are illustrated in Fig. 6.10.

The full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ as defined in (6.18) and (6.19) can be determined by the product of this

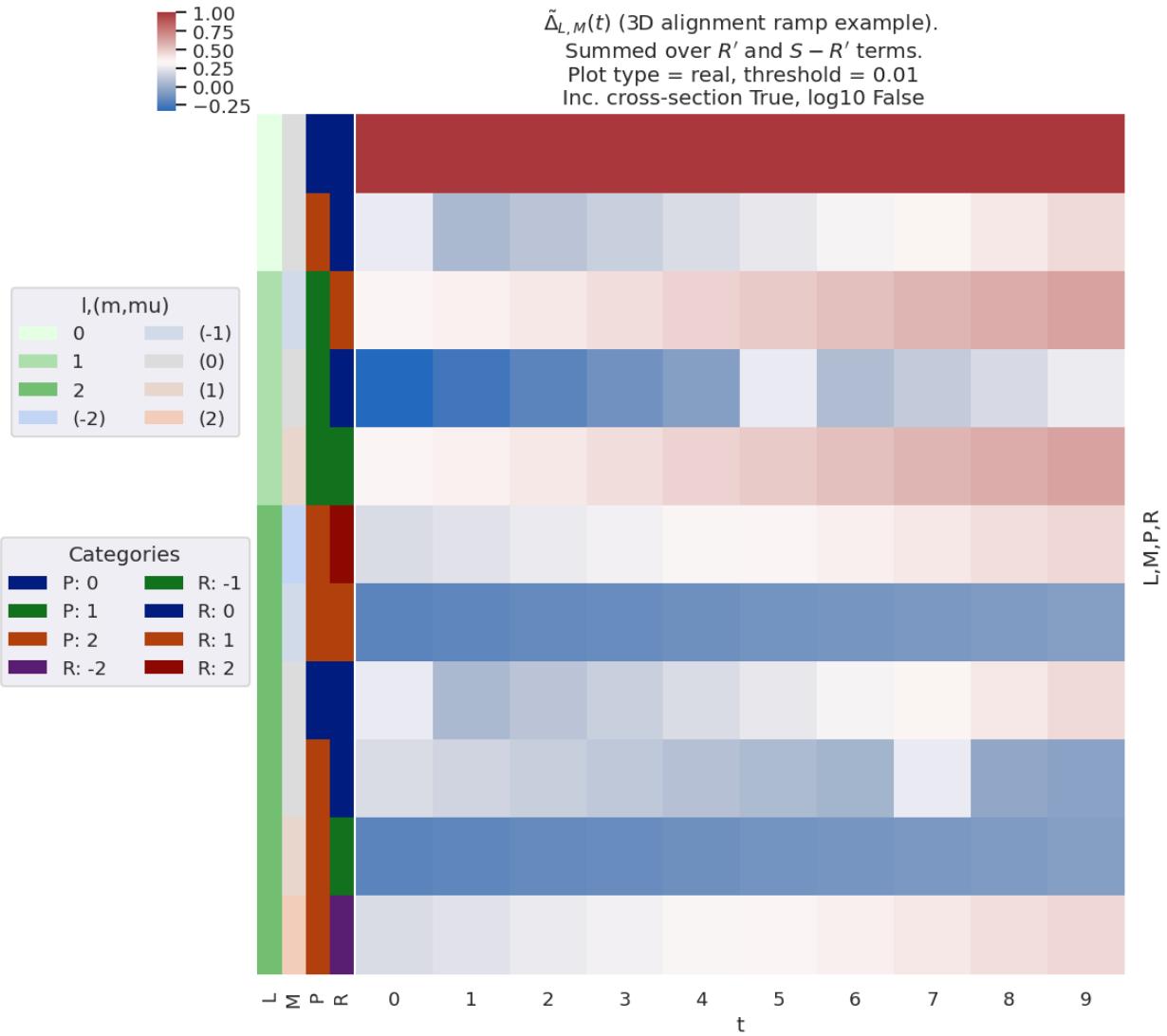


Fig. 6.9: Example $\tilde{\Delta}_{L,M}(t)$ basis values for various choices of “3D” alignment, i.e. including some $K \neq 0$ and $S \neq 0$ terms. Note, in particular, the presence of $M \neq 0$ terms in general, and a complicated dependence of the allowed terms on the alignment (*ADMs*), which may increase, decrease, or even change sign for a given L, M .

term with the $B_{L,M}$ tensor. This is essentially the complete geometric basis set, hence equivalent to the $\text{AF-}\beta_{LM}$ if the ionization matrix elements were set to unity. This illustrates not only the coupling of the geometric terms into the observable L, M , but also how the partial wave $|l, m\rangle$ terms map to the observables, and hence the sensitivity of the observables to given partial wave properties. Example results, making use of the linear-ramp *ADMs* of Sect. 6.3.8 are illustrated in Fig. 6.11.

```
# Set data - set example ADMs to data structure & subset for calculation
data.setADMs(ADMs = inputADMs)
data.setSubset(dataKey = 'ADM', dataType = 'ADM')

# Using PEMtk - this only returns the product basis set as used for fitting
BetaNormX, basisProduct = data.afblmMatEfit(selDims={}, sqThres=False)
```

Subselected from dataset 'ADM', dataType 'ADM': 50 from 50 points (100.00%)

```
basisKey = 'polProd' # Key for BLM basis set

# Plot with subselection on pol state (by label, 'A'=z-pol case)
titleString=('"Polprod" basis term, $\Lambda_{\{R\}}\otimes E_{\{PR\}}(\hat{e})\otimes$'
            '$\Delta_{\{L,M\}(K,Q,S)}\otimes A^{\{K\}_{\{Q,S\}}(t)}$.'.
            '\nSubselected for $z$-pol.')
*, gFig = ep.lmPlot(basisProduct[basisKey].sel(Labels='A'),
                     xDim='t', cmap=cmap, mDimLabel='mu',
                     labelCols=labelCols,
                     titleString=titleString);

# Glue versions for JupyterBook output
glue("polProd-linearRamp-lmPlot", gFig.fig, display=False)
```

```
# Full channel functions
# Plot with subselection on pol state (by label, 'A'=z-pol case)
titleString='Channel functions example.\nSubselected for $z$-pol, $S-R\|=0$.'
*, gFig = ep.lmPlot((basisProduct['BLMtableResort'] *
                     basisProduct['polProd']).sel(Labels='A').sel({'S-Rp':0}).
                     sel(L=2),
                     xDim='t', cmap=cmap, mDimLabel='m',
                     titleString=titleString);

# Glue versions for JupyterBook output
glue("channelFunc-linearRamp-lmPlot", gFig.fig, display=False)
```

6.4 Density matrix representation

6.4.1 General introduction

For a general introduction, and discussion of density matrix techniques and applications in AMO physics, see Blum's textbook *Density Matrix Theory and Applications* [97], which is referred to extensively herein. The general density operator, for a mixture of independent states $|\psi_n\rangle$, can be defined as per Eqn. 2.8 in Blum [97]:

$$\hat{\rho} = \sum_n W_n |\psi_n\rangle\langle\psi_n|$$

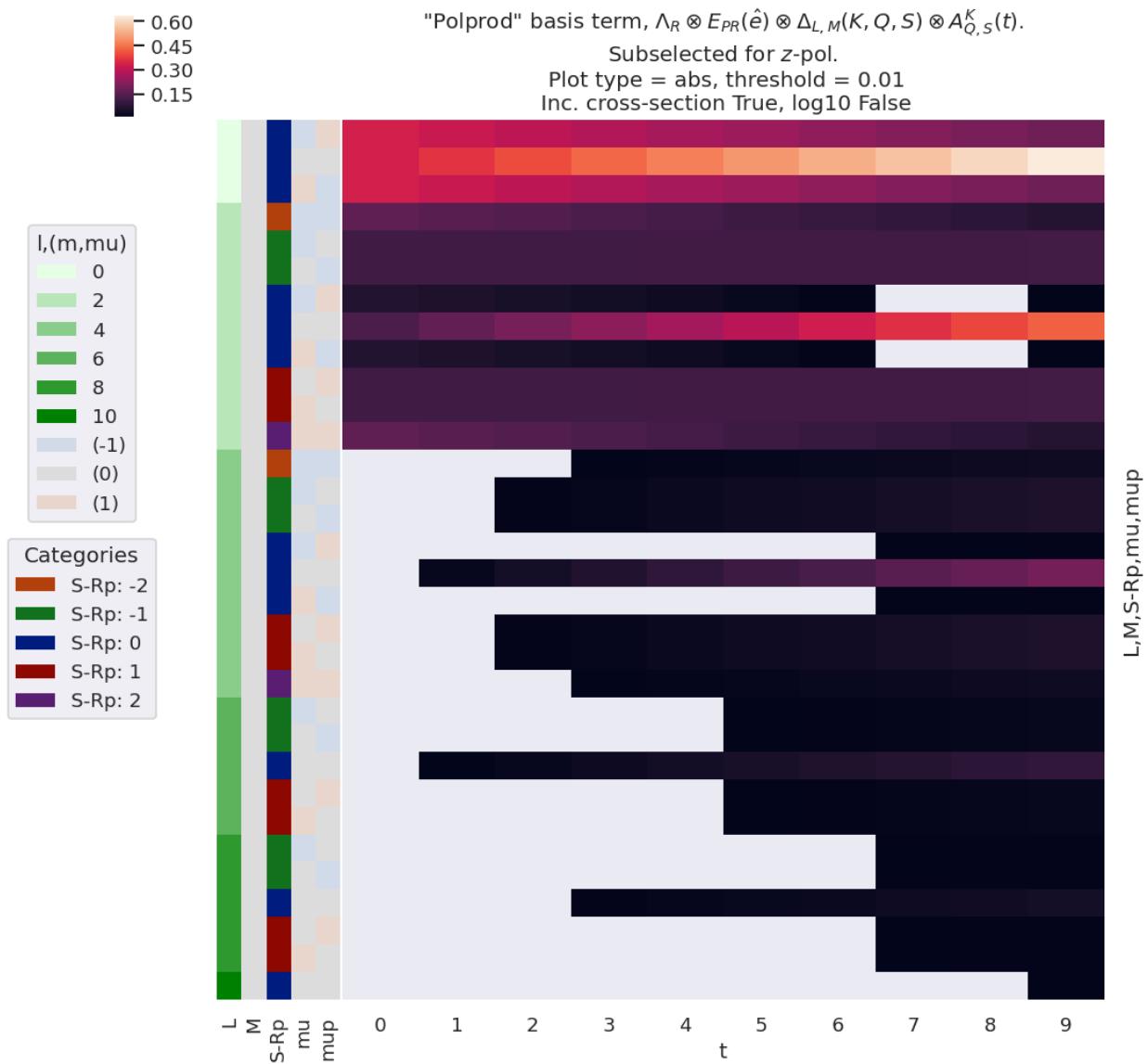


Fig. 6.10: Example product basis function for the polarisation and ADM terms, as given by $\Lambda_R \otimes E_{PR}(\hat{e}) \otimes \Delta_{L,M}(K, Q, S) \otimes A_{Q,S}^K(t)$. Shown for z-pol case only ($p = 0$).

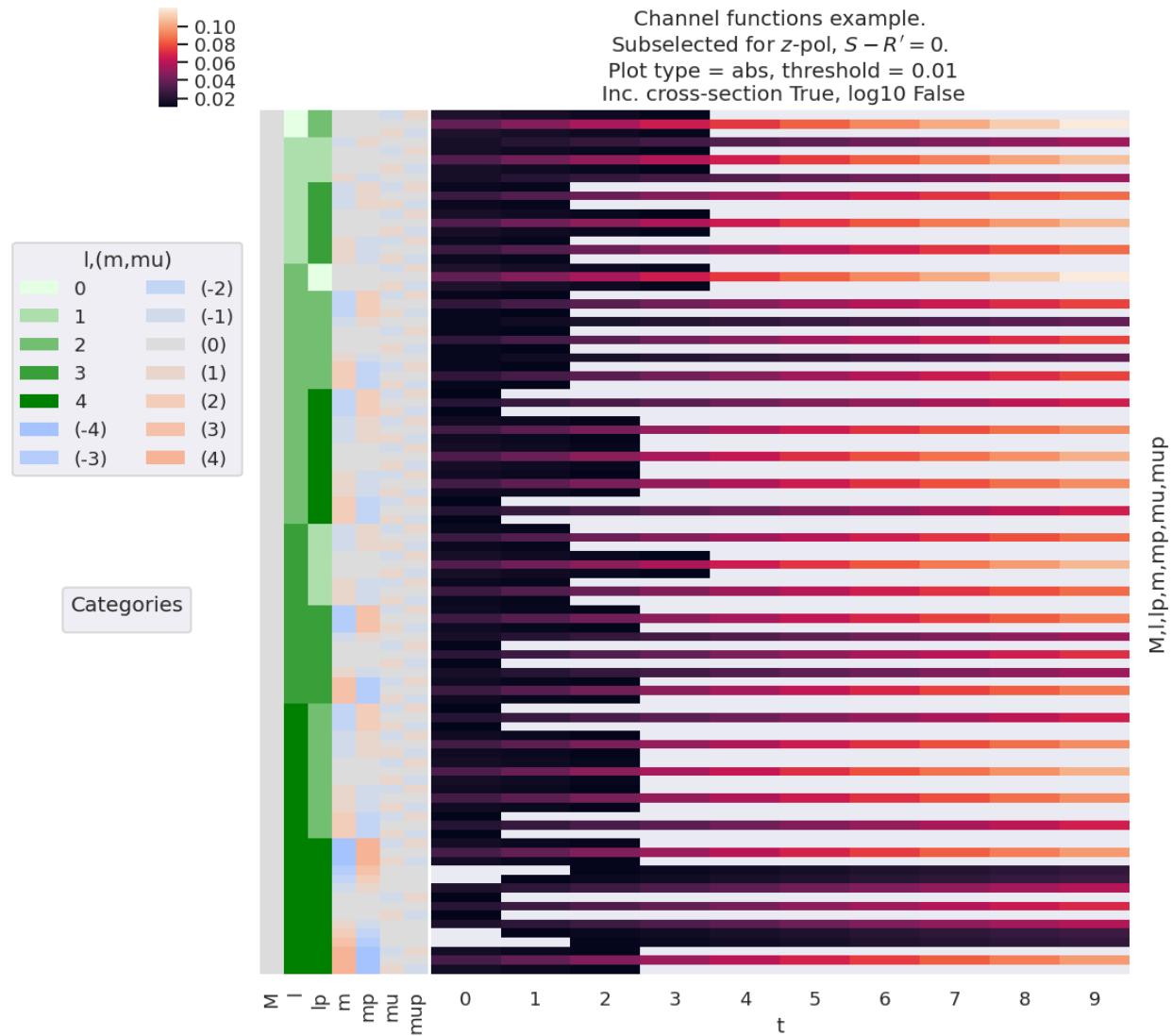


Fig. 6.11: Example of $\tilde{\Upsilon}_{L,M}^{u,\zeta\zeta'}$ basis values for various choices of alignment (as per Fig. 6.7), shown for $L = 2$ and z -pol case only ($p = 0$). The basis essentially shows the observable terms if the ionization matrix elements are neglected, hence the sensitivity of the configuration to each pair of partial wave terms. Note, in general, that the sensitivity to any given pair $\langle l', m' | l, m \rangle$, increases with alignment (hence with t in this example) for the linear polarisation case ($\mu = \mu' = 0$), but typically decreases with alignment for cross-polarised terms.

Where W_n defines the (statistical) weighting of each state ψ_n in the mixture.

For a given basis set, $|\phi_m\rangle$, the states can be expanded and the matrix elements of ρ defined as per Eqns. 2.9 - 2.11 in Blum [97]:

$$\begin{aligned} |\psi_n\rangle &= \sum_{m'} a_{m'}^{(n)} |\phi_{m'}\rangle \\ \hat{\rho} &= \sum_n \sum_{mm'} W_n a_{m'}^{(n)} a_m^{(n)*} |\phi_{m'}\rangle \langle \phi_m| \end{aligned} \quad (6.29)$$

And the matrix elements - *the density matrix* - given explicitly as:

$$\rho_{i,j} = \langle \phi_i | \hat{\rho} | \phi_j \rangle = \sum_n W_n a_i^{(n)} a_j^{(n)*} \quad (6.30)$$

For all pairs of basis states (i, j) . This defines the density matrix in the $\{|\phi_n\rangle\}$ representation (basis space). Of particular note here is that the mixed states are assumed to be incoherent (independent), whilst the basis expansion is coherent.

6.4.2 Continuum density matrices

In general, the discussion herein will focus on the photoelectron properties and generally assume a single final ion, and associated free-electron state of interest, hence the final state (Eq. (6.1)) can be simplified to $|\Psi_f\rangle \equiv |\mathbf{k}\rangle$. This is equivalent to a “pure state” in density matrix terminology, which can then be expanded (coherently) in an appropriate representation (basis). Following this, the density operator associated with the continuum state can be written as $\hat{\rho} = |\Psi_f\rangle \langle \Psi_f| \equiv |\mathbf{k}\rangle \langle \mathbf{k}|$. Making use of the tensor notation introduced in Sect. 6.3, the final continuum state can then be expanded as a density matrix in the $\zeta\zeta'$ representation (with the observable dimensions $\{L, M\}$ explicitly included in the density matrix), which will also be dependent on the choice of *channel functions* (hence “experiment” u); the density matrix can then be given as:

$$\rho_{L,M}^{u,\zeta\zeta'} = \Upsilon_{L,M}^{u,\zeta\zeta'} \mathbb{I}^{\zeta,\zeta'} \quad (6.31)$$

Here the density matrix can be interpreted as the final, **LF/AF** or **MF** density matrix (depending on the *channel functions* used), incorporating both the intrinsic and extrinsic effects (i.e. all channel couplings and radial matrix elements for the given measurement), with dimensions dependent on the unique sets of quantum numbers required - in the simplest case, this will just be a set of partial waves $\zeta = \{l, m\}$.

In the channel function basis, a radial, or reduced, form of the density matrix can also be constructed, and is given by the coherent product of the radial matrix elements (as defined in Eq. (6.14)):

$$\rho^{\zeta\zeta'} = \mathbb{I}^{\zeta,\zeta'} \quad (6.32)$$

This form encodes purely intrinsic (molecular scattering) photoionization dynamics (thus characterises the scattering event), whilst the full form $\rho_{L,M}^{u,\zeta\zeta'}$ of Eq. (6.31) includes any additional effects incorporated via the channel functions. For reconstruction problems, it is usually the reduced form of Eq. (6.32) that is of interest, since the remainder of the problem is already described analytically by the *channel functions* $\Upsilon_{L,M}^{u,\zeta\zeta'}$. In other words, the retrieval of the radial matrix elements $\mathbb{I}^{\zeta,\zeta'}$ and the radial density matrix $\rho^{\zeta\zeta'}$ are equivalent, and both can be viewed as completely describing the photoionization dynamics.

The L, M notation for the full density matrix $\rho_{L,M}^{u,\zeta\zeta'}$ (Eq. (6.31)) indicates here that these dimensions should not be summed over, hence the tensor coupling into the $\beta_{L,M}^u$ parameters can also be written directly in terms of the density matrix (cf. Eq. (6.13)):

$$\beta_{L,M}^u = \sum_{\zeta,\zeta'} \rho_{L,M}^{u,\zeta\zeta'} \quad (6.33)$$

In fact, this form arises naturally since the $\beta_{L,M}^u$ terms are the state multipoles (geometric tensors) defining the system, which can be thought of as a coupled basis equivalent of the density matrix representations (see, e.g., Ref. [97], Chpt. 4.).

In a more traditional notation (following Eq. (6.1), see also Ref. [99]), the density operator can be expressed as:

$$\rho(t) = \sum_{LM} \sum_{KQS} A_{QS}^K(t) \sum_{\zeta\zeta'} \Upsilon_{L,M}^{u,\zeta\zeta'} |\zeta, \Psi_+\rangle\langle\zeta, \Psi_+| \mu_q \rho_i \mu_{q'}^* |\zeta', \Psi_+\rangle\langle\zeta', \Psi_+| \quad (6.34)$$

This is, effectively, equivalent to an expansion in the various tensor operators defined in the channel function notation above (Eq. (6.31)), but in a standard state-vector notation. Note, also, that this form explicitly defines the initial state of the system as a density matrix $\rho_i = |\Psi_i\rangle\langle\Psi_i|$, and explicitly allows for time-dependence via the $A_{QS}^K(t)$ term. (For further discussion of the use of density matrices in other specific cases, see *Quantum Metrology Vol. 1* [4], particularly Chpts. 2 & 3, and refs. therein.)

The main benefit of a (continuum) density matrix representation in the current work is as a rapid way to visualize the phase relations between the photoionization matrix elements (the off-diagonal density matrix elements), and the ability to quickly check the overall pattern of the elements, hence confirm that no phase-relations are missing and orthogonality relations are fulfilled - some numerical examples are given below. Since the method for computing the density matrices is also numerically equivalent to a tensor outer-product, density matrices and visualizations can also be rapidly composed for other properties of interest, e.g. the various *channel functions* defined herein, providing another complementary methodology and tool for investigation. (Further examples can be found in the [ePSproc documentation](#) [35], as well as in the literature, see, e.g., Ref. [97] for general discussion, Ref. [84] for application in pump-probe schemes.)

Furthermore, as noted above, the density matrix elements provide a complete description of the photoionization event, and hence make clear the equivalence of the “complete” photoionization experiments (and associated continuum reconstruction methods) discussed herein, with general quantum tomography schemes [100]. The density matrix can also be used as the starting point for further analysis based on standard density matrix techniques - this is discussed, for instance, in Ref. [97], and can also be viewed as a bridge between traditional methods in spectroscopy and AMO physics, and more recent concepts in the quantum information sciences (see, e.g., Refs. [101, 102] for recent discussions in this context). A brief numerical diversion in this direction is given in [Sect. 6.4.6](#), which illustrates the use of the [the QuTiP \(*Quantum Toolbox in Python*\) library](#) [103, 104, 105] with the density matrix results derived herein.

6.4.3 Numerical setup

This follows the setup in [Sect. 6.3 Tensor formulation of photoionization](#), using a symmetry-based set of basis functions for demonstration purposes. (Repeated code is hidden in PDF version.)

6.4.4 Compute a density matrix

A basic density matrix computation routine is implemented in the [ePSproc codebase](#) [33, 34, 35]. This makes use of input tensor arrays, and computes the density matrix as an outer-product of the defined dimension(s). The numerics essentially compute the outer product from the specified dimensions, which can be written generally as per Eqs. (6.29), (6.30), where $a_i^{(n)} a_j^{(n)*}$ are the values along the specified dimensions/state vector/representation. These dimensions must be in input arrays, but will be restacked as necessary to define the effective basis space, and all coherent pairs will be computed.

For instance, considering the ionization matrix elements demonstrated herein, setting indexes (quantum numbers) as `[l, m]` will select the $|\zeta\rangle = |l, m\rangle$ basis, hence define the density operator as $\hat{\rho} = |\zeta\rangle\langle\zeta'| = |l, m\rangle\langle l', m'|$ and the corresponding density matrix elements $\rho^{\zeta, \zeta'} = \langle\zeta|\hat{\rho}|\zeta'\rangle = a_{l,m} a_{l',m'}^*$. Similarly, setting `['l', 'm', 'mu']` will set the $|\zeta\rangle = |l, m, \mu\rangle$ as the basis vector and so forth, where $|\zeta\rangle$ is used as a generic state vector denoting all required quantum numbers. Additionally, other quantum numbers/dimensions can be kept, summed or selected from the input tensors prior to computation, thus density matrices can be readily computed as a function of other parameters, or averaged, according to the properties of interest, experimental parameters and observables.

Note, however, that this selection is purely based on the numerics, which compute the outer product along the defined dimensions $|\zeta\rangle\langle\zeta'|$ to form the density matrix, hence does not guarantee a well-formed density matrix in the strictest sense (depending on the basis set), although will always present a basis state correlation matrix of sorts. A brief example, for the D2h defined matrix element is given below; for more examples see the ePSproc documentation [35].

```

# See the docs for more,
# https://epsproc.readthedocs.io/en/dev/methods/density_mat_notes_demo_300821.html

# Import routines for density calculation and plotting
from epsproc.calc import density

#*** Compose density matrix

# Set dimensions/state vector/representation
# These must be in original data, but will be restacked as
# necessary to define the effective basis space.

# Set dimensions for density matrix. Note stacked dims are OK, in this case LM = {l,m}
denDims = 'LM'
selDims = None # Select on any other dimensions?
sumDims = None # Sum over any other dimensions?
# (Set sumDims=True to sum over all dims except denDims.)
pTypes=['r','i'] # Plotting types 'r'=real, 'i'=imaginary
thres = 1e-4 # Threshold for outputs (otherwise set to zero and/or dropped from
# result)
normME = False # Normalise matrix elements before computing?
normDen = 'max' # Method to normalise density matrix

# Calculate - Ref case
k = sym
matE = data.data[k]['matE'].copy() # Set data from main class instance by key

# Normalise input matrix elements?
if normME:
    matE = matE/np.max()

#*** Compute density matrix for given parameters
# See demo at:
#   https://epsproc.readthedocs.io/en/latest/methods/density_mat_notes_demo_300821.
#   html
# API docs:
#   https://epsproc.readthedocs.io/en/latest/modules/epsproc.calc.density.html
#   #epsproc.calc.density.densityCalc
daOut, *_ = density.densityCalc(matE, denDims = denDims,
                                 selDims = selDims, thres = thres)

# Renormalise output?
if normDen=='max':
    daOut = daOut/np.max()
elif normDen=='trace':
    # Need sym sum here to get 2D trace
    daOut = daOut/(daOut.sum('Sym').pipe(np.trace)**2)

# Plot density matrix with Holoviews
# Note sum over 'Sym' dimension to flatten plot to (l,m) dims only.
daPlot = density.matPlot(daOut.sum('Sym'), pTypes=pTypes)

```

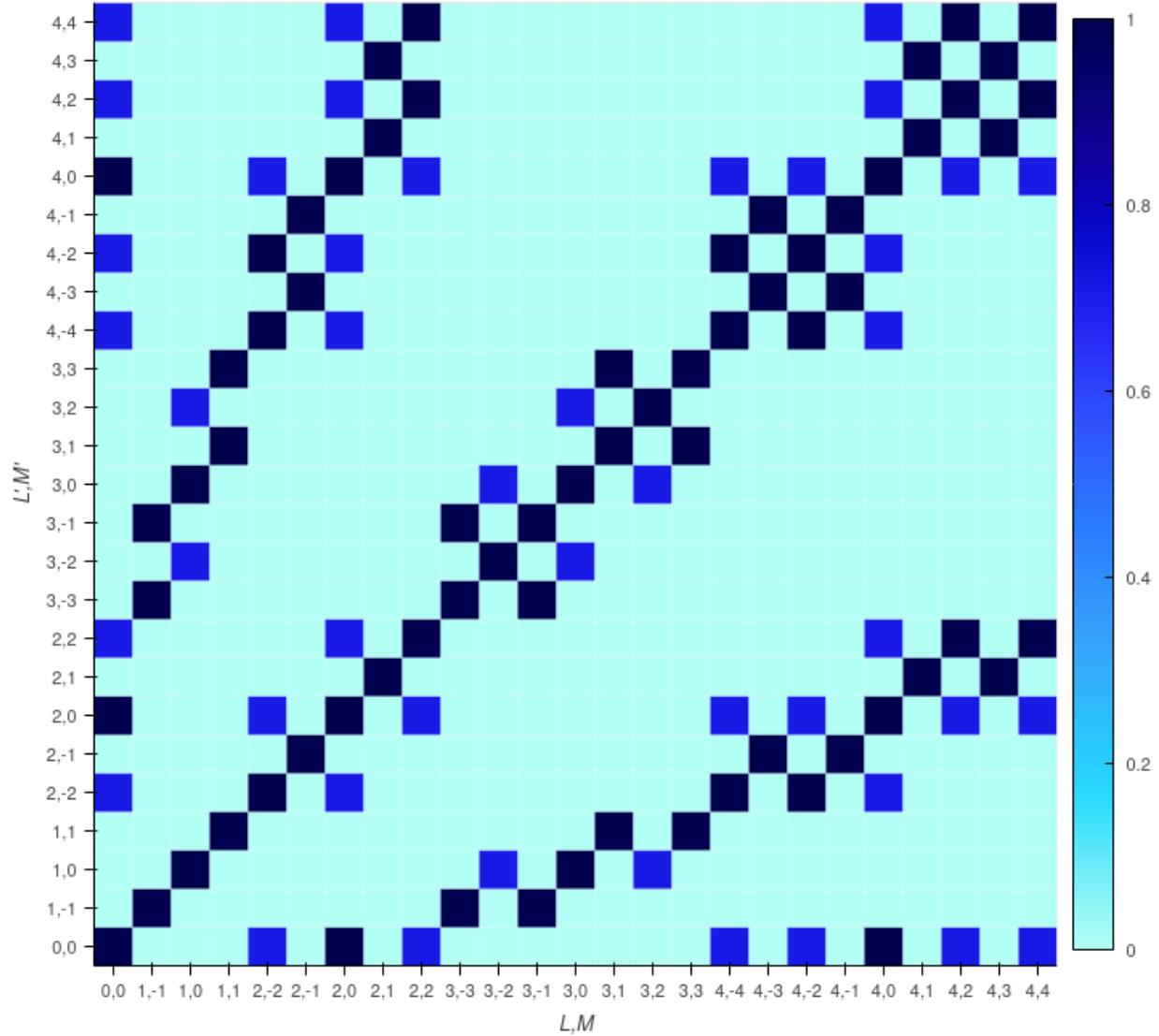


Fig. 6.12: Example density matrix, computed from matrix elements defined purely by D_{2h} symmetry. Note in this case only the real part is non-zero. Axes labels give terms $\{L, M\}$ and $\{L', M'\}$.

6.4.5 Visualising matrix element reconstruction fidelity with density matrices

To demonstrate the use of the density matrix representation as a means to test similarity or fidelity between two sets of matrix elements, a trial set of matrix elements can be derived from the original set used above, plus random noise, and the differences in the density matrices directly computed. An example is shown in Fig. 6.13; in this example up to 10% random noise has been added to the original (input) matrix elements, and the resultant density matrix computed. The difference matrix (Fig. 6.13(c)) then provides the fidelity between the original and noisy case. In testing retrieval methodologies, this type of analysis thus provides a quick means to test reconstruction results vs. known inputs. Although this case is only illustrated for real density matrices, a similar analysis can be used for the imaginary (or phase) components, thus coherences can also be quickly visualised in this manner.

```

 $\text{#*** Set trial matrix element for comparison with the original case computed above}$ 
matE = data.data[k]['matE'].copy()

 $\text{if normME:}$ 
    matE = matE/np.sqrt(np.sum(matE**2))

 $\text{# Add random noise, +/- 10\%}$ 
 $\text{\# Note this is applied to normalised matE}$ 
 $\text{\# For the normalised case this results in a standard deviation in the difference}$ 
 $\text{\# density matrix elements of } \sim \sqrt{2*(0.1^2) + 2*0.1} = 0.2$ 
 $\text{\# (Derived from basic error propagation, ignoring the actual values -}$ 
 $\text{\# see https://en.wikipedia.org/wiki/Propagation_of_uncertainty\#Example_formulae.)}$ 
noise = 0.1
SD = np.sqrt(4*(noise**2))
 $\text{\# Set range to random values +/- 1 * noise}$ 
matE_noise = matE + matE*((np.random.rand(*list(matE.shape)) - 0.5) * 2*noise)

 $\text{\# Compute density matrix}$ 
daOut_noise, *_ = density.densityCalc(matE_noise, denDims = denDims, selDims = selDims, thres = thres)

 $\text{\# Renormalise output?}$ 
 $\text{if normDen=='max':}$ 
    daOut_noise = daOut_noise/daOut_noise.max()
 $\text{elif normDen=='trace':}$ 
    daOut_noise = daOut_noise/(daOut_noise.sum('Sym').pipe(np.trace)**2)

daPlot_noise = density.matPlot(daOut_noise.sum('Sym'), pTypes=pTypes)

 $\text{\# Compute differences}$ 
daDiff = daOut.sum('Sym') - daOut_noise.sum('Sym')
daDiff.name = 'Difference'
daPlotDiff = density.matPlot(daDiff, pTypes=pTypes)

print(f'Noise = {noise}, SD (approx) = {SD}')
maxDiff = daDiff.max().values
print(f'Max difference = {maxDiff}')

 $\text{\#*** Layout plot from Holoviews objects for real parts, with custom titles.}$ 
daLayout = (daPlot.select(pType='Real').opts(title="(a) Original", xlabel='L,M',
                                             ylabel='L,M')
            + daPlot_noise.select(pType='Real').opts(title="(b) With noise",
                                                       xlabel='L,M', ylabel='L,M')
            + daPlotDiff.select(pType='Real').opts(title="(c) Difference (fidelity)",
                                                       xlabel='L,M', ylabel='L,M'))

```

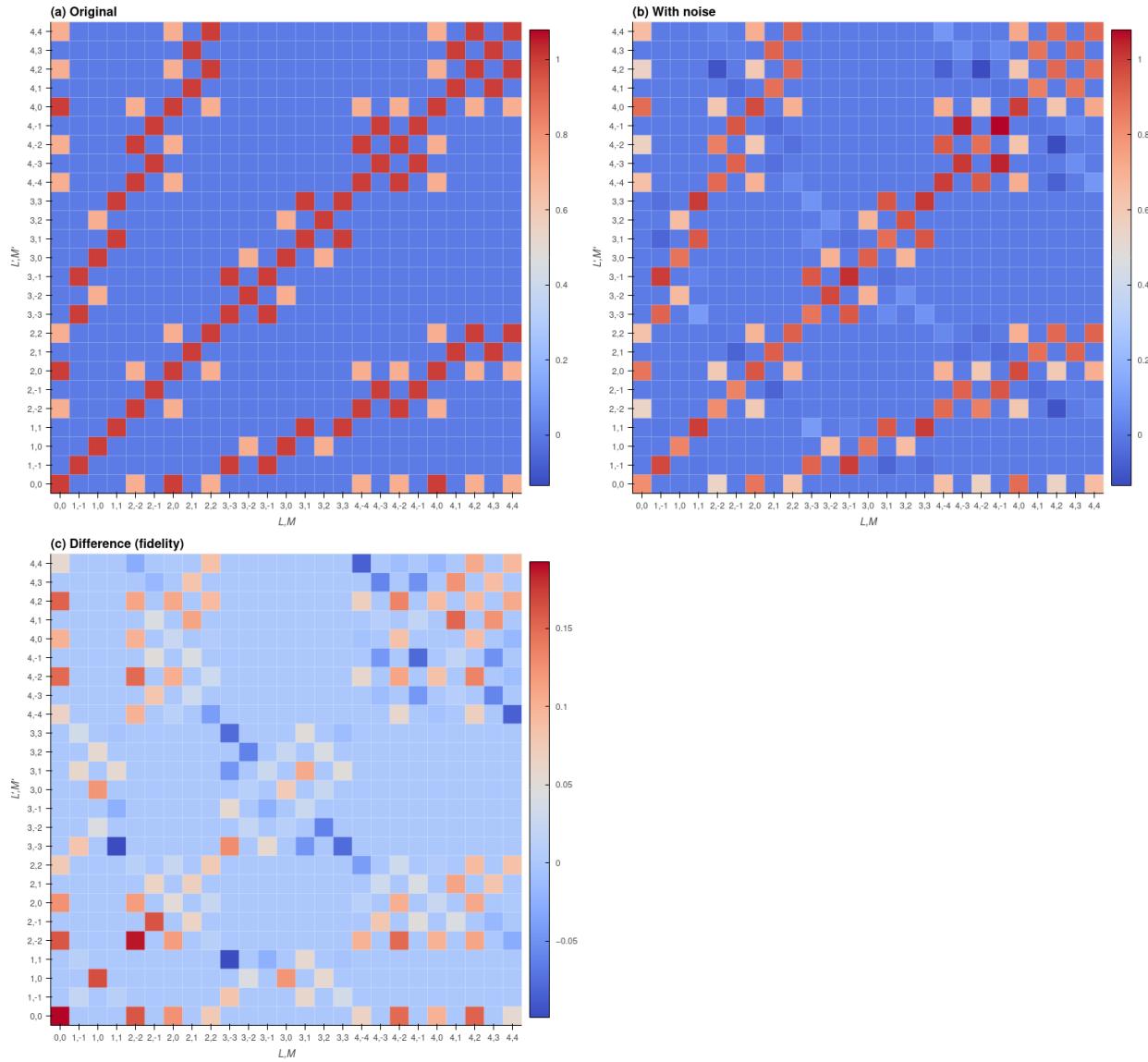


Fig. 6.13: Example density matrices, computed from matrix elements defined purely by D_{2h} symmetry. Here the panels show (a) the original density matrix, (b) density matrix computed with +/- 10% random noise added to the original matrix elements, (c) the difference matrix, which indicates the fidelity of the noisy case relative to the original case. For normalised density matrices the 10% noise case translates to a standard deviation $\sigma \approx 0.2$ on the differences; the maximum error in the test case as illustrated =0.193.

6.4.6 Working with density matrices with QuTiP library functions

From the numerical density matrix, a range of other standard properties can be computed - of particular interest are likely to be various standard quantities such as the trace, Von Neuman entropy and so forth. Naturally these can be computed numerically directly from the relevant formal definitions; however, many of the fundamentals are already implemented in other libraries, and numerical representations can be passed directly to such libraries. In particular, the QuTiP (*Quantum Toolbox in Python*) library [103, 104, 105] implements a range of standard functions, metrics, transforms and utility functions for working with state vectors and density matrices. A brief numerical example is given below, see the QuTiP documentation [105] for more possibilities.

Convert numerical arrays to QuTiP objects

```
# Import QuTip
from qutip import *

# Wrap density matrices to QuTip objects
# Note sum('Sym') to ensure 2D matrix, and .data to pass Numpy data array only
pa = Qobj(daOut.sum('Sym').data)      # Reference continuum density matrix
pb = Qobj(daOut_noise.sum('Sym').data) # Noisy case

# QuTip objects have data as Numpy arrays, and render as typeset matrices in a
# notebook
# DEBUG NOTE 22/04/23 - QuTip matrix latex output currently causing PDF build errors,
# so set hide output for testing.
# See https://github.com/phockett/Quantum-Metrology-with-Photoelectrons-Vol3/issues/8
if buildEnv != 'pdf':
    display(pa)
    # print(pa)
```

Fidelity metric

Fidelity between two density matrices ρ_a, ρ_b can be defined as per Refs. [106, 107]:

$$F(\rho_a, \rho_b) = \text{Tr} \sqrt{\sqrt{\rho_a} \rho_b \sqrt{\rho_a}}$$

This is implemented by the `fidelity` function in the QuTiP (*Quantum Toolbox in Python*) library [103, 104, 105]. Of note in this test case is that the resultant is close to limiting-case value of $F(\rho_a, \rho_b) = 1$ for the test case herein, despite the added noise and some per-element disparities as shown in Fig. 6.13(c). This reflects the conceptual difference between an element-wise evaluation of the differences, vs. a formal scalar metric.

```
# Test fidelity, =1 if trace-normalised
print(f"Fidelity (a,a) = {fidelity(pa,pa)}")
print(f"Trace = {pa.tr()}")
print(f"Trace-normed fidelity = {fidelity(pa,pa)/pa.tr()}")
```

```
Fidelity (a,a) = 25.00000031332397
Trace = 25.0
Trace-normed fidelity = 1.0000000125329587
```

```
# Test fidelity vs noisy case
print(f"Fidelity (a,b) = {fidelity(pa,pb)}")
print(f"Trace a = {pa.tr()}, Trace b = {pb.tr()}")
print(f"Trace-normed fidelity = {fidelity(pa/pa.tr(),pb/pb.tr())}")
```

```
Fidelity (a,b) = 24.09520330700228
Trace a = 25.0, Trace b = 23.289960248949274
Trace-normed fidelity = 0.9985704465969232
```

```
# This can also be computed rapidly with lower-level QuTip functionality...
# Compute inner term, note .sqrtm() for square root.
inner = pa.sqrtm() * pa * pa.sqrtm()

# Compute fidelity
inner.sqrtm().tr()
```

```
(25.00000059788841+8.879641550582872e-08j)
```

6.5 Molecular alignment

6.5.1 A very brief introduction to molecular alignment

The term *molecular alignment* can be used, in general, to define any case where the *MF* is specified relative to the *LF* in some way - for instance if the molecular symmetry axis is constrained to the *LF* *z*-axis. Herein, it is generally used more specifically, to refer to the case of a (time-dependent) aligned molecular ensemble in gas-phase experiments (e.g. as illustrated in Fig. 4.1). Any such axis distribution, in which there is a defined arrangement of axes created in the *LF*, can be discussed, and characterised, in terms of the axis distribution moments (*ADMs*), which have already been introduced passing in Sect. 6.3. More specifically, *ADMs* are coefficients in a multipole expansion, usually in terms of *Wigner rotation matrix elements*, of the molecular axis probability distribution. In this section some additional definitions are given, along with numerical examples.

The creation of an aligned ensemble in the gas phase can be achieved via a single, or sequence of, N-photon transitions, or strong-field mediated techniques. Of the latter, adiabatic and non-adiabatic alignment methods are particularly powerful, and make use of a strong, slowly-varying or impulsive laser field respectively. (Here the “slow” and “impulsive” time-scales are defined in relation to molecular rotations, roughly on the ps time-scale, with ns and fs laser fields corresponding to the typical slow and fast control fields.) In the former case, the molecular axis, or axes, will gradually align along the electric-field vector(s) while the field is present. In the latter, impulsive case, a broad rotational wavepacket (*RWP*) can be created, initiating complex rotational dynamics including field-free revivals of ensemble alignment. For further general discussion, there is a rich literature on molecular alignment available, see, for instance, Refs. [108, 109, 110, 111] for reviews and further introductory materials, and further discussion in the current context can be found in *Quantum Metrology* Vols. 1 & 2 [4, 9] and Refs. [3, 95, 96, 112, 113, 114] and references therein.

For *radial matrix elements* retrieval problems based on *RWP* methods, the absolute degree of alignment may - or may not - be critical in a given case. The sampling of a range of *different* alignments, however, is vital, since this directly feeds into the information content of the measurements (see Sect. 6.3.8 and Sect. 6.7). In the case-studies of *Part II*, the *ADMs* are assumed to be known, but in general these must be determined from experimental data, this is discussed in Sect. 7.1.1.

6.5.2 Alignment distribution moments (ADMs)

The parametrization of an aligned distribution can be given generally by an expansion in *Wigner rotation matrix elements*:

$$P(\Omega, t) = \sum_{K,Q,S} A_{Q,S}^K(t) D_{Q,S}^K(\Omega) \quad (6.35)$$

Where $P(\Omega, t)$ is the full axis distribution probability, expanded for a set of *Euler angles* Ω , and the expansion parameters $A_{Q,S}^K(t)$ are the *ADMs*.

This reduces to the 2D case if $S = 0$, which can equivalently be described as an expansion in spherical harmonics (note that the normalisation of the *ADMs* may be different in this case):

$$P(\theta, \phi, t) = \sum_{K,Q} A_{Q,0}^K(t) D_{Q,0}^K(\Omega) = \sum_{K,Q} A_Q^K(t) Y_{K,Q}(\Omega) \quad (6.36)$$

In the examples given in Sect. 6.3, some arbitrary choices of $A_{Q,S}^K(t)$ were demonstrated to investigate their effects on the tensor basis sets; in the case-studies presented in Part II realistic *ADMs* are used for specific fitting problems. In practice this equates to (accurately) simulating rotational wavepackets, hence obtaining the corresponding $A_{Q,S}^K(t)$ parameters (expectation values), as a function of laser fluence and rotational temperature. (Given experimental data, a 2D uncertainty (or error) surface in these two fundamental quantities can then be obtained from a linear regression for each set of $A_{Q,S}^K(t)$, see Ref. [3] for further introductory discussion on this point.) Note that, as discussed in Sect. 5.5, computation of molecular alignment is not yet implemented in the *Photoelectron Metrology Toolkit* [5], so values must be obtained from other codes. *ADMs* used herein were all computed with codes developed by V. Makhija [115], and are available from the ePSproc [34] repo on Github.

6.5.3 Numerical setup

For illustrative purposes, the *ADMs* used for the N_2 fitting example are here loaded and used to compute $P(\Omega, t)$. (Note these *ADMs* are for a 2-pulse alignment scheme, as outlined in Ref. [1].)

```
# Quick plot for subselected ADMs (setup in the script),
# using basic plotter
data.ADMplot(keys = data.subKey)

# Quick plot for subselected ADMs (setup in the script), using hvplot
# data.data['subset']['ADM'].unstack().squeeze().real.hvplot.line(x='t').overlay('K')

# As above, but plot K>0 terms only, and keep 'Q', 'S' indexes (here all =0)
figObj = data.data['subset']['ADM'].unstack().where(data.data['subset']['ADM'].
    unstack().K>0) \
    .real.hvplot.line(x='t').overlay(['K', 'Q', 'S']).opts(width=700)

# Glue plot for later
glue("fitSystem", fitSystem, display=False)
glue("ADMdemoPlot", figObj)
```

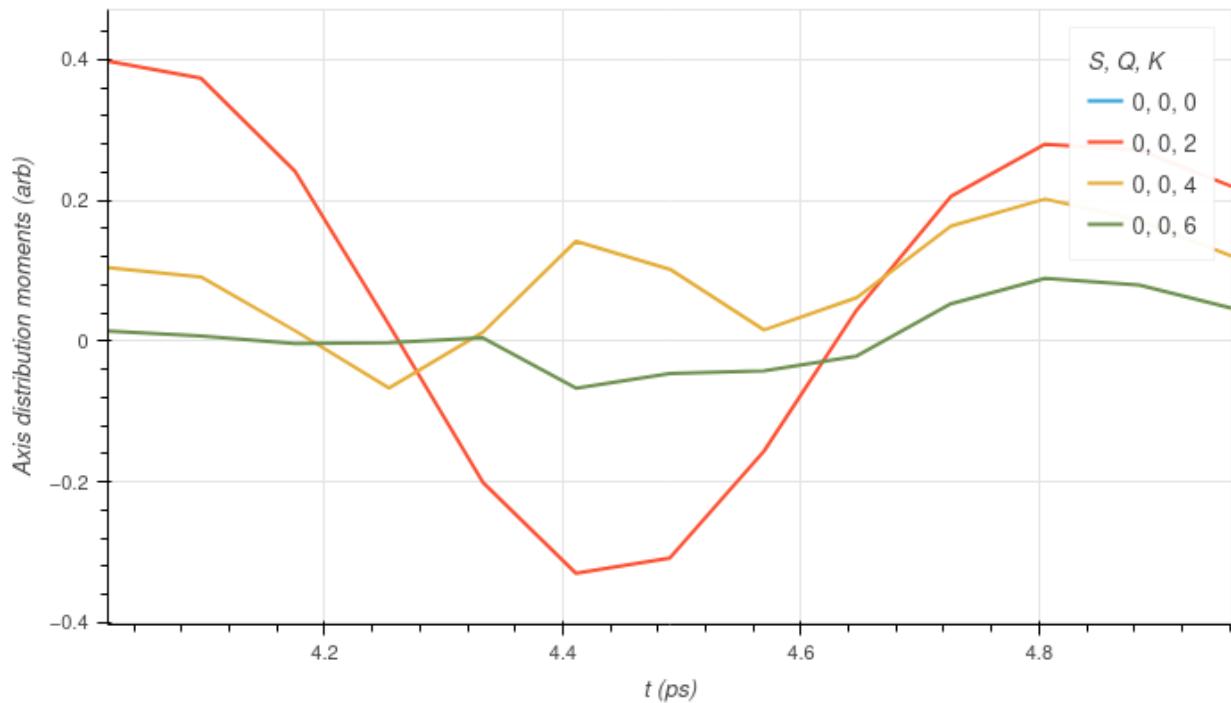


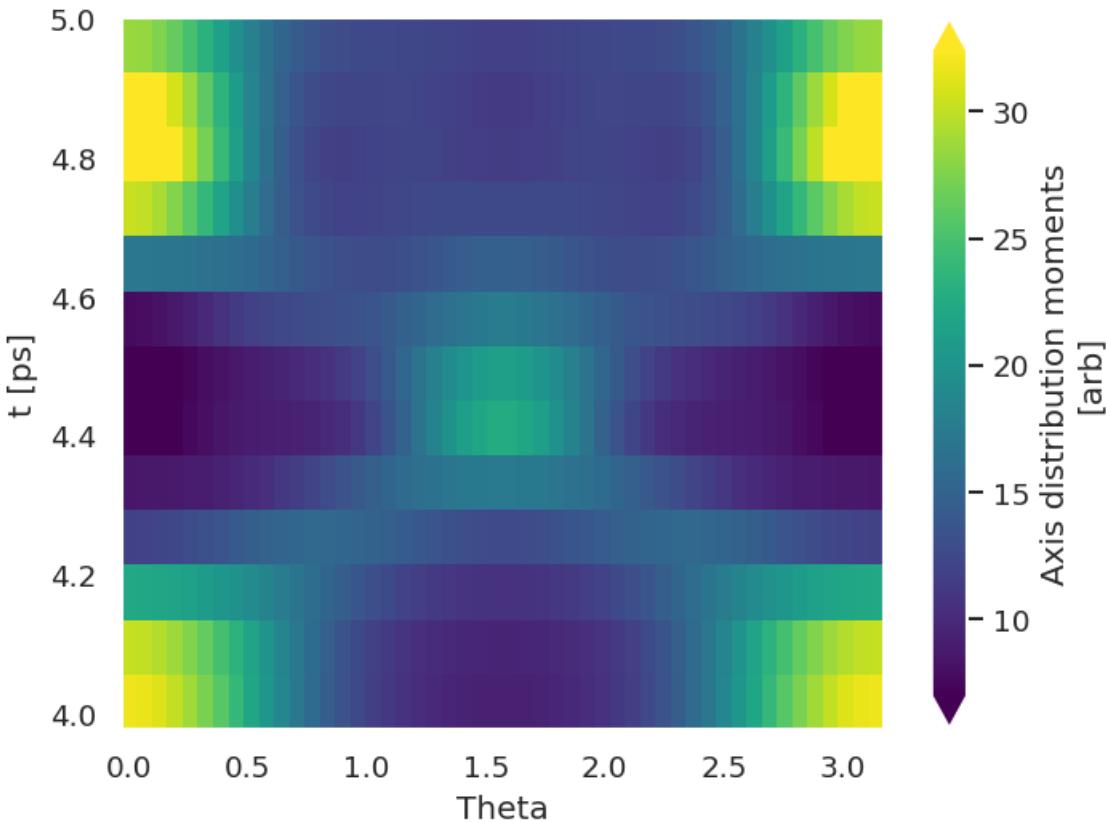
Fig. 6.14: Example ADMs for N2.

6.5.4 Compute $P(\theta, \Phi, t)$ distributions

For 1D and 2D cases, the full axis distributions can be expanded in spherical harmonics and plotted using Photoelectron Metrology Toolkit [5] class methods. This is briefly illustrated below. Note that expansions in *Wigner rotation matrix elements* are not currently supported by these routines.

```
# NOTE - need this in some builds if Matplotlib has call-back errors.
%matplotlib inline
# Plot P(theta,t) with summation over phi dimension
# Note the plotting function automatically expands the ADMs in spherical harmonics
dataKey = 'subset'
data.padPlot(keys = dataKey, dataType='ADM', Etype = 't', pStyle='grid', reducePhi=
    ↪'sum', returnFlag = True)
```

```
Using default sph betas.
Summing over dims: set()
Plotting from self.data[subset][ADM], facetDims=['t', None], pType=a with_
    ↪backend=mpl.
Grid plot: ('subset', 'ADM'), dataType: ADM, plotType: a
Set plot to self.data['subset']['plots']['ADM']['grid']
```



```

# Plot full axis distributions at selected time-steps
# tPlot = [39.402, 40.791, 42.18] # Manual setting for baseline case, and at max and
# min K=2 times. OCS
tPlot = [4.018, 4.254, 4.49] # N2

# Alternatively, plot at selected times by index slice
# Note that selDims below requires labels (not index inds)
# tPlot = data.data[dataKey]['ADM'].t[::5] # OCS
# tPlot = data.data[dataKey]['ADM'].t[0:7:3] # N2

# Plot
ep.plot.hvPlotters.setPlotters(width=1200, height=600)    # Force plot dims for HTML
# render (avoids subplot clipping issues)
data.padPlot(keys = dataKey, dataType='ADM', Etype = 't', pType='a',
            returnFlag = True, selDims={'t':tPlot}, backend='pl')

# And GLUE for display later with caption
figObj = data.data[dataKey]['plots']['ADM']['polar'][0]
glue("axisDistDemo", figObj)

```

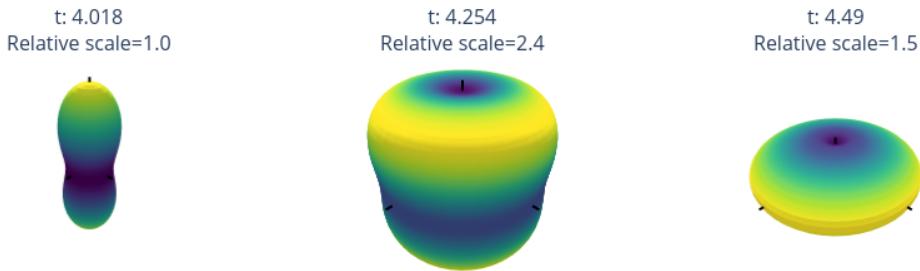


Fig. 6.15: Molecular axis distributions $P(\theta, \phi)$ at selected times. In this demo case the alignment is “1D”, and cylindrically symmetric.

6.6 Observables: photoelectron flux in the LF and MF

The observables of interest herein - the photoelectron flux as a function of energy, ejection angle, and time (see Fig. 4.1) - can be written quite generally as expansions in radial and angular basis functions. Various types and definitions are given in this section, including worked numerical examples. The final section, Sect. 6.6.4, illustrates typical data from a time-resolved measurement, resulting in time and energy dependant observables - this is the type of data required for the *bootstrap retrieval protocol*.

6.6.1 Spherical harmonics

The photoelectron flux as a function of energy, ejection angle, and time, can be written generally as an expansion in spherical harmonics:

$$\bar{I}(\epsilon, t, \theta, \phi) = \sum_{L=0}^{2n} \sum_{M=-L}^L \bar{\beta}_{L,M}(\epsilon, t) Y_{L,M}(\theta, \phi) \quad (6.37)$$

Here the flux in the laboratory frame (*LF*) or aligned frame (*AF*) is denoted $\bar{I}(\epsilon, t, \theta, \phi)$, with the bar signifying ensemble averaging, and the molecular frame flux by $I(\epsilon, t, \theta, \phi)$. Similarly, the expansion parameters $\bar{\beta}_{L,M}(\epsilon, t)$ include a bar for the LF/AF case. These observables are generally termed photoelectron angular distributions (*PADs*), often with a prefix denoting the reference frame, e.g. LFPADs, MFPADs, and the associated expansion parameters $\beta_{L,M}(\epsilon, t)$ are generically termed *anisotropy parameters*. The polar coordinate system (θ, ϕ) is referenced to an experimentally-defined axis in the *LF/AF* case (usually defined by the laser polarization), and the molecular symmetry axis in the *MF*, as illustrated in Fig. 4.1. Some arbitrary examples are given in Fig. 6.16, which illustrates both a range of distributions of increasing complexity, and some basic code to set $\beta_{L,M}$ parameters and visualise them; the values used as tabulated in Fig. 6.17.

Numerically, there are some choices and conventions which apply to the spherical harmonics. As noted in Sect. 5.3: “spherical harmonics are defined with the usual physics conventions: orthonormalised, and including the Condon-Shortley phase. Numerically they are implemented directly or via SciPy’s sph_harm function (see the SciPy docs for details [59].” For further details, including conversion routines, see the pySHTools [60, 61, 62, 63] documentation, and numerical examples below.

```
# Plot some distributions from specified BLMs
# Set specific LM coeffs by list with setBLMs, items are [l,m,value(s)]
```

(continues on next page)

(continued from previous page)

```

# Multiple values are automatically assigned to an index 't'
from epsproc.sphCalc import setBLMs

BLM = setBLMs([[0,0,1,1,1,1,1,1],
                [1,0,0,0.5,0.8,1,0.5,0],[1,-1,0,0.5,0.8,1,0.5,0],[1,1,0,0.5,-0.5,1,0.5,
                ↪0],
                [2,0,1,0.5,0,0,0.5,1],
                [4,2,0,0,0,0.5,0.8,1],[4,-2,0,0,0,0,-0.8,1]])

# Output a quick tabulation of the values with Pandas
BLM.to_pandas()

# Glue for later use
glue("blm-tab", BLM.to_pandas())

```



```

# Note also that the Xarray contains metadata (attributes) on type and normalisation
# This uses the SHtools format specification.

# Display full Xarray, including metadata
BLM

```



```

# Show harmonics info only
BLM.attrs['harmonics']

```



```

{'dtype': 'Complex harmonics',
 'kind': 'complex',
 'normType': 'ortho',
 'csPhase': True}

```



```

# Plot some PADs from BLMs
# Set the backend to 'pl' for an interactive surface plot with Plotly

# Explicit row,column layout setting for figure
rc = [2,3]

# Compute expansions from BLM parameters and return figure object
dataPlot, figObj = ep.sphFromBLMPlot(BLM, facetDim='t', backend = plotBackend, rc=rc, ↪
plotFlag=True);

```

In general, the spherical harmonic rank and order (L, M) of Eq. (6.37) are constrained by experimental factors in the **LF** or **AF**, and n is effectively limited by the molecular alignment (which is correlated with the photon-order for gas phase experiments, or conservation of angular momentum in the **LF** more generally [116]), but in the **MF** is defined by the maximum continuum angular momentum $n = l_{max}$ imparted by the scattering event [117] (note lower-case l here refers specifically to the continuum photoelectron wavefunction, see Eq. (6.5)).

For basic cases these limits may be low: for instance, a simple 1-photon photoionization event ($n = 1$) from an isotropic ensemble (zero net ensemble angular momentum) defines $L_{max} = 2$; for cylindrically symmetric cases (i.e. $D_{\infty h}$ symmetry) $M = 0$ only. For **MF** cases, $l_{max} = 4$ is often given as a reasonable rule-of-thumb for the continuum - hence $L_{max} = 8$ - although in practice higher- l may be populated. Some realistic example cases are discussed later (*Part II*), see also ref. [4] for more discussion and complex examples.

In general, these observables may also be dependent on various other parameters; in Eq. (6.37) two such parameters, (ϵ, t) , are included, as the usual variables of interest. Usually ϵ denotes the photoelectron energy, and t is used in the case of time-dependent (usually pump-probe) measurements. As discussed in Sect. 6.1, the origin of such dependencies

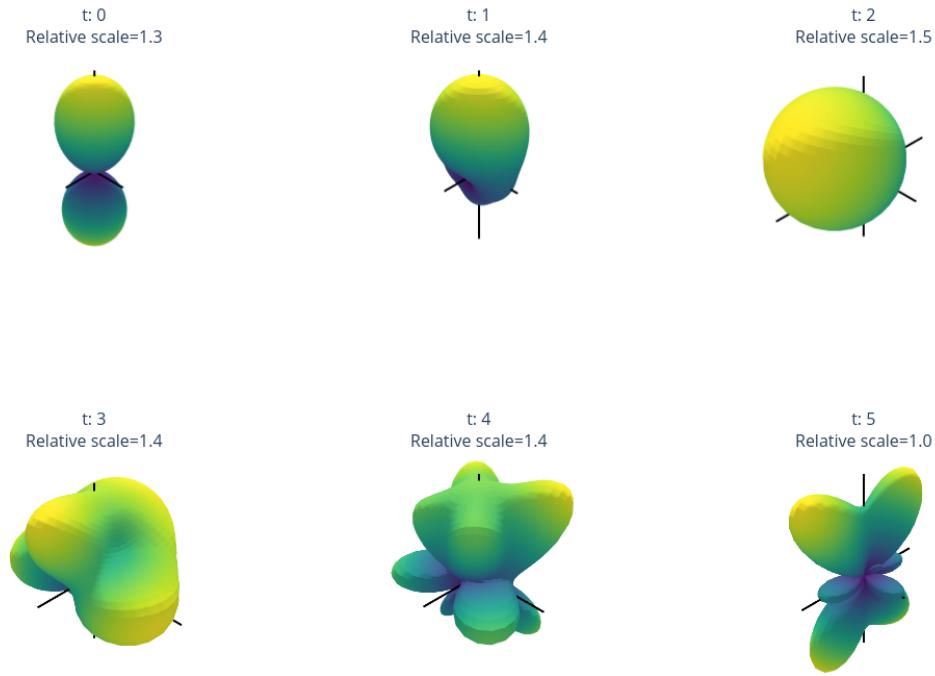


Fig. 6.16: Examples of angular distributions (expansions in spherical harmonics $Y_{L,M}$), for a range of cases indexed by t . Note that up-down asymmetry is associated with odd- l contributions (e.g. $t = 1, 2$), breaking of cylindrical symmetry with $m \neq 0$ terms (all $t > 0$), and asymmetries in the (x,y) plane (skew/directionality) with different $\pm m$ terms (magnitude or phase, e.g. $t = 2, 3, 4$). Higher-order L, M terms have more nodes, and lead to more complex angular structures, as shown in the lower row ($t = 3, 4, 5$).

	t	0	1	2	3	4	5
1	m						
0	0	1.0	1.0	1.0	1.0	1.0	1.0
1	0	0.0	0.5	0.8	1.0	0.5	0.0
	-1	0.0	0.5	0.8	1.0	0.5	0.0
	1	0.0	0.5	-0.5	1.0	0.5	0.0
2	0	1.0	0.5	0.0	0.0	0.5	1.0
4	2	0.0	0.0	0.0	0.5	0.8	1.0
	-2	0.0	0.0	0.0	0.0	-0.8	1.0

Fig. 6.17: Values used for the plots in Fig. 6.16.

may be complicated but, in general, the associated photoionization matrix elements are energy-dependent, and time-dependence may also appear for a number of intrinsic or extrinsic (experimental) reasons, e.g. electronic or nuclear dynamics, rotational (alignment) dynamics, electric field dynamics etc. In many cases only one particular aspect may be of interest, so t can be used as a generic label to index changes as per Fig. 6.16.

6.6.2 Symmetrized harmonics

Symmetrized (or generalised) harmonics, which essentially provide correctly symmetrized expansions of spherical harmonics (Y_{LM}) functions for a given irreducible representation, Γ , of the molecular point-group can be defined by linear combinations of spherical harmonics [118, 119, 120]:

$$X_{hl}^{\Gamma\mu*}(\theta, \phi) = \sum_{\lambda} b_{hl\lambda}^{\Gamma\mu} Y_{l,\lambda}(\theta, \phi) \quad (6.37)$$

where:

- Γ is an irreducible representation;
- (l, λ) define the usual spherical harmonic indices (rank, order), but note the use of (l, λ) by convention, since these harmonics are usually referenced to the **MF**;
- $b_{hl\lambda}^{\Gamma\mu}$ are symmetrization coefficients;
- index μ allows for indexing of degenerate components (note here the unfortunate convention that the label μ is also used for photon projection terms in general, as per Sect. 6.3.2 - in ambiguous cases the symmetrization term will instead be labelled herein as μ_X , although in many cases may actually be redundant and safely dropped from the symmetrization coefficients);
- h indexes cases where multiple components are required with all other quantum numbers identical.

Analogously to Eq. (6.37), a general expansion of an observable in the symmetrized harmonic basis set can then be defined as:

$$\bar{I}^{\Gamma}(\epsilon, t, \theta, \phi) = \sum_{\Gamma\mu h} \bar{\beta}_{hl}^{\Gamma\mu}(\epsilon, t) X_{hl}^{\Gamma\mu*}(\theta, \phi) \quad (6.38)$$

Alternatively, by substitution into Eq. (6.37), and assigning $l = L$ and $\lambda = M$, a general symmetrized expansion may also be defined as:

$$\bar{I}(\epsilon, t, \theta, \phi) = \sum_{\Gamma\mu h} \sum_{L=0}^{2n} \sum_{M=-L}^L b_{hLM}^{\Gamma\mu} \bar{\beta}_{L,M}(\epsilon, t) Y_{L,M}(\theta, \phi) \quad (6.39)$$

However, in many cases the symmetrization coefficients are subsumed into the $\beta_{L,M}$ terms (or underlying matrix elements); in this case a simplified symmetrized expansion can be defined as:

$$\bar{I}^{\Gamma}(\epsilon, t, \theta, \phi) = \sum_{L=0}^{2n} \sum_{M=-L}^L \bar{\beta}_{L,M}^{\Gamma}(\epsilon, t) Y_{L,M}(\theta, \phi) \quad (6.39)$$

Where the expansion is defined for a given symmetry and irreducible representation with the shorthand Γ ; in many systems a single label may be sufficient here, since allowed (L, M) terms will be defined uniquely by irreducible representation, although multiple quantum numbers may be required for unique definition in the most general cases as per Eq. (6.37) (e.g. for cases with degenerate components). Further details and usage in relation to channel functions are also discussed in Sect. 6.3 (see, in particular, Eq. (6.13) for a similar general case), and in relation to fitting for specific cases in Part II.

The exact form of these coefficients will depend on the point-group of the system, see, e.g. Refs. [120, 121]. Numerical routines for the generation of symmetrized harmonics are implemented in **Photoelectron Metrology Toolkit** [5]: point-groups, character table generation and symmetrization (computing $b_{hl\lambda}^{\Gamma\mu}$ parameters) is handled by `libmsym` [64, 65]; additional handling also makes use of `pySHTools` [60, 61, 62, 63].

A brief numerical example is given below, and more details can be found in the PEMtk documentation [20]. In this case, full tabulations of the parameters list all $b_{hLM}^{1\mu}$ for each irreducible representation, and the corresponding PADs are illustrated in Fig. 6.19.

Note: Full tabulations of the parameters available in HTML or notebook formats only.

```
# Import class
from pemtk.sym.symHarm import symHarm

# Compute hamronics for Td, lmax=4
sym = 'Td'
lmax=6

symObj = symHarm(sym,lmax)

# Character tables can be displayed - this will render directly in a notebook.
symObj.printCharacterTable()
```

Character	dim	E	C2^1	S4^1	σd	C3^1
A1	1	1.0	1.0	1.0	1.0	1.0
A2	1	1.0	1.0	-1.0	-1.0	1.0
E	2	2.0	2.0	0.0	0.0	-1.0
T1	3	3.0	-1.0	1.0	-1.0	0.0
T2	3	3.0	-1.0	-1.0	1.0	0.0

Fig. 6.18: Example character table for Td symmetry generated with the Photoelectron Metrology Toolkit [5] wrapper for libmsym [64, 65].

```
# The full set of expansion parameters can be tabulated

# pd.set_option('display.max_rows', 100)

symObj.displayXlm() # Display values (note this defaults to REAL harmonics)
# symObj.displayXlm(YlmType='comp') # Display values for COMPLEX harmonic expansion.
```

```
# To plot using ePSproc/PEMtk class, these values can be converted to ePSproc BLM_
→data type...

# Run conversion - the default is to set the coeffs to the 'BLM' data type
symObj.toePSproc()

# Set to new key in data class
data.data['symHarm'] = {}

for dataType in ['BLM']: #['matE', 'BLM']:
    # Select expansion in complex harmonics
```

(continues on next page)

(continued from previous page)

```

data.data['symHarm'][dataType] = symObj.coeffs[dataType]['b (comp)']
data.data['symHarm'][dataType].attrs = symObj.coeffs[dataType].attrs

# Plot full harmonics expansions, plots by symmetry
# Note 'squeeze=True' to force drop of singleton dims may be required.
# data.padPlot(keys='symHarm', dataType='BLM', facetDims = ['Cont'], squeeze = True, ↴
#               backend=plotBackend)

# As above, with some additional layout options
rc = [2,3] # Explicit layout setting
data.padPlot(keys='symHarm', dataType='BLM', facetDims = ['Cont'], squeeze = True, ↴
            backend=plotBackend,
            rc = rc, plotFlag=False, returnFlag=True)
figObj = data.data['symHarm']['plots']['BLM']['polar'][0]

# And GLUE for display later with caption
gluePlotly("symHarmPADs", figObj)

```

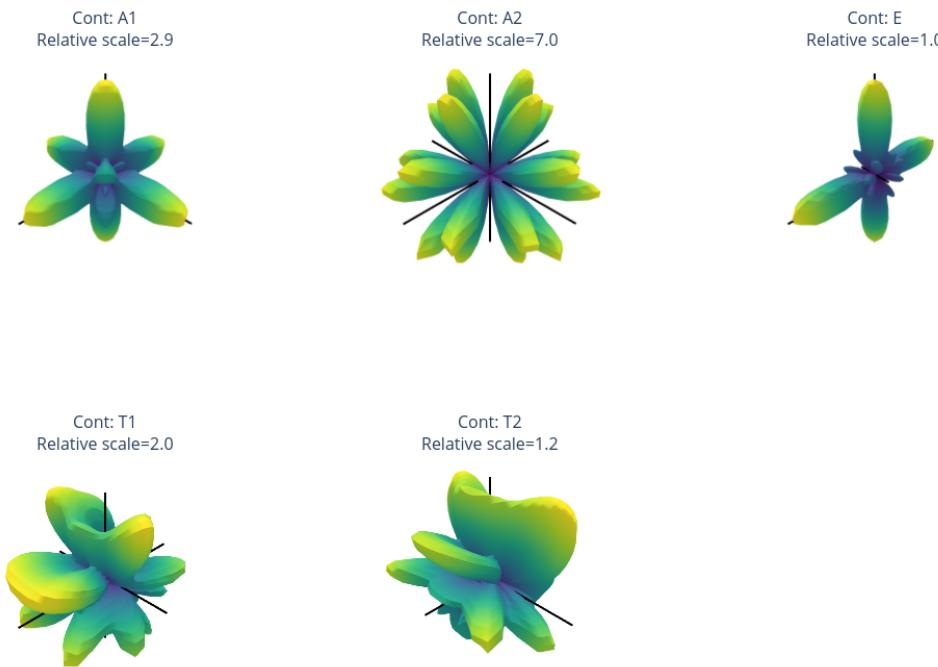


Fig. 6.19: Examples of angular distributions from expansions in symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta, \phi)$, for all irreducible representations in Td symmetry ($l_{max}=6$). (Note A_2 only has components for $l \geq 6$.)

Real & complex forms

By convention, the complex form of the spherical harmonics are usually used for photoionization problems. However, real harmonics are also in common use (and have already appeared in the numerical routines above). The relationships can be defined as (per the [Wikipaedia definitions \[122\]](#)):

$$Y_{\ell m} = \begin{cases} \frac{i}{\sqrt{2}} (Y_\ell^m - (-1)^m Y_\ell^{-m}) & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \frac{1}{\sqrt{2}} (Y_\ell^{-m} + (-1)^m Y_\ell^m) & \text{if } m > 0. \end{cases}$$

$$= \begin{cases} \frac{i}{\sqrt{2}} (Y_\ell^{-|m|} - (-1)^m Y_\ell^{|m|}) & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \frac{1}{\sqrt{2}} (Y_\ell^{-|m|} + (-1)^m Y_\ell^{|m|}) & \text{if } m > 0. \end{cases} \quad (6.39)$$

$$= \begin{cases} \sqrt{2} (-1)^m \Im[Y_\ell^{|m|}] & \text{if } m < 0 \\ Y_\ell^0 & \text{if } m = 0 \\ \sqrt{2} (-1)^m \Re[Y_\ell^m] & \text{if } m > 0. \end{cases}$$

Where the notation here uses $Y_{\ell m}$ for real harmonics, and Y_ℓ^m for complex.

Conversion between types is handled in the `ePSproc` codebase [33, 34, 35] either directly or via the `pySHtools` [60, 61, 62, 63] library, and objects usually have the type specified in their metadata (if missing, they are assumed to be complex). The symmetry routines outlined above automatically compute both types, and these are available in the output data structure (see the `PEMtk` documentation [20] for further details and examples, and the “Working with real spherical harmonics” note from the `ePSproc` documentation [35], and the relevant `pySHtools` documentation pages).

```
# Display complex values
symObj.displayXlm(YlmType='comp')      # Display values for COMPLEX harmonic expansion.
```

```
# Access complex values from SH tools objects
# SHtools object are stored in nested dicts by character and type
print(symObj.coeffs['SH']['A1'].keys())
print(symObj.coeffs['SH']['A1']['comp'])
```

```
dict_keys(['real', 'comp'])
kind = 'complex'
normalization = '4pi'
cspphase = -1
lmax = 6
error_kind = None
header = None
header2 = None
name = None
units = None
```

```
# Similarly Xarray forms include both types
symObj.coeffs['XR']
```

6.6.3 Legendre polynomials

Finally, it is of note that Legendre polynomial expansions are also in common use in the description of photoionization observables. These are suitable for cylindrically symmetric cases only, and form a subset of the general spherical harmonic case. Using the same notation as Eq. (6.37), the 1D expansion can be given as:

$$\bar{I}(\epsilon, t, \theta) = \sum_{L=0}^{2n} \bar{\beta}_L(\epsilon, t) P_L(\cos(\theta)) \quad (6.40)$$

Where $P_L(\cos(\theta))$ are Legendre polynomials in $\cos(\theta)$ (equivalently, associated Legendre polynomials $P_L^{M=0}(\cos(\theta))$). Note that, since the normalisation is different, care must be taken when comparing associated anisotropy parameters between Legendre polynomial and spherical harmonic expansions. Specifically:

$$\beta_{L,0}^{Sph} = \sqrt{(2L+1)/4\pi} \beta_L^{Lg} \quad (6.41)$$

Where *Sph* and *Lg* labels have been added to make explicit that the expansion parameters in the spherical harmonic and Legendre polynomial basis sets respectively.

Herein only spherical harmonic expansions are used, but the ePSproc codebase [33, 34, 35] does include a conversion routine to convert expansion parameters as required. Again more information can be found in the ePSproc documentation [35], particularly the “Working with spherical harmonics” notebook.

```
# Set example BLMs and convert to Lg basis
# Note 'renorm=True' setting to renorm by B0 (will affect abs value of B0, but not ↴
# form of distribution)
# Note also 'harmonics' and 'normType' specifications in output data.

BLMsph = setBLMs([[0,0,1],[1,0,0.5],[2,0,0.8]]).squeeze(drop=True)
BLMlg = ep.util.conversion.conv_BL_BLM(BLMsph, to = 'lg', renorm = True)
BLMlg
```

6.6.4 Time-resolved data

In general, the datasets from a time-resolved measurement can be expressed as some form of *anisotropy parameters*, as outlined above. In a similar manner to the *ADMs* detailed in Sect. 6.5, the $\beta_{L,M}(t)$ can be plotted directly, or expanded as distributions $I(\epsilon, t, \theta, \phi\dots)$. A brief demonstration is given below, making use of the N_2 dataset explored in the case study in Chpt. 11, for further details of the computational and plotting tools see *Part II*, and the ePSproc documentation [35] and PEMtk documentation [20].

```
# Configure settings for case study
# %run '../scripts/setup_notebook.py'

# Fitting setup including data generation and parameter creation
# NOTE this assumes relevant data is available in ../part2/n2fitting.
# See Part 2 for more details and data sources
fitSystem = 'N2'
dataName = 'n2fitting'

# Set datapath,
dataPath = Path(Path.cwd().parent, 'part2', dataName)

# Run general config script with dataPath set above
%run "../scripts/setup_fit_case-studies_270723.py" -d {dataPath} -c {fitSystem}
```

```
# Compute AFBLMs for all matrix elements (energies) for selected orbital
# (Note the main fitting script only computes for a single E-point.)
# Use ADMs from fitting subset
orbKey = 'orb5'
data.AFBLM(keys = orbKey, AKQS = data.data['subset']['ADM'],
           selDims = {'Type': 'L'}, thres=1e-2)
```

Calculating AF-BLMs for job key: orb5

In this case, making use of *ab initio radial matrix elements*, a full set of $\beta_{L,M}(\epsilon, t)$ are computed, for the ionizing channel(s) defined (in this case as labelled by the `orbKey` parameter). These can be plotted vs. energy or time as line-plots or colourmaps. The Photoelectron Metrology Toolkit [5] currently has some basic routines for some specific cases, or low-level functionality from the base libraries can be used for more control.

```
# Default plot with Holoviews/Bokeh
# Will plot BLM(E,t) data with selectors & sliders for other dimensions
data.BLMplot(keys=orbKey, backend='hv')
```

BLMplot set data and plots to self.plots['BLMplot']

```
:HoloMap [Orb,t]
:NdOverlay [l,m]
:Curve [Eke] (BLM)
```

True

```
# For a full BLM(E,t) surface, the output Holoviews dataset can be used.
# See the Holoviews (https://holoviews.org) and HVplot (https://hvplot.holoviz.org)
# docs for more details

# Set opts to match sizes - should be able to link plots or set gridspace to handle
# this?
ep.plot.hvPlotters.setPlotDefaults(fSize = [750,300], imgSize = 600)

# Plot heatmap for l=2 vs. Eke and add ADM plot to layout with hvplot
daLayout = (data.plots['BLMplot']['hvDS'].select(l=[2]).to(hv.HeatMap, kdims=['t', 'Eke'])
            .opts(cmap='coolwarm') +
            data.data['subset']['ADM'].unstack().sel({'K': [2, 4]}).squeeze().real.hvplot.
            line(x='t').overlay('K')).cols(1)
```

```
# Similarly, AFPADs can be plotted.
# Plot I(theta,t) for a single E
# %matplotlib inline
dataKey = 'subset' # Use subset data set for fitting
data.padPlot(keys = dataKey, dataType='AFBLM', selDims={'Labels':'A'},
             Etype = 't', pStyle='grid', reducePhi='sum',
             returnFlag = True)
```

Using default sph betas.
 Summing over dims: set()
 Plotting from self.data[subset][AFBLM], facetDims=['t', None], pType=a with
 backend=mpl.

(continues on next page)

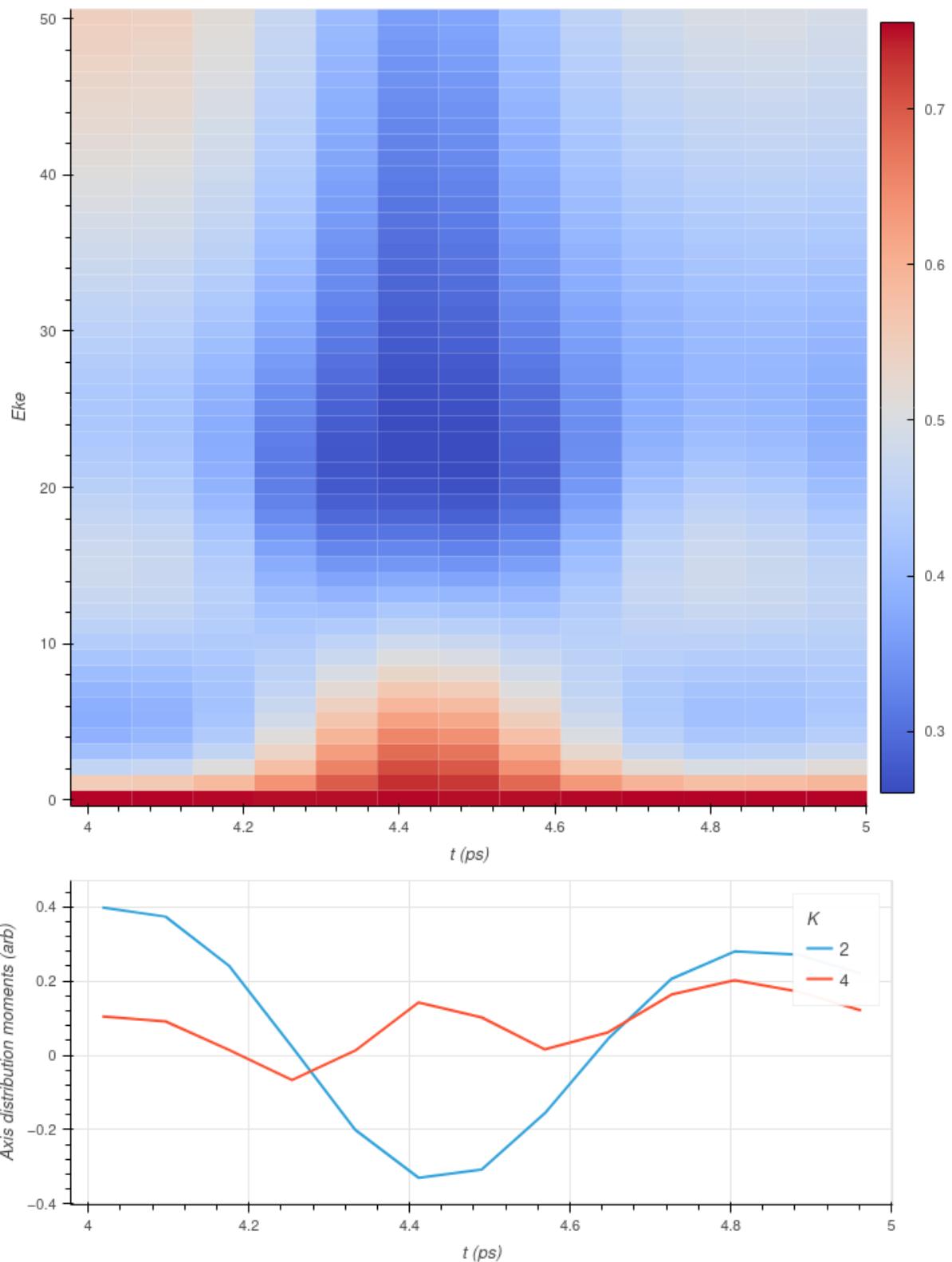
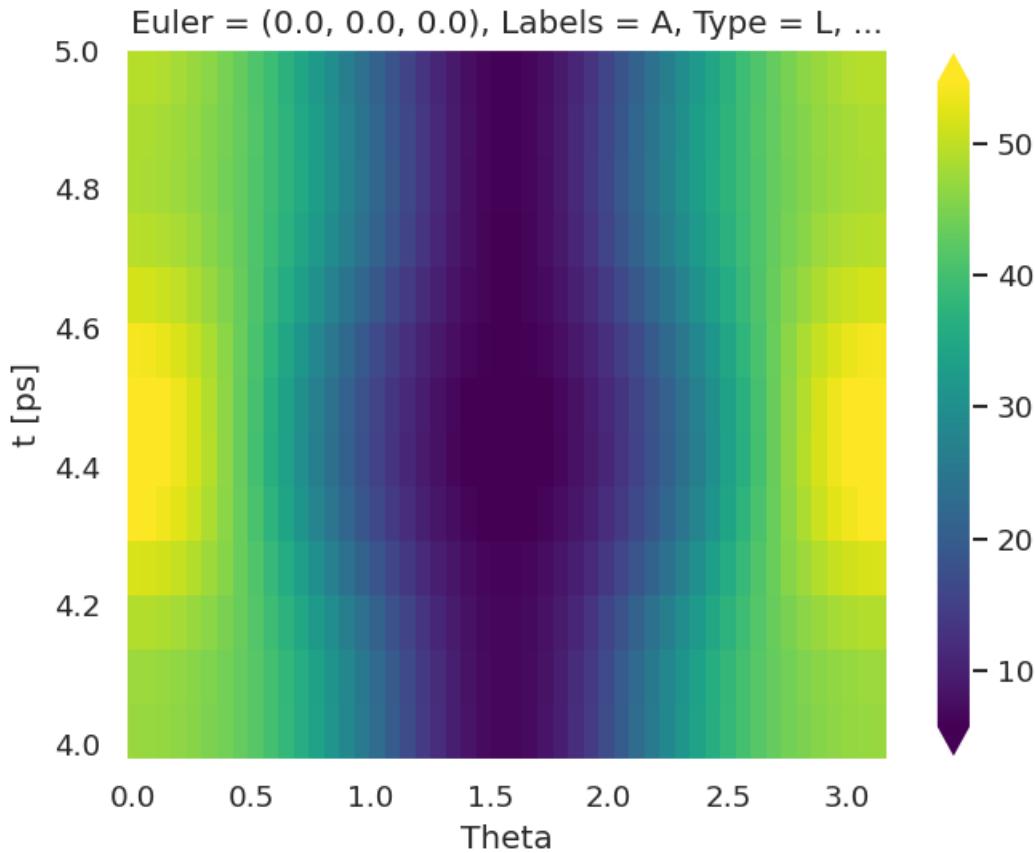


Fig. 6.20: Example $\beta_{L,M}(\epsilon, t)$ data, computed for N_2 (upper panel), and the [ADMs](#) used in the calculation.

(continued from previous page)

```
Grid plot: 3sg-1, dataType: AFBLM, plotType: a
Set plot to self.data['subset']['plots']['AFBLM']['grid']
```



```
# Plot I(theta,phi) at selected (E,t)
data.padPlot(keys = dataKey, dataType='AFBLM', selDims={'Labels':'A'},
            Erange=[4,5,5], Etype = 't', returnFlag = True, backend='pl')

# And GLUE for display later with caption
figObj = data.data[dataKey]['plots']['AFBLM']['polar'][0]
glue("N2AFPADsdemo", figObj)
```

6.7 Information content & sensitivity

A useful tool in considering the possibility of matrix element retrieval is the response, or sensitivity, of the experimental observables to the matrix elements of interest. Aspects of this have already been explored in Sect. 6.3, where consideration of the various geometric tensors (or geometric basis set) provided a route to investigating the coupling - hence sensitivity - of various parameters into product terms. In particular the tensor products discussed in Sect. 6.3.9, including the full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ ((6.18) and (6.19)), can be used to examine the overall sensitivity of a given measurement to the underlying observables. By careful consideration of the problem at hand, experiments may then be tailored for particular cases based on these sensitivities. A related question, is how a given experimental sensitivity might be more readily quantified, and interpreted, for reconstruction problems, in a simpler manner. In general, this can be termed as the *information content* of the measurement(s); an important aspect of such a metric is that it should be readily

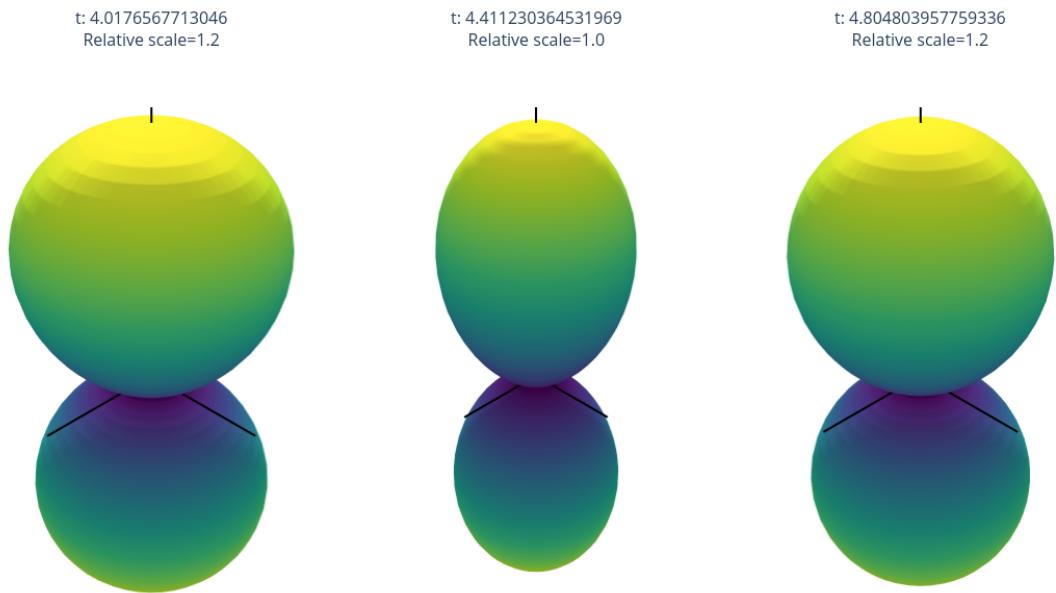


Fig. 6.21: *AF-PADs* example for N_2 at selected (ϵ, t) . In this demo case the alignment and *PADs* are “1D”, and cylindrically symmetric.

interpretable and, ideally, related to whether a reconstruction will be possible in a given case (this has, for example, been considered by other authors for specific cases, e.g. Refs. [112, 123]).

Work in this direction is ongoing, and some thoughts are given below. In particular, the use of the observable $\beta_{L,M}$ presents an experimental route to (roughly) define a form of information content, whilst metrics derived from channel functions or density matrices may present a more rigorous theoretical route to a useful parameterization of information content.

6.7.1 Numerical setup

This follows the setup in Sect. 6.3 *Tensor formulation of photoionization*, using a symmetry-based set of basis functions for demonstration purposes. (Repeated code is hidden in PDF version.)

6.7.2 Experimental information content

As discussed in *Quantum Metrology* Vol. 2 [9], the information content of a single observable might be regarded as simply the number of contributing $\beta_{L,M}$ parameters. In set notation:

$$M = n\{\beta_{L,M}\} \quad (6.42)$$

where M is the information content of the measurement, defined as $n\{\dots\}$ the cardinality (number of elements) of the set of contributing parameters. A set of measurements, made for some experimental variable u , will then have a total information content:

$$M_u = \sum_u n\{\beta_{L,M}^u\}$$

In the case where a single measurement contains multiple $\beta_{L,M}$, e.g. as a function of energy ϵ or time t , the information content will naturally be larger:

$$\begin{aligned} M_{u,\epsilon,t} &= \sum_{u,\epsilon,t} n\{\beta_{L,M}^u(\epsilon,t)\} \\ &= M_u \times M_{\epsilon,t} \end{aligned}$$

where the second line pertains if each measurement has the same native information content, independent of u . It may be that the variable k is continuous (e.g. photoelectron energy), but in practice it will usually be discretized in some fashion by the measurement.

In terms of purely experimental methodologies, a larger M_u clearly defines a richer experimental measurement which explores more of the total measurement space spanned by the full set of $\{\beta_{L,M}^u(k,t)\}$. However, in this basic definition a larger M_u does not necessarily indicate a higher information content for quantum retrieval applications. The reason for this is simply down to the complexity of the problem (cf. Eq. (6.13)), in which many couplings define the sensitivity of the observable to the underlying system properties of interest. In this sense, more measurements, and larger M , may only add redundancy, rather than new information.

From a set of numerical results, it is relatively trivial to investigate some of these properties as a function of various constraints, using standard Python functionality, as shown in the code blocks below. For example, M can be determined numerically as the number of elements in the dataset, the number of *unique* elements, the number of elements within a certain range or above a threshold, and so on.

```
# For the basic case, the data (Xarray object) can be queried,
# and relevant dimensions investigated

print(f"Available dimensions: {BetaNorm.dims}")

# Show BLM dimension details from Xarray dataset
display(BetaNorm.BLM)
```

```
Available dimensions: ('Labels', 't', 'Type', 'it', 'Eke', 'h', 'muX', 'BLM')
```

```
<xarray.DataArray 'BLM' (BLM: 9)>
array([(0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1), (2, -1), (2, 0),
       (2, 1)], dtype=object)
Coordinates:
  * BLM      (BLM) MultiIndex
  - l       (BLM) int64 0 0 0 1 1 1 2 2 2
  - m       (BLM) int64 -1 0 1 -1 0 1 -1 0 1
```

```
# Note, however, that the indexes may not always be physical,
# depending on how the data has been composed and cleaned up.
# For example, the above has l=0, m=+/-1 cases, which are non-physical.

# Clean array to remove terms |m|>l, and display
cleanBLMs(BetaNorm).BLM
```

```
<xarray.DataArray 'BLM' (BLM: 7)>
array([(0, 0), (1, -1), (1, 0), (1, 1), (2, -1), (2, 0), (2, 1)], dtype=object)
Coordinates:
  * BLM      (BLM) MultiIndex
  - l       (BLM) int64 0 1 1 1 2 2 2
  - m       (BLM) int64 0 -1 0 1 -1 0 1
```

```
# Thresholding can also be used to reduce the results
ep.matEleSelector(BetaNorm, thres=1e-4).BLM
```

```
<xarray.DataArray 'BLM' (BLM: 2)>
array([(0, 0), (2, 0)], dtype=object)
Coordinates:
  * BLM      (BLM) MultiIndex
  - l       (BLM) int64 0 2
  - m       (BLM) int64 0 0
```

```
# The index can be returned as a Pandas object, and statistical routines applied...
# For example, nunique() will provide the number of unique values.
```

```
thres=1e-4

print(f"Original array M={BetaNorm.BLM.indexes['BLM'].nunique()}")
print(f"Cleaned array M={cleanBLMs(BetaNorm).BLM.size}")
print(f"Thresholded array (thres={thres}), \
      M={ep.matEleSelector(BetaNorm, thres=thres).BLM.indexes['BLM'].nunique()}")
```

```
Original array M=9
Cleaned array M=7
Thresholded array (thres=0.0001),          M=2
```

For more complicated cases, with $u > 1$, e.g. time-dependent measurements, interrogating the statistics of the observables may also be an interesting avenue to explore. The examples below investigate this for the example “linear ramp” **ADMs** case. Here the statistical analysis is, potentially, a measure of the useful/non-redundant information content, for instance the range or variance in a particular observable can be analysed, as can the number of unique values and so forth.

```
# Convert to PD and tabulate with epsproc functionality
# Note restack along 't' dimension
BetaNormLinearADMsPD, _ = ep.util.multiDimXrToPD(BetaNormLinearADMs.squeeze().real,
                                                 thres=1e-4, colDims='t')

# Basic describe with Pandas,
# see https://pandas.pydata.org/docs/user_guide/basics.html#summarizing-data-describe
# This will give properties per t
BetaNormLinearADMsPD.describe()
```

t	0	1	2	3	4	5	6	7	8	9
count	5.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000
mean	-0.169	-0.074	-0.063	-0.052	-0.041	-0.031	-0.020	-0.009	0.002	0.012
std	0.158	0.126	0.115	0.104	0.093	0.083	0.073	0.065	0.058	0.053
min	-0.282	-0.254	-0.226	-0.198	-0.170	-0.142	-0.114	-0.086	-0.058	-0.038
25%	-0.282	-0.201	-0.179	-0.157	-0.135	-0.113	-0.092	-0.072	-0.052	-0.030
50%	-0.282	-0.002	-0.004	-0.006	-0.008	-0.010	-0.011	-0.009	-0.007	-0.005
75%	-0.045	0.004	0.008	0.011	0.015	0.019	0.023	0.027	0.031	0.034
max	0.045	0.050	0.054	0.059	0.064	0.069	0.075	0.088	0.100	0.113

```
# Basic describe with Pandas,
# see https://pandas.pydata.org/docs/user_guide/basics.html#summarizing-data-describe
# By transposing the input array, this will give properties per BLM
BetaNormLinearADMsPD.T.describe()
```

h	0	1	2	3	4	5	6	7	8	9
1	0	2	0	2	4	6	0	2	4	6
m	0	0	0	0	0	0	0	0	0	0
count	10.000	9.000	10.000	10.000	9.000	9.000	10.000	10.000	9.000	9.000e+00
mean	-0.156	0.063	-0.156	0.066	0.021	0.012	-0.156	-0.029	-0.021	1.669e-03
std	0.085	0.034	0.085	0.014	0.012	0.007	0.085	0.011	0.011	9.140e-04
min	-0.282	0.013	-0.282	0.045	0.004	0.002	-0.282	-0.045	-0.038	3.337e-04
25%	-0.219	0.038	-0.219	0.056	0.013	0.007	-0.219	-0.037	-0.029	1.001e-03
50%	-0.156	0.063	-0.156	0.066	0.021	0.012	-0.156	-0.029	-0.021	1.669e-03
75%	-0.093	0.088	-0.093	0.077	0.030	0.017	-0.093	-0.021	-0.013	2.336e-03
max	-0.030	0.113	-0.030	0.088	0.038	0.022	-0.030	-0.013	-0.004	3.004e-03

For further insight and control, specific aggregation functions and criteria can be specified. For instance, it may be interesting to look at the number of unique values to a certain precision (e.g. depending on experimental uncertainties), or consider deviation of values from the mean.

```
# Round values to 2 d.p., then apply statistical methods
ndp = 2
BetaNormLinearADMsPD.round(ndp).agg(['min', 'max', 'var', 'count', 'nunique']).round(3)
```

t	0	1	2	3	4	5	6	7	8	9
min	-0.280	-0.250	-0.230	-0.200	-0.170	-0.140	-0.110	-0.090	-0.060	-0.040
max	0.040	0.050	0.050	0.060	0.060	0.070	0.080	0.090	0.100	0.110
var	0.024	0.015	0.014	0.011	0.009	0.007	0.005	0.004	0.003	0.003
count	5.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000	10.000
nunique	3.000	5.000	7.000	7.000	8.000	8.000	8.000	8.000	8.000	8.000

```
# Define demean function and apply (from https://stackoverflow.com/a/26110278)
demean = lambda x: x - x.mean()

# Compute differences from mean
BetaNormLinearADMsPD.transform(demean, axis='columns')
```

		t	0	1	2	3	4	5	6	7	8	9
h	l	m										
0	0	0	-0.126	-0.098	-0.070	-4.205e-02	-1.402e-02	1.402e-02	4.205e-02	7.009e-02	0.098	0.126
	2	0	NaN	-0.050	-0.038	-2.508e-02	-1.254e-02	0.000e+00	1.254e-02	2.508e-02	0.038	0.050
1	0	0	-0.126	-0.098	-0.070	-4.205e-02	-1.402e-02	1.402e-02	4.205e-02	7.009e-02	0.098	0.126
	2	0	-0.022	-0.017	-0.012	-7.172e-03	-2.391e-03	2.391e-03	7.172e-03	1.195e-02	0.017	0.022
4	0	0	NaN	-0.017	-0.013	-8.537e-03	-4.269e-03	3.469e-18	4.269e-03	8.537e-03	0.013	0.017
	6	0	NaN	-0.010	-0.007	-4.912e-03	-2.456e-03	-3.469e-18	2.456e-03	4.912e-03	0.007	0.010
2	0	0	-0.126	-0.098	-0.070	-4.205e-02	-1.402e-02	1.402e-02	4.205e-02	7.009e-02	0.098	0.126
	2	0	-0.016	-0.013	-0.009	-5.366e-03	-1.789e-03	1.789e-03	5.366e-03	8.943e-03	0.013	0.016
4	0	0	NaN	0.017	0.013	8.364e-03	4.182e-03	0.000e+00	-4.182e-03	-8.364e-03	-0.013	-0.017
	6	0	NaN	-0.001	-0.001	-6.675e-04	-3.337e-04	-1.735e-18	3.337e-04	6.675e-04	0.001	0.001

```
# Apply statistical functions to differences from mean.
BetaNormLinearADMsPD.transform(demean, axis='columns') . \
    round(ndp).agg(['min', 'max', 'var', 'count', 'nunique']).round(3)
```

t	0	1	2	3	4	5	6	7	8	9
min	-0.130	-0.100	-0.070	-0.04	-0.01	-0.00	0.00	-0.010	-0.010	-0.020
max	-0.020	0.020	0.010	0.01	-0.00	0.01	0.04	0.070	0.100	0.130
var	0.004	0.002	0.001	0.00	0.00	0.00	0.00	0.001	0.002	0.003
count	5.000	10.000	10.000	10.00	10.00	10.00	10.00	10.000	10.000	10.000
nunique	2.000	6.000	5.000	5.00	2.00	2.00	3.00	5.000	6.000	6.000

In this case the analysis suggests that $t = 2 - 5$ contain minimal information (low variance), and $t = 4, 5$ potentially redundant information (low nunique), whilst $t = 1, 7 - 9$ show a greater total information content and number of unique values. However, this analysis is not necessarily absolutely definitive, since some nuances may be lost in this basic statistical analysis, particularly for weaker channels.

For a more detailed analysis, other standard analysis tools can be deployed. For instance, the covariance matrix can be investigated, given by $K_{i,j} = \text{cov}[X_i, X_j] = \langle (X_i - \langle X_i \rangle)(X_j - \langle X_j \rangle) \rangle$. For the linear ramp case this analysis is not particularly useful, but will become more informative for more complicated cases.

```
# Compute covariance matrix with Pandas
# Note this is the pairwise covariance of the columns,
# see https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.cov.
#html
covMat = BetaNormLinearADMsPD.cov()

# Plot with holoviews
figObj = covMat.hvplot.heatmap(cmap='viridis')
```

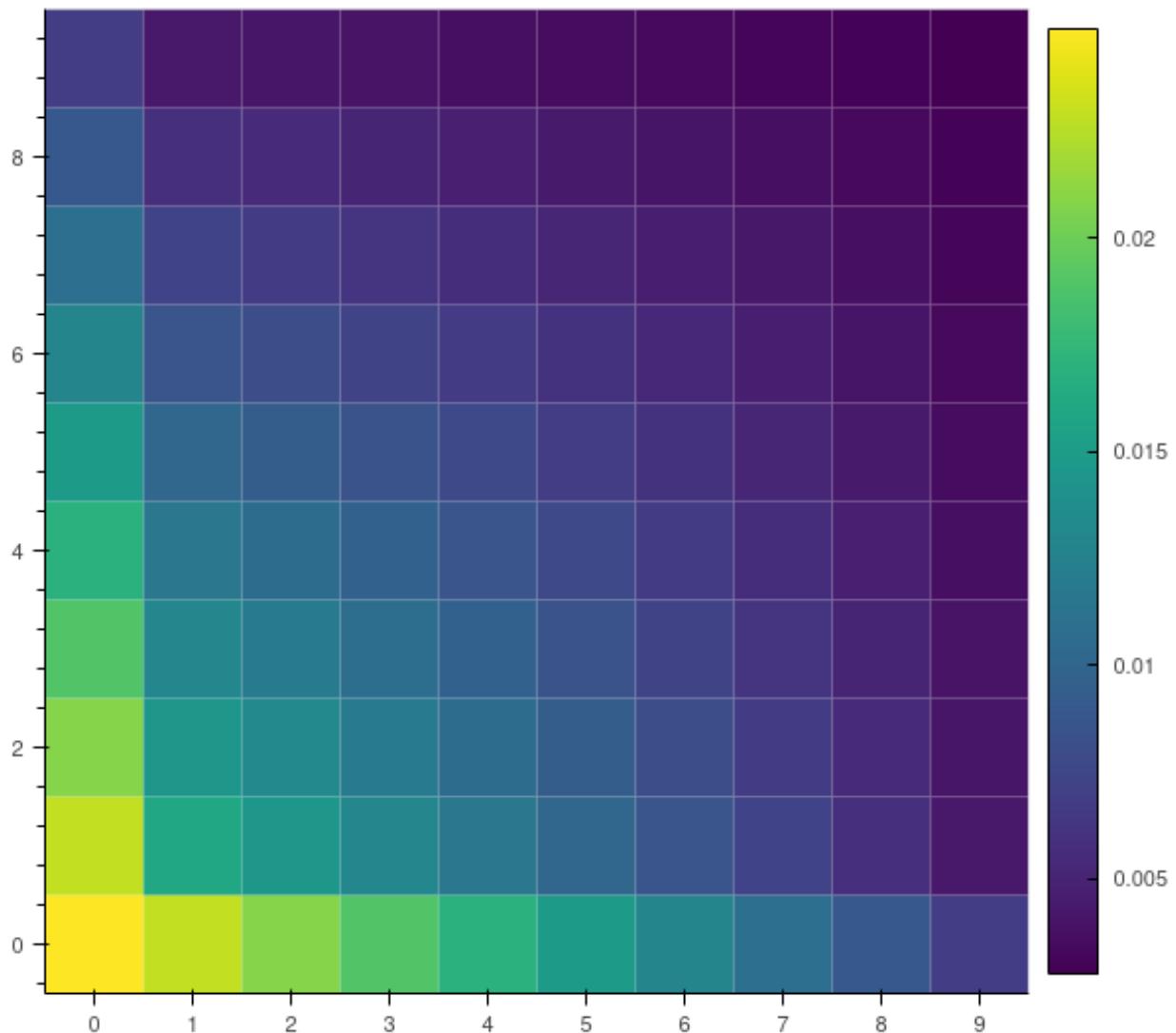


Fig. 6.22: Example $\beta_{L,M}(t)$ covariance matrix, see text for details.

6.7.3 Information content from channel functions

A more complete accounting of information content would, therefore, also include the channel couplings, i.e. sensitivity/dependence of the observable to a given system property, in some manner. For the case of a time-dependent measurement, arising from a rotational wavepacket, this can be written as:

$$M_u = n\{\Upsilon_{L,M}^u(\epsilon, t)\}$$

In this case, each (ϵ, t) is treated as an independent measurement with unique information content, although there may be redundancy as a function of t depending on the nature of the rotational wavepacket and channel functions.

(Note this is in distinction to previously demonstrated cases where the time-dependence was created from a shaped laser-field, and was integrated over in the measurements, which provided a coherently-multiplexed case, see refs. [124, 125, 126] for details.)

In the numerical examples below, this is considered in terms of the full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ as defined in (6.18) and (6.19) (see Sect. 6.3.9). Numerically, the routines follow from those already introduced above for exploring the information content of $\beta_{L,M}$ terms, with the caveat that there are more dimensions to handle in the channel functions, indexed by the relevant set of quantum numbers $\{\zeta, \zeta'\}$ - these can be included in the criteria for determination of M , or selected or summed over as desired.

```
# Define a set of channel functions to test
channelFuncs = (basisProductLinearADMs['BLMtableResort'] * basisProductLinearADMs[
    'polProd'])

# For illustrative purposes, define a subset to use for analysis
channelFuncsSubset = channelFuncs.sel(Labels='A').sel({'S-Rp':0, 'mu':0, 'mup':0})  #.
    .sel(L=2)

# Check dimensions
print(f"Available dimensions: {channelFuncs.dims}")
print(f"Subset dimensions: {channelFuncsSubset.dims}")
```

```
Available dimensions: ('m', 'mp', 'S-Rp', 'l', 'lp', 'L', 'mu', 'mup', 'Labels', 'M',
    't')
Subset dimensions: ('m', 'mp', 'l', 'lp', 'L', 'M', 't')
```

Note: Full tabulations of the parameters available in HTML or notebook formats only.

```
# Convert to PD and tabulate with epsproc functionality
# Note restack along 't' dimension
channelFuncsSubsetPD, _ = ep.util.multiDimXrToPD(channelFuncsSubset.squeeze().real,
    thres=1e-4, colDims='t')

# Round values to 1 d.p., then apply statistical methods
# Compute per basis index and display
channelFuncsSubsetPD.T.round(2).agg(['min', 'max', 'var', 'count', 'nunique']).T
```

For the higher-dimensional case, it is useful to plot terms relative to all quantum numbers. For example, in a similar manner to the basis set explorations of Sect. 6.3.9, related properties such as the distance from the mean can be examined

with `lmPlot()`. And, as previously demonstrated, other properties, such as the covariance, may be examined and plotted.

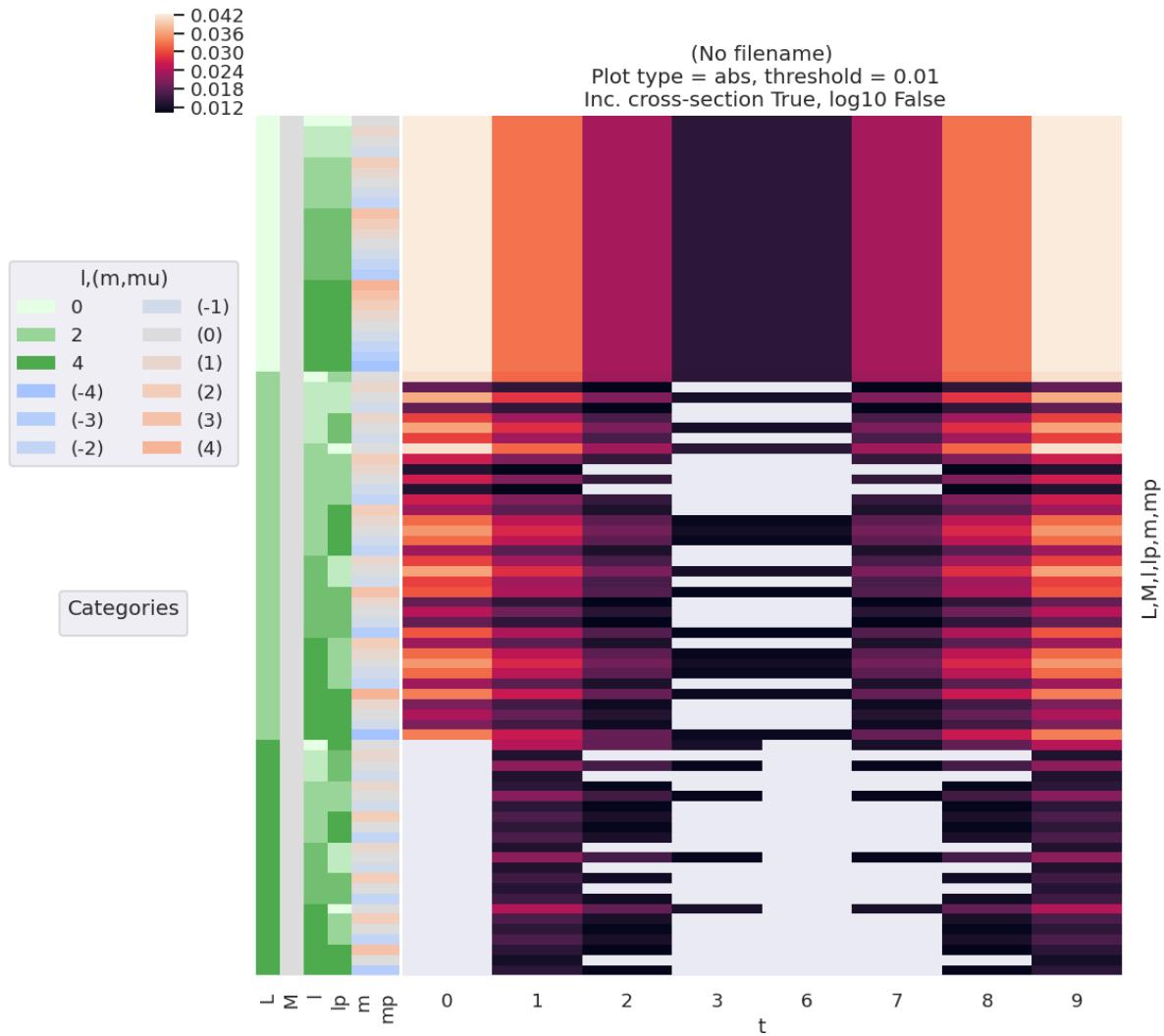
```
# channelFuncsSubsetPD.transform(demean, axis='columns')
# cmap=None    # cmap = None for default. 'vlag' good?
# cmap = 'vlag'

# De-meaned channel functions
channelFuncsDemean = channelFuncsSubsetPD.transform(demean, axis='columns')

# Plot using lmPlot routine - note this requires conversion to Xarray data type first.
daPlot, daPlotpd, legendList, gFig = ep.lmPlot(channelFuncsDemean.to_xarray().to_
       array('t'))
       , xDim='t', cmap=cmap, mDimLabel='m');
```

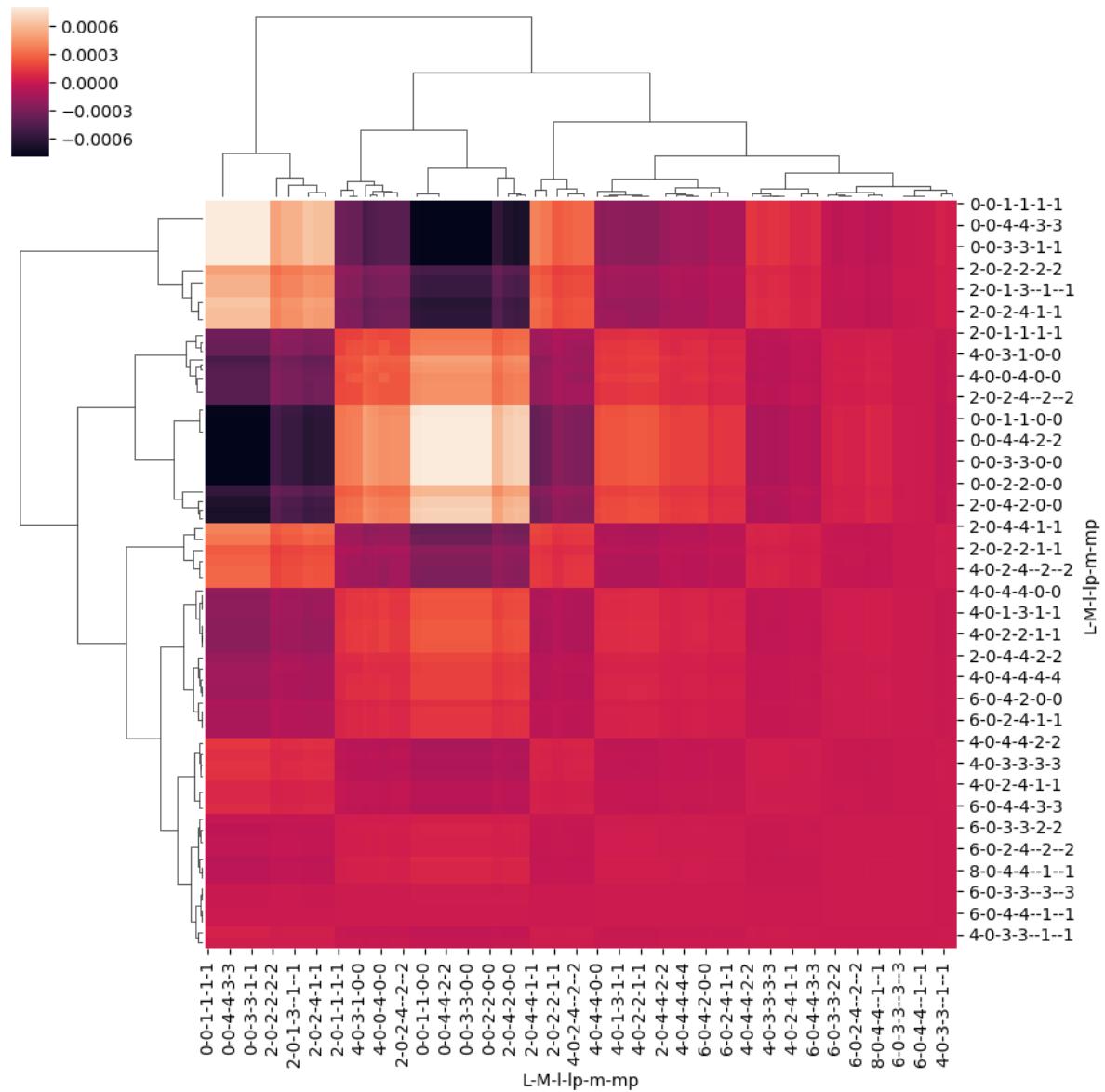
```
Set dataType (No dataType)
Plotting data (No filename), pType=a, thres=0.01, with Seaborn
```

```
No artists with labels found to put in legend. Note that artists whose label
  ↵start with an underscore are ignored when legend() is called with no argument.
```



```
# Full covariance mapping along all dims
%matplotlib inline
sns.clustermap(channelFuncsSubsetPD.T.cov().fillna(0)) #.round(3)
```

<seaborn.matrix.ClusterGrid at 0x7ff1512359c0>



NUMERICAL METHODOLOGIES FOR EXTRACTING MATRIX ELEMENTS

Following the tensor notation outline in Sect. 6.3, the complete quantum metrology of a photoionization event (a.k.a. a “complete” photoionization experiment) can be characterized as recovery of the matrix elements $I^{\zeta}(\epsilon)$ (per Eqs. (6.13), (6.14)) from the experimental measurements or, equivalently, the density matrix $\hat{\rho}^{\zeta\zeta'}$ (Eqs. (6.31) - (6.33)). In general, such a recovery or reconstruction may be possible provided the *channel functions* are known, and the information content of the measurements is sufficient; it is, however, also possible that restrictions in any given case may preclude reconstruction, or restrict the level of recovery possible (e.g. to a lower symmetry group, or certain subsets of *radial matrix elements*) or fidelity of such a reconstruction. (For further discussion and background, see Refs. [92, 94] and *Quantum Metrology* Vol. 1 [4].)

Of particular import for *radial matrix elements* retrieval is the phase-sensitive nature of the observables (Sect. 6.6), which is required in order to obtain phase information on the *partial-waves*. In this context, *PADs* can also be considered as *angular interferograms*, and reconstruction can be considered conceptually similar to other phase-retrieval problems, e.g. optical field recovery with techniques such as FROG [127], and general quantum tomography [100].

As introduced previously (Sect. 4.1.2), the focus herein is the development and testing of the *generalised bootstrap retrieval protocol*, based on time-resolved *RWP* photoionization experiments. A general outline of the simplest 2-stage version of this protocol is shown in Fig. 7.1. In this scheme, the *channel functions* are assumed to be known, and the *ADMs* assumed to be accurately computable: these are, in general, required to determine the *radial matrix elements* in this protocol.

An extended outline comparing some similar approaches is shown in Fig. 7.2; of particular note here is the possibility for retrieval directly from *MF* measurements. Alternative, but conceptually similar, protocols involving different control parameters (as distinct from the *RWP* and *MF* cases), may also be also be used, see *Quantum Metrology* Vols. 1 & 2 [4, 9] for examples. For protocols making use of control methods, the key requirement is for the contribution of the control parameters to the observable, and associated coupling to the *channel functions* to be accurately accounted for. In general, these contributions may be computed and/or obtained from experiment depending on the scheme used. One advantage of the *RWP* case is that the *RWP* can be accurately computed, and the determination of the corresponding molecular alignment (*ADMs*) from an experiment can be treated as a reduced-dimensionality linear signal retrieval problem. As indicated in Fig. 7.1, this stage is separable, and forms *level 1* of the *bootstrap retrieval protocol*. In this procedure sets of computed *ADMs* form the basis set for the fitting (as a function of laser fluence and rotational temperature), allowing the accurate determination of the experimentally-achieved alignment; this is discussed further in Sect. 7.1.1. *Level 2* involves non-linear data fitting, making use of the *ADMs* and the *channel functions*, in order to compute observables, and obtaining the *radial matrix elements* as the fitted parameters; this is discussed further in Sect. 7.1.2.¹

¹ As noted elsewhere: here the *radial matrix elements* are assumed to be time-independent, although that may not be the case for the most complicated examples including vibronic dynamics, see *Quantum Metrology* Vol. 2 [9] for further discussion on this point.

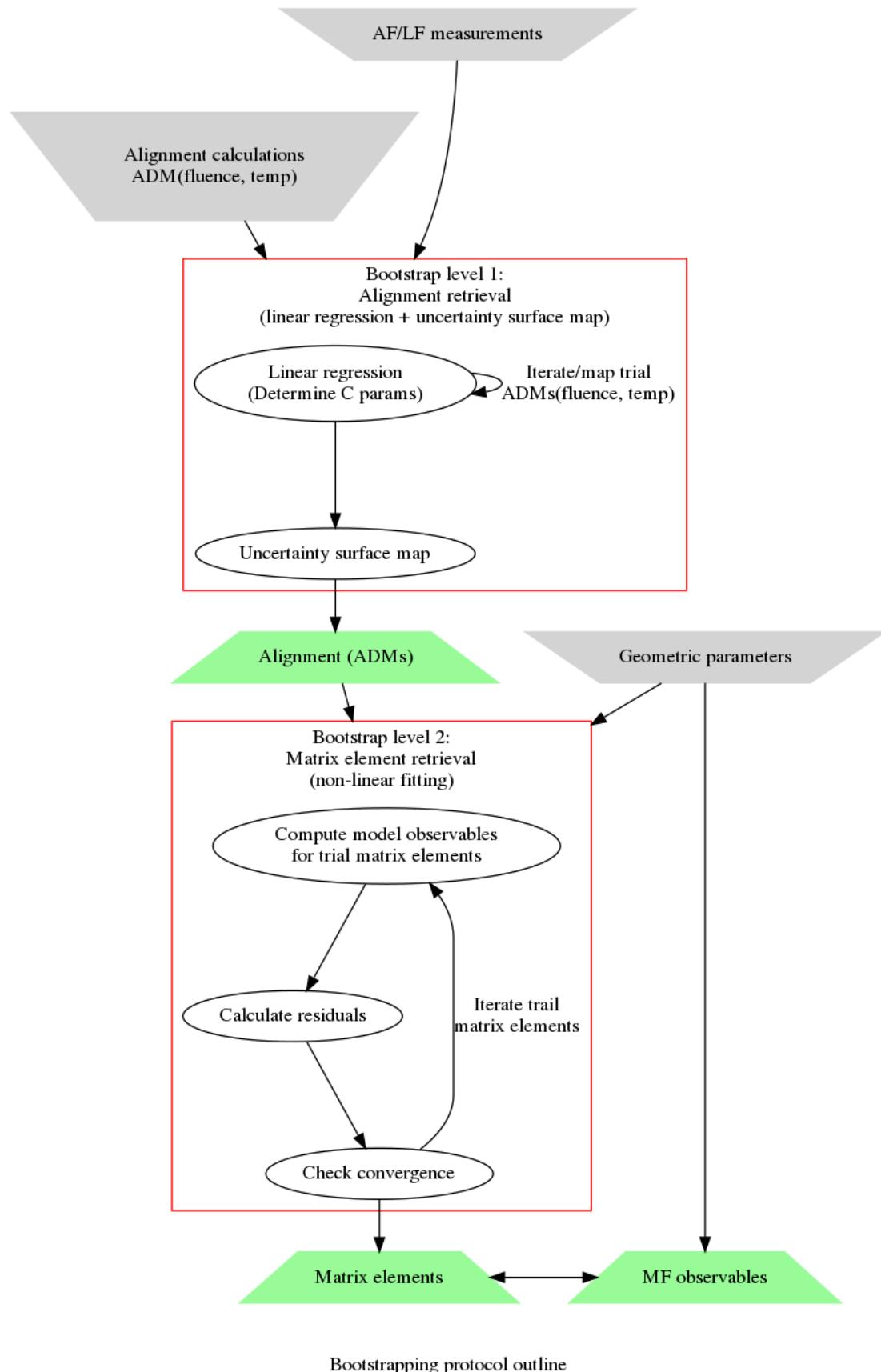


Fig. 7.1: Outline of the 2-stage generalised bootstrap *radial matrix elements* retrieval protocol (Sect. 4.1.2). In this case, level 1 outlines the determination of *ADMs*, and level 2 bootstraps from this to recover *radial matrix elements* and *MF observables*. Grey inverted trapezoids indicate required inputs to the protocol, green trapezoids indicate the retrieved quantities.



Fig. 7.2: Comparison of similar *radial matrix elements* retrieval protocols, illustrating (left) pure *MF* case; (middle) generalised bootstrap protocol; (right) matrix inversion protocol (see Ref. [98]). Grey inverted trapezoids indicate required inputs to the protocol, green trapezoids indicate the retrieved quantities. For the retrieval from the *MF* measurements case, no *RWP* and associated *ADMs* are required. For the matrix inversion protocol, *MF* observables are recovered, but not *radial matrix elements*, although the latter may be possible by subsequent analysis of the *MF-PADs*.

7.1 Fitting methodologies

In general, the extraction of parameters from a data set can be viewed as a general minimization (fitting) problem. This type of treatment is versatile, and can be multi-stage depending on the complexity of the problem. For the generalised bootstrapping method, the treatment of photoionization data is split into two types of fit, as shown in Fig. 7.1. Firstly, a linear fitting stage to retrieve the molecular axis distribution, characterised by a set of *ADMs* (see Sect. 6.3.8 for details); secondly, a non-linear fitting stage to retrieve the complex-valued matrix elements.

In terms of the data, the 1st stage can be written as:

$$\bar{\beta}_{L,M}^u(\epsilon, t) = \sum_{K,Q,S} A_{Q,S}^K(t) \bar{C}_{KQS}^{LM}(\epsilon) \quad (7.1)$$

And the 2nd stage as per Eqs. (6.13) and (6.17) for the AF case:

$$\bar{\beta}_{L,M}^u(\epsilon, t) = \sum_{\zeta, \zeta'} \bar{Y}_{L,M}^{u,\zeta\zeta'}(t) \mathbb{I}^{\zeta\zeta'}(\epsilon)$$

In these forms, the terms are:

1. $A_{Q,S}^K(t)$, the set of *ADMs* defining the molecular alignment, and associated parameters $\bar{C}_{KQS}^{LM}(\epsilon)$.
2. $\bar{Y}_{L,M}^{u,\zeta\zeta'}(t)$, the channel functions in the *AF* (Eq. (6.17), and matrix elements $\mathbb{I}^{\zeta\zeta'}(\epsilon)$.

Hence stage (2) relies on the inputs of stage (1), i.e. the *ADMs*; and the parameters in stage (1) can be determined via fitting the data (linear regression) making use of computed sets of $A_{Q,S}^K(t)$ as a function of experimental parameters (laser fluence and rotational temperature). In this case, a range of *ADMs* (“basis sets”) are computed, and the best match to the experimental data chosen - more details are discussed in Sect. 7.1.1. In a similar manner, the 2nd stage makes use of a known basis set - the *channel functions* - but a non-linear fit is required to determine the set of matrix elements, see Sect. 7.1.2.

Finally, it is also of note that, although the case herein focusses on rotational wavepackets as a control parameter, the same general approach can be applied to other cases, e.g. fitting *MF-PADs* directly (for which only the 2nd stage is required), fitting *PADs* obtained via rotational state-resolved transitions, with shaped laser pulses and so on, as detailed in *Quantum Metrology* Vols. 1 & 2 [4, 9]. Although only rotational wavepacket cases are illustrated in this work (see Part II), by suitable choice of dataset and channel functions many other experimental schemes may be modelled and analysed; the Photoelectron Metrology Toolkit [5] is designed with this flexibility in mind.

7.1.1 Computation and linear fitting for alignment characterisation

Efforts to align and orient molecules in recent decades have led to detailed studies of the rotational dynamics of molecules after interaction with a non-resonant femtosecond laser pulse. A significant outcome of these studies has been the development of a reliable model capable of accurate simulations of rotational wavepacket dynamics that quantitatively agree with experimental results. By measurement of a signal from a time evolving rotational wavepacket, this ability to accurately simulate the wavepacket dynamics can be used to reconstruct the measured signal in the *MF*. Since in this case the time resolved measurement constitutes a set of measurements of the same quantity from a variety of molecular axes distributions (sets of *ADMs*), it is reasonable to conclude that if the axes distributions are known, and provided a large enough space of orientations is explored by the molecule over the experimental time window, the *MF* signal should be extractable.

This is relatively straight forward for a signal that is a single number (scalar) in the *MF* for a given polarization of the light, such as the photoionization yield. Such a signal may, in general, be expressed as an expansion,

$$S(\theta, \chi) = \sum_{jk} C_{jk} D_{0k}^j(\theta, \chi), \quad (7.2)$$

where θ and χ are the *MF* spherical polar and azimuthal angles of the linearly polarized electric field vector generating the signal (Sect. 6.3.3); C_{jk} are unknown expansion coefficients; and D_{0k}^j are the *Wigner rotation matrix elements*, a basis on the space of orientations. A time resolved measurement of S from a rotational wavepacket is the quantum expectation value of this expression,

$$\langle S \rangle(t) = \sum_{jk} C_{jk} \langle D_{0k}^j \rangle(t). \quad (7.3)$$

Since the rotational wavepacket can be accurately simulated, the $\langle D_{0k}^j \rangle(t)$ are considered known. From the measurement of a time resolved signal $\langle S \rangle(t)$, the unknown coefficients C_{jk} can be determined by linear regression, and the molecular frame signal in Eq. (7.2) constructed. In this form the method was initially applied to strong field ionization and dubbed Orientation Reconstruction through Rotational Coherence Spectroscopy (ORRCS) [128, 129]. It has since been applied to strong field ionization of various molecules [130, 131, 132], strong field dissociation [133] and few-photon ionization [134].²

The case of PADs is a more challenging one, since they are not generally described by Eq. (7.2). Instead, both *AF* and *MF-PADs* are determined by the *radial matrix elements*, as discussed in Chpt. 6. However, the correspondence of the problem with an equation of the form of Eq. (7.3) - essentially a convolution - can be made. This is discussed in detail in Ref. [96]. In the current case Eqs. (6.13) and (6.17) can be rewritten in a similar form to Eq. (7.3) by explicitly separating out the *ADMs* $A_{Q,S}^K(t)$ and collapsing all other terms. The case of photoionization from a time-dependent ensemble can then be reparameterized as indicated in Eq. (7.1). Here the set of axis distribution moments can thus be viewed as modulating all observables $\beta_{L,M}^u(t)$. The unknowns, \bar{C}_{KQS}^{LM} and axis distribution moments $A_{Q,S}^K(t)$, can be retrieved in a similar manner to that discussed for the simpler scalar observable case above (Eq. (7.3)), i.e. via linear regression with *RWP*.

In practice this equates to (accurately) simulating *RWPs*, hence obtaining the corresponding $A_{Q,S}^K(t)$ parameters (expectation values), as a function of laser fluence and rotational temperature. Given experimental data, a 2D uncertainty (or error) surface in these two fundamental quantities can then be obtained from a linear regression for each set of $A_{Q,S}^K(t)$. The closest set of parameters to the experimental case is then determined by selection of the best results (smallest uncertainty) from such a parameter-space mapping, which constitutes determination of both the rotational wavepacket (hence $A_{Q,S}^K(t)$) and $\bar{C}_{KQS}^{LM}(\epsilon)$. Optimally, the corresponding physical properties can be cross-checked with other experimental estimates for additional confirmation of the fidelity of the protocol, although this may not always be possible. Note that, in this case, the photoionization dynamics are phenomenologically described by the real parameters \bar{C}_{KQS}^{LM} , but details of the matrix elements are not obtained directly.

At the time of writing, computation of *RWPs* and this stage of the *bootstrap retrieval protocol* analysis is not implemented in the *Photoelectron Metrology Toolkit* [5], although is planned for the near future, and has been demonstrated in practice [1]. The examples in *Part II* instead make use of computed *ADMs* directly, essentially corresponding to the assumption that level 1 of the *bootstrap retrieval protocol* was successful. Given accurate *RWPs*, a successful fit to experimental data is generally assumed to be a given outcome, as this stage of the analysis requires only linear fitting in a two-parameter basis space.

7.1.2 Non-linear fitting for matrix elements

The nature of the photoionization problem suggests that a fitting approach can work, in general, which can be expressed (for example) in the standard way as a (non-linear) least-squares minimization problem:

$$\chi^2(\|\zeta\zeta'\|) = \sum_u \left[\beta_{L,M}^u(\epsilon, t; \|\zeta\zeta'\|) - \beta_{L,M}^u(\epsilon, t) \right]^2 \quad (7.4)$$

where $\beta_{L,M}^u(\epsilon, t; \|\zeta\zeta'\|)$ denotes the values from a model function, computed for a given set of (complex) matrix elements $\|\zeta\zeta'\|$, and $\beta_{L,M}^u(\epsilon, t)$ the experimentally-measured parameters, for a given configuration u . Implicit in the notation is that

² A large range of other experimental methods have also addressed alignment and orientation dependence and retrieval, other recent examples include Coulomb-explosion imaging [135], high-harmonic spectroscopy [136, 137], optical imaging [138] and rotational echo spectroscopy [139], see Refs. [110, 113] for further discussion.

the matrix elements are independent of u (or otherwise averaged over u). Once the matrix elements are obtained in this manner then *MF* observables, for any arbitrary u , can be calculated. Generally fitting routines do not handle complex-valued functions, so the fitting parameter space is usually defined by parameters in magnitude-phase form (Eq. (6.15); see also discussion in Sect. 6.3.1)

Although in principle a very general approach, outstanding questions with such protocols remain, in particular fit uniqueness and reproducibility, the optimal measurement space u - or associated information content M_u - for any given case or measurement schema, how well they will scale to larger problems (more matrix elements/partial waves), the most efficient fitting methodologies/strategies, and so forth. In general, determination of *radial matrix elements* is *not* expected to be trivial, nor to always be successful, due to the complexity of the problem; one significant issue is the topology of the χ^2 hypersurface, which is of $2N - 1$ dimensions (where N is the number of *radial matrix elements*), and may contain many local minima. Exploration of these questions for various exemplar systems is the topic of the *Part II*, and further general discussion can be found in *Quantum Metrology* Vols. 1 & 2 [4, 9], see in particular *Quantum Metrology* Vol. 2 [9] Sect. 8.2.2.

7.1.3 Implementation in PEMtk

As outlined in Sect. 5.3 and Sect. 5.5, Photoelectron Metrology Toolkit [5] uses the `lmfit` library [66, 67] to implement general fitting routines, along with the `ePSproc` codebase [33, 34, 35] for computation of the required basis sets and observables. The latter has already been illustrated in Chpt. 6, and the illustration of the former is the subject of *Part II*. However, in these demonstrations only the *RWP* case is investigated, and analysis routines used only a standard Levenberg-Marquardt least-squares minimization method.

More generally, it is of note that the routines are written to be (somewhat) general and modular, such that other optimization methods may readily be implemented - either via those already supported by `lmfit` library [66, 67] (e.g. Levenberg-Marquardt, basin hopping, Nelder-Mead and so on - see the `lmfit` library [66, 67] documentation for supported methods), or by making use of other fitting libraries and methodologies.

Similarly, modification of the routines to other retrieval schemes should be fairly easy, and usually requires only:

1. a function which computes the required basis set (e.g. *channel functions*)
2. observables for the problem at hand.

Examples are given in *Part II* for the *generalised* bootstrap retrieval protocol, and *MF-PADs* based retrieval is also implemented in the codebase. For further details see the `PEMtk` documentation [20], particularly the *fitting* model backends and *fitting MF and other datasets* pages.

7.2 Fitting strategies

The overall approach to complex non-linear fitting incorporates a number of aspects, broadly “fitting strategies”, which may influence the likelihood of a successful *radial matrix elements* retrieval, and/or the time required to achieve this result:

1. The choice of numerical fitting method (Sect. 7.1.3).
2. The choice of dataset to analyse (and/or the choice of experimental measurement).
3. Additional statistical and/or other meta-analysis.

At the time of writing, work is ongoing in all these areas, and the illustrations in *Part II* herein include further notes regarding limitations or expectations for specific cases. Clearly, there are many choices numerically, and detailed investigation is required to determine the optimal strategy in any given case (this is examined partially in *Part II* in terms of limiting cases by symmetry). Emerging data analysis methods may also be useful here, in particular GPU-based routines and specialist high-dimensional space fitting methods.

One aspect that is intrinsic to these examples, but has not been discussed elsewhere, is the meta-analysis of the retrieved *radial matrix elements*. This is discussed generally in *Quantum Metrology* Vol. 2 [9] Sect. 8.2.2, and implemented numerically in the examples in *Part II* herein. The general aim in this type of analysis is to ascertain whether a given set of retrieved parameters is accurate and unique in a given case. Naturally, for test cases with synthetic data (as in *Part II*), testing against the known input *radial matrix elements* is the best solution and provides the most stringent test for the applicability of the retrieval method - but this is, of course, not generally useful for experimental datasets. Instead, as per previous analysis cases (see *Quantum Metrology* Vol. 2 [9]), a statistical (or “numerical experiment”) approach can be used. In this type of approach, a number of fits n (typically on the order of $10^2\text{--}10^4$) are run independently, with different seed parameters and/or different input data (cf. Monte Carlo methods, and statistical bootstrap methods), and the set of results analysed for consistency and uniqueness over the retrieved parameter sets. As outlined in *Quantum Metrology* Vol. 2 [9] (updated to match the notation herein), “Each fit yields a solution set $\mathbb{I}^{\zeta\zeta'}(n)$, with a final value of $\chi^2(\mathbb{I}^{\zeta\zeta'}(n))$. Analysis of the fitted parameters $\chi^2(\mathbb{I}^{\zeta\zeta'})$ can be employed to probe the behaviour of the fitting algorithm, and also to gain information on how well the experimental data defines each fitted parameter. Although it is non-trivial to visualize the full χ^2 hypersurface, aspects can be probed by plotting histograms and correlation plots of the fitted parameters. A large scatter in the value of a given fit parameter over a range of fits to the same data suggests a poorly defined parameter; a consistent result meanwhile shows that a particular parameter is well defined by the dataset. The experimental data can show different sensitivities to different parameters depending on the type of ionizing transitions present, because different transitions will (according to the magnitude of the geometrical parameters and symmetry constraints [i.e. *channel functions* herein]) be more sensitive to certain partial-waves. Additionally, the presence of multiple minima in the fit may be revealed by the presence of more than one feature in the histogram, reflecting more than one “best” fit result, while correlations appearing between supposedly uncorrelated parameters can indicate emergent behaviours in the high-dimensional space or - more prosaically - issues with the fitting methodology or coding.”

The same approach is taken in the case-studies of *Part II*, which include statistical analysis to determine the “correct” *radial matrix elements* from a set of non-linear fits, and associated uncertainties, again with the hope of illustrating general methods.

Part III

Part II - Extracting matrix elements - numerical methods & case studies

CHAPTER
EIGHT

EXTRACTING MATRIX ELEMENTS OVERVIEW

In this part, various case studies are presented. To provide context, and ensure that the examples are transparent and can be run directly from the source notebooks, there are also chapters covering the general setup and configuration for the fitting routines. These are, unavoidably, rather technical and code-heavy, so readers only interested in the results should skip these sections. Additionally, these sections may be rather truncated in hard-copy (or PDF) versions of the text, but are available in full in the [Quantum Metrology Vol. 3 \(HTML version\)](#) and source notebooks for readers that wish to perform their own calculations.

Warning: As *noted elsewhere*, many components of the toolkit are still in active development, and some numerical details may change. This is particularly true for 3D alignment examples, which are here presented as new, and provisional, results.

The layout for this part is as follows:

- Technical introductions
 - Chapter 9: *Basis sets for fitting*: introduces methods for setting the basis set used for fitting, defined in terms of symmetrized harmonics.
 - Chapter 10: *General fit setup and numerics*: introduces methods for setting up the data to fit, and running fits in various ways.
- Case studies
 - Chapter 11: *Case study: Generalised bootstrapping for a homonuclear diatomic scattering system, $N_2 \sim (D_{\infty h})$* : A “simple” 1D case, here the $D_{\infty h}$ molecular symmetry matches the rotational wavepacket and detection symmetry.
 - Chapter 12: *Case study: Generalised bootstrapping for a linear heteronuclear scattering system, $OCS \sim (C_{\infty v})$* : A more complicated example. In this case, $C_{\infty v}$, up-down symmetry is broken in the molecular frame. Fitting for various cases is explored, looking at 1D and 3D alignment.
 - Chapter 13: *Case study: Generalised bootstrapping for a general asymmetric top scattering system, $C_2H_4 \sim (D_{2h})$* : The most general example of an asymmetric top system, in this case C_2H_4 (ethylene), C_{2h} . Again various cases and limitations are examined, for 1D and 3D alignment.
- Summary, discussion, conclusions and outlook: Chapter 14 presents a general overview of the case study results - particularly the density matrix and *MF-PADs* - and conclusions.

8.1 General notes on the case studies

The case studies presented herein represent various different types (classes or degrees-of-difficulty) of fitting problem. They are intended both to indicate ways to proceed in general, and also aim to address what is, and isn't, possible in these given cases. The simplest case, N_2 , has been well-studied, and reconstruction of *radial matrix elements* via the *bootstrap retrieval protocol* has proved successful in a range of valence ionization cases - see Ref. [1] for the initial experimental demonstration, and updated analysis using the *Photoelectron Metrology Toolkit* [5] in Ref. [3], as well as Chapter 11 herein.

The more complex cases, involving 3D alignment, are new, and very much work-in-progress. The current state-of-the-art is presented herein, but it is of note that these results are provisional, and work is ongoing. In particular, there are a large range of fitting configuration options and parameters which have yet to be explored in these cases, which may significantly improve on the current results in terms of computational effort required, reliability of the fitting protocol, and size of the required dataset. Nonetheless, these new results are interesting and present a stepping-stone for studies on *radial matrix elements* retrieval in complex systems, and a launching point for similar studies.

Each case study has a setup script and associated data, which was used to create sample data and configure the bootstrap retrieval protocol in each case (the general procedure is outlined in Chapter 10). However, since the fitting is computationally demanding, code execution for large fitting cases is not run upon building the book. Rather the code illustrates the general procedure (for a small batch), and existing fit data files are used if present. Each case study therefore contains an initial setup stage which can be used to run fits, or load existing data, prior to analysis.

In each case study, *bootstrap retrieval protocol* results sets are analysed at length; for a quick overview of results and fidelities in each case readers may consult the final subsection in each chapter, which provide visual summaries of the results as both density matrices and *MF-PADs*, and comparison with the reference case. These results form the basis for the summary and discussion in Chapter 14, readers not interested in the full technical details may skip directly to that chapter for general discussion, conclusions and outlook.

Finally, it is of note that - as discussed in the *preamble on book versions* - output may be significantly truncated in some cases. Readers wishing to obtain all computational details should pull the source notebooks from *Quantum Metrology Vol. 3* ([Github repo](#)), or consult the online HTML version at *Quantum Metrology Vol. 3* ([HTML version](#)). Each case study notebook is configured to pull the required data files to setup and run fits from the *ePSproc* [34] Github repo; data files for the specific example fit results examined in the chapters as published can be obtained separately from the *Quantum Metrology Vol. 3* ([Github repo](#)).

BASIS SETS FOR FITTING

As outlined in Part I, particularly Chpt. 6, in order to compute and/or retrieve a set of matrix elements from experimental results, various physical properties of the system at hand are required. In particular, the symmetry of the system, the ionizing channel(s) of interest, and the properties of the ionizing radiation are all required to define the basis set used for *radial matrix elements* retrieval via fitting (Chpt. 7).

Numerically, there are two main methods to do this demonstrated herein:

1. The use of symmetry to define the basis set, in terms of symmetry-allowed components.
2. The use of *ab initio* calculations to define the basis functions. In the work presented here, it is specifically ePolyScat (ePS) [36, 37, 38, 39] matrix elements that are used, since these are also used to generate the synthetic data used to test the *bootstrap retrieval protocol*.

Manual creation of basis sets is also possible, and may be useful in some cases, particularly when exploring limiting cases.

9.1 Symmetry-defined basis sets

As illustrated in Sect. 6.2, a set of symmetry-allowed continuum functions can be determined, corresponding to a given ionizing transition and specific dipole symmetries. Such a basis set defines the allowed matrix elements in terms of a set of symmetrized harmonics, and these can be used as a basis for fitting with only minimal knowledge of the system required.

The main advantage of a purely symmetry-defined approach is that no additional *ab initio* computations are required, and that the resulting basis set is general. As discussed and illustrated in Sect. 6.3, the resulting *channel functions* can also be used to guide analysis (and experiment if available prior to experimental work). The main disadvantage of such a blind approach is that initial data analysis may require significantly more effort than working from a computationally-defined basis, in particular the initial fitting space may be larger than required (leading to computationally more-demanding fitting runs), and testing for contributing/non-negligible terms by, e.g. using varying l_{max} , may be required in initial fitting runs. For further discussion, see *Quantum Metrology* Vols. 1 & 2 [4, 9].

9.2 Computationally-defined basis sets

In cases where photoionization calculations are available, the results can also be used to determine allowed basis components. Use of *ab initio* computational results is, of course, useful for simulation as well as direct analysis, and may lead to a reduced basis set relative to the symmetry-defined case, since some components may be small and can be ignored. However, it does also require that substantial calculations are performed (or results are available, e.g. from ePSdata [48]), and - potentially - may bias the matrix element analysis/retrieval in cases where the experimental results and computational results are significantly different.

9.3 Basis creation worked examples

In the following chapters basis functions are created for each case at hand, typically starting from *ab initio* computational results (since these are required to simulate the sample data). Here a more detailed worked example is given to illustrate the basic methodology behind the assignments, the differences between the approaches, and highlight some issues of convention which may arise.

9.3.1 Simple case: N_2 $3\sigma_g^{-1}$ ionization

As a simple example, consider the case of a homonuclear diatomic ($D_{\infty h}$ PG, cylindrically symmetric), and a totally symmetric case, e.g. the ionization of the Σ_g^+ HOMO of N_2 . For this example,

- $\Gamma^i = \Gamma^+ = \Gamma^s = \Sigma_g^+$. Note that, for single-electron ionization from a fully-occupied valence orbitals, the ion symmetry will correspond to the hole symmetry, hence the symmetry of the ionized orbital. This simple picture may break down for more complicated cases, e.g. if multi-electron effects or substantial nuclear motions are involved; for radical systems the overall symmetry of all partially-occupied orbitals must be accounted for.
- The dipole symmetries correspond to the Cartesian axes/translations given in the character tables, hence $\Sigma_u^+ = (z)$ and $\Pi_u^- = (x, y)$ for $D_{\infty h}$. Note that, in this cylindrically symmetric case, the Cartesian (x, y) components are spanned by the doubly-degenerate Π_u^- irreducible representation - physically this corresponds to the arbitrary orientation of these axes in the MF (i.e. no preferred direction is defined).

Following Eq. (6.12), the allowed components (irreducible representations) can be determined by hand making use of character and direct product tables, and substituting in for the terms defined above:

$$\Gamma^+ \otimes \Gamma^e \otimes \Gamma_{\text{dipole}} \otimes \Gamma^i \supseteq \Sigma_g^+ \quad (9.1)$$

$$\Sigma_g^+ \otimes \Gamma^e \otimes \frac{\Pi_u^-(x, y)}{\Sigma_u^+(z)} \otimes \Sigma_g^+ \supseteq \Sigma_g^+ \quad (9.2)$$

$$\Sigma_g^+ \otimes \Gamma^e \otimes \frac{\Pi_u^-(x, y)}{\Sigma_u^+(z)} \supseteq \Sigma_g^+ \quad (9.3)$$

$$\Gamma^e \otimes \frac{\Pi_u^-(x, y)}{\Sigma_u^+(z)} \supseteq \Sigma_g^+ \quad (9.4)$$

Group theory character tables and related

Character tables, direct product tables and related information can be found at various sources online, or in textbooks, e.g. Refs. [140, 141, 142]. For $D_{\infty h}$ the pages at symmetry.jacobs-university.de provide a good quick reference; for extended tables including spherical harmonic symmetries and direct products the pages from G. Katzer are useful.

Hence the allowed continuum components are given by:

$$\Gamma^e = \frac{\Pi_u^-(x, y)}{\Sigma_u^+(z)} \quad (9.5)$$

As indicated above, this case is split by symmetry into a “parallel” and “perpendicular” continua, accessed by the z or (x, y) dipole components in the MF respectively; these are often denote generically by appending \perp and \parallel symbols to derived quantities, e.g. σ_{\parallel} for the corresponding cross-section. In the LF or AF these continua can be mixed according to the polarization state and geometry of the ionizing radiation, and the molecular alignment (ADMs).

The total scattering state symmetry is often also used to label continuum states, and is given by $\Gamma_{\text{scat}} = \Gamma^+ \otimes \Gamma^e$:

$$\Gamma_{\text{scat}} = \Sigma_g^+ \otimes \frac{\Pi_u^-(x, y)}{\Sigma_u^+(z)} = \frac{\Pi_u^-(x, y)}{\Sigma_u^+(z)} \quad (9.6)$$

This is identical to Γ^e in this simple case.

9.3.2 Degenerate case example: N_2 $3\pi_u^{-1}$ ionization

For more complicated cases, multiple symmetry components may be found. For example, the ionization from a Π_u orbital, e.g. N_2 HOMO-1.

In this case, $\Gamma^+ = \Pi_u$, and - working through the direct products as above - yields the allowed continuum components:

$$\Gamma^e = \frac{\Sigma_g^+ + \Delta_g}{\Pi_g(z)}(x, y) \quad (9.7)$$

And total scattering state symmetries:

$$\Gamma_{\text{scat}} = \Pi_u \otimes \frac{\Sigma_g^+ + \Delta_g}{\Pi_g(z)}(x, y) = \frac{\Pi_u + \Delta_u + \Phi_u}{\Sigma_u^+ + \Sigma_u^- + \Delta_u}(z) \quad (9.8)$$

In this case, the direct product results give multiple components for each continua. However, since the continuum wavefunction is already defined for multiple components, as given by Γ^e , only the first Γ_{scat} symmetry is required as a unique label here. In this case, therefore, $\Gamma_{\text{scat}} = \Pi_u(x, y)$ and $\Gamma_{\text{scat}} = \Sigma_u^+(z)$ suffice to label the total scattering states.

9.3.3 Defining symmetrized harmonics

In the examples above, the allowed irreducible representations are defined by hand from direct product tables for illustrative purposes. But, in general, it is tedious to categorize/define the allowed spherical harmonics and linear combinations. With the [Photoelectron Metrology Toolkit](#) [5], both the direct products illustrated above, and the determination of associated spherical harmonics, can be automated and the full basis set rapidly defined numerically. This was illustrated briefly in Sect. 6.2, and is extended here for the example cases above.

Note: Full tabulations of the parameters available in HTML or notebook formats only.

Computationally, the cylindrically-symmetric ∞ groups can be approximated by a high-order group, e.g. $D_{\infty h} \approx D_{10h}$. (For a cross-check, see the [full tables and direct products online](#).) Here the notational convention is $A1 = \Sigma^+$, $A2 = \Sigma^-$, $E1 = \Pi$, $E2 = \Delta$ and so forth (see the $D_{\infty h}$ character table for more details).

```
# Example following symmetrized harmonics demo

# Import class
from pemtk.sym.symHarm import symHarm

# Compute hamronics for D10h, lmax=4
sym = 'D10h'
lmax=4

symObjA1g = symHarm(sym, lmax)

# Allowed terms and mappings are given in 'dipoleSyms'
# symObj.dipole['dipoleSyms']
```

```
*** Mapping coeffs to ePSproc dataType = matE
Remapped dims: {'C': 'Cont', 'mu': 'it'}
Added dim Eke
```

(continues on next page)

(continued from previous page)

```

Added dim Targ
Added dim Total
Added dim mu
Added dim Type
Found dipole symmetries:
{'A2u': {'m': [0], 'pol': ['z']}, 'E1u': {'m': [-1, 1, -1, 1], 'pol': ['x', 'y']}}

```

```

# Setting the symmetry for the neutral and ion allows direct products to be computed,
# and allowed terms to be determined.

sNeutral = 'A1g'
sIonSG = 'A1g'

symObjA1g.directProductContinuum([sNeutral, sIonSG])

# Results are pushed to self.continuum, in dictionary and Pandas DataFrame formats,
# and can be manipulated using standard functionality.
# The subset of allowed values are also set to a separate DataFrame and list.

# Glue figure for later - real part only in this case
# Also clean up axis labels from default state labels ('LM' and 'LM_p' in this case).
glue("dipoleTermsD10hA1g", symObjA1g.continuum['allowed']['PD'])

```

Dipole	Target	allowed	m	pol	result	terms
A2u	A2u	True	[0]	[z]	[A1g]	[A1g, A1g]
E1u	E1u	True	[-1, 1, -1, 1]	[x, y]	[A1g, A2g, E2g]	[A1g, A1g]

Fig. 9.1: Dipole-allowed continuum symmetries (“Target”) for D_{10h} , A_{1g} ionization.

```

# Setting the symmetry for the neutral and ion allows direct products to be computed,
# and allowed terms to be determined.

sNeutral = 'A1g'
sIonPU = 'E1u'

# Define new object for E1u case
symObjE1u = symHarm(sym, lmax)
symObjE1u.directProductContinuum([sNeutral, sIonPU])

# Results are pushed to self.continuum, in dictionary and Pandas DataFrame formats,
# and can be manipulated using standard functionality.
# The subset of allowed values are also set to a separate DataFrame and list.

# Glue table for later
glue("dipoleTermsD10hE1u", symObjE1u.continuum['allowed']['PD'])

```

The allowed terms can be further expressed in terms of the spherical harmonic components.

```

# Basis table with the Character values limited to those defined in
# self.continuum['allowed']['PD'] Target column

```

(continues on next page)

Dipole	Target	allowed	m	pol	result	terms
A2u	E1g	True	[0]	[z]	[A1g, A2g, E2g]	[A1g, E1u]
E1u	A1g	True	[-1, 1, -1, 1]	[x, y]	[A1g, A2g, E2g]	[A1g, E1u]
	A2g	True	[-1, 1, -1, 1]	[x, y]	[A1g, A2g, E2g]	[A1g, E1u]
	E2g	True	[-1, 1, -1, 1]	[x, y]	[A1g, A2g, E2g, E4g]	[A1g, E1u]

Fig. 9.2: Dipole-allowed continuum symmetries (“Target”) for D_{10h} , E_{1u} ionization.

(continued from previous page)

```

symObjE1u.displayXlm(symFilter = True, YlmType='comp')

# Glue table for later
glue("dipoleTermsD10hBasis", symObjE1u.displayXlm(symFilter = True,
                                                    YlmType='comp', returnPD=True))

```

Character (\$\Gamma\$)	SALC (h)	PFIX (\$\mu\$)	m	b			3	4
				1	0	1		
A1g	0	0	0	(1+0j)				
	1	0	0		(1+0j)			
	2	0	0					(1+0j)
E1g	0	0	-1			(0.7071067811865475+0j)		
			1			(0.7071067811865475-0j)		
			1	-1		(-0.7071067811865475-0j)		
			1	1		(0.7071067811865475-0j)		
	1	0	-1				(0.7071067811865475+0j)	
			1				(0.7071067811865475-0j)	
E2g	0	0	-2			(-0.7071067811865475+0j)		
			2			(0.7071067811865475+0j)		
			1	-2		(-0.7071067811865475+0j)		
			2	2		(0.7071067811865475+0j)		
	1	0	-2				(0.7071067811865475+0j)	
			2				(0.7071067811865475+0j)	
			1	-2			(-0.7071067811865475+0j)	
			2	2			(-0.7071067811865475+0j)	

Fig. 9.3: Dipole-allowed basis set for D_{10h} , E_{1u} ionization.

9.3.4 Mapping symmetrized harmonics to fit parameters

The final preparatory steps for tackling a specific retrieval problem is to map the allowed channels to photoionization matrix elements - including the assignment of any missing terms - and from these to fitting parameters. The matrix elements are currently defined in the [Photoelectron Metrology Toolkit](#) [5] and the [ePSproc codebase](#) [33, 34, 35] following the definitions in [ePolyScat \(ePS\)](#) [36, 37, 38, 39], and the symmetry-defined cases can be remapped to this format. (Further information can be found in the [PEMtk documentation](#) [20].) For a comparison with *ab initio* matrix elements, see [Sect. 9.4](#).

1. For symmetry-defined basis sets, these must first be mapped to an ePSproc data structure, although this step is not necessary when working from *ab initio* basis sets. This is explored in [Sect. 9.3.4](#).
2. From a given basis set, the parameters used for fitting data are determined. This converts the parameters to [lmfit library](#) [66, 67] objects in magnitude-phase form, and (optionally) sets various symmetry relations and constraints. This is explored in [Sect. 9.3.4](#).

Further examples can be found in the remainder of this text, and also in the [PEMtk documentation](#) [20].

Remapping to ePolyScat definitions

In the remapping, the code attempts to assign all the symmetries matching ePolyScat definitions from the direct products.

The terms are:

1. Cont is the continuum (free electron) symmetry, Γ^e .
2. Targ is the target state symmetry, Γ^+ .
3. Total is the overall symmetry of the scattering state, $\Gamma_{\text{scat}} = \Gamma^+ \otimes \Gamma^e$.

Additionally, the current default remapping changes some of the terms defined by the symmetrized harmonics routines and conventions:

- Symmetry C > Cont (continuum symmetry label in ePSproc)
- Index h > it (degeneracy index in ePSproc)
- Index mu > muX (to avoid confusion with photon index mu in ePSproc)

Note, in particular, that μ is - unfortunately - the photon polarization term in the conventional photoionization equations, but also used in the standard definition of the symmetrized harmonics as a degeneracy index. In some cases, the symmetrization indices μ or h may be redundant, and can be dropped or summed over, but care must be taken here to avoid breaking the symmetry of the simplified basis set.

Finally, the remapping adds additional labels used by [ePolyScat \(ePS\)](#) [36, 37, 38, 39], but which are not necessarily required in general:

- Type: for ePolyScat results, this labels length or velocity gauge results; this is assigned as U (unassigned) in the conversion.
- Eke: the photoelectron kinetic energy, for the basis states this is just set to 0.
- mu: this is set to NaN by the main routine; if the ionizing channel symmetries are defined these values (the photon polarization) can be determined from the dipole-allowed terms, and this is done by the `assignSymMuTerms()` method, as illustrated below.

```
# Run conversion with a different dimMap & dataType - note this includes all
# symmetries,
# and both real and complex harmonic basis sets
dataType = 'matE'
```

(continues on next page)

(continued from previous page)

```

# With custom dim mapping (optional)...
dimMap = {'C':'Cont', 'mu':'it'}    # Default dimMap = {'C':'Cont', 'h':'it', 'mu':'muX'}
# dimMap = {'C':'Cont', 'h':'it', 'mu':'muX'}  # Default case

# Map to ePSproc definitions
symObjA1g.toePSproc(dataType=dataType, dimMap=dimMap)

# To assign specific terms, use self.assignMissingSym
# Note this can take a single value, or a list which must match the size of the
# Sym multiindex defined in the Xarray dataset.
symObjA1g.assignMissingSym('Targ', sIonSG)

# To define terms from products, use self.assignMissingSymProd
symObjA1g.assignMissingSymProd()

# To attempt to assign mu values (by symmetry), use self.assignSymMuTerms()
symObjA1g.assignSymMuTerms()

# Show Pandas table of results
symObjA1g.coeffs['symAllowed']['PD'].fillna('')

```

```

*** Mapping coeffs to ePSproc dataType = mate
Remapped dims: {'C': 'Cont', 'mu': 'it'}
Added dim Eke
Added dim Targ
Added dim Total
Added dim mu
Added dim Type
*** Updated self.coeffs['mateE'] with new coords.
Assigned 'Total' from A1g x A1g = ['A1g']
Assigned 'Total' from A2u x A1g = ['A2u']
Assigned 'Total' from E1g x A1g = ['E1g']
Assigned 'Total' from E1u x A1g = ['E1u']
Assigned 'Total' from E2g x A1g = ['E2g']
Assigned 'Total' from E2u x A1g = ['E2u']
Assigned 'Total' from E3g x A1g = ['E3g']
Assigned 'Total' from E3u x A1g = ['E3u']
Assigned 'Total' from E4g x A1g = ['E4g']
*** Updated self.coeffs['mateE'] with new coords.
Assigned dipole-allowed terms for dim = 'Cont' to self.coeffs['symAllowed']

```

Eke	Targ	Total	Type	h	it	l	m	Cont	A2u	E1u
								mu		
0	A1g	A2u	U	0	0	1	0	0	(1+0j)	
				1	0	3	0	0	(1+0j)	
		E1u	U	0	0	1	-1	-1		(0.7071067811865475+0j)
								1		(0.7071067811865475+0j)
								1		(-0.7071067811865475-0j)
								1		(-0.7071067811865475-0j)
						1	-1	-1		(-0.7071067811865475+0j)
								1		(-0.7071067811865475+0j)
								1		(-0.7071067811865475-0j)
								1		(-0.7071067811865475-0j)
				1	0	3	-1	-1		(0.7071067811865475+0j)
								1		(0.7071067811865475+0j)
								1		(-0.7071067811865475-0j)
								1		(-0.7071067811865475-0j)
				1	3	-1	-1	-1		(-0.7071067811865475+0j)
								1		(-0.7071067811865475+0j)
								1		(-0.7071067811865475-0j)
								1		(-0.7071067811865475-0j)

```
# Plot values
%matplotlib inline
titleString=f'Symmetrized matrix elements defined for {sym}, {sNeutral}^{{{-1}}}{_{-1}}_{-ionization''

_, gFig = ep.lmPlot(symObjA1g.coeffs['symAllowed']['XR']['b (comp)'],
                     titleString=titleString, xDim={'LM': ['l', 'm']}, sumDims='h',
                     labelCols = [1,1])

# Glue figure for later
glue("matEremapD10hA1g", gFig.fig)
```

Mapping to fitting parameters (and reduction)

Finally, the basis set of matrix elements can be set to a set of fitting parameters. In this case, as per Eq. (6.15), the parameters are mapped to magnitude-phase form; additionally, the fitting routine allows for the definition of relationships between the parameters. This provides a way to reduce the effective size of the basis set to only the unique values, with other terms defined purely by their symmetry relations. Consequently, degenerate cases, as detailed above, as well as cases with defined phase relations, can be efficiently reduced to a smaller basis set for fitting. Note that the default routine labels parameters by the full set of quantum numbers, with an `m` or `p` prefix to denote the magnitude and phase terms corresponding to the partial-wave channel; this ensures a unique naming scheme, but is also rather unwieldy (as can be seen below). In cases where fewer quantum numbers are required these can be defined and the parameter names remapped, see the [PEMtk documentation](#) [20] for further details.

For quick setup, there is an automated routine to set relations if applicable. The automated routine currently checks for the following relationships: identity (equal complex values), magnitude and phase equality, complex rotations by $\pm\pi$, where matrix elements are grouped by symmetry (specifically `Cont`) and `l` prior to pair-wise testing. For more control, additional functions can be passed. Alternatively, the automatic setting can be skipped and/or relationships redefined or set manually. This provides a way to test if the symmetry-definitions are manifest in experimental data, rather than imposing them during fitting, or to explore other possible correlations between fitted parameters. Note, however, that in some cases

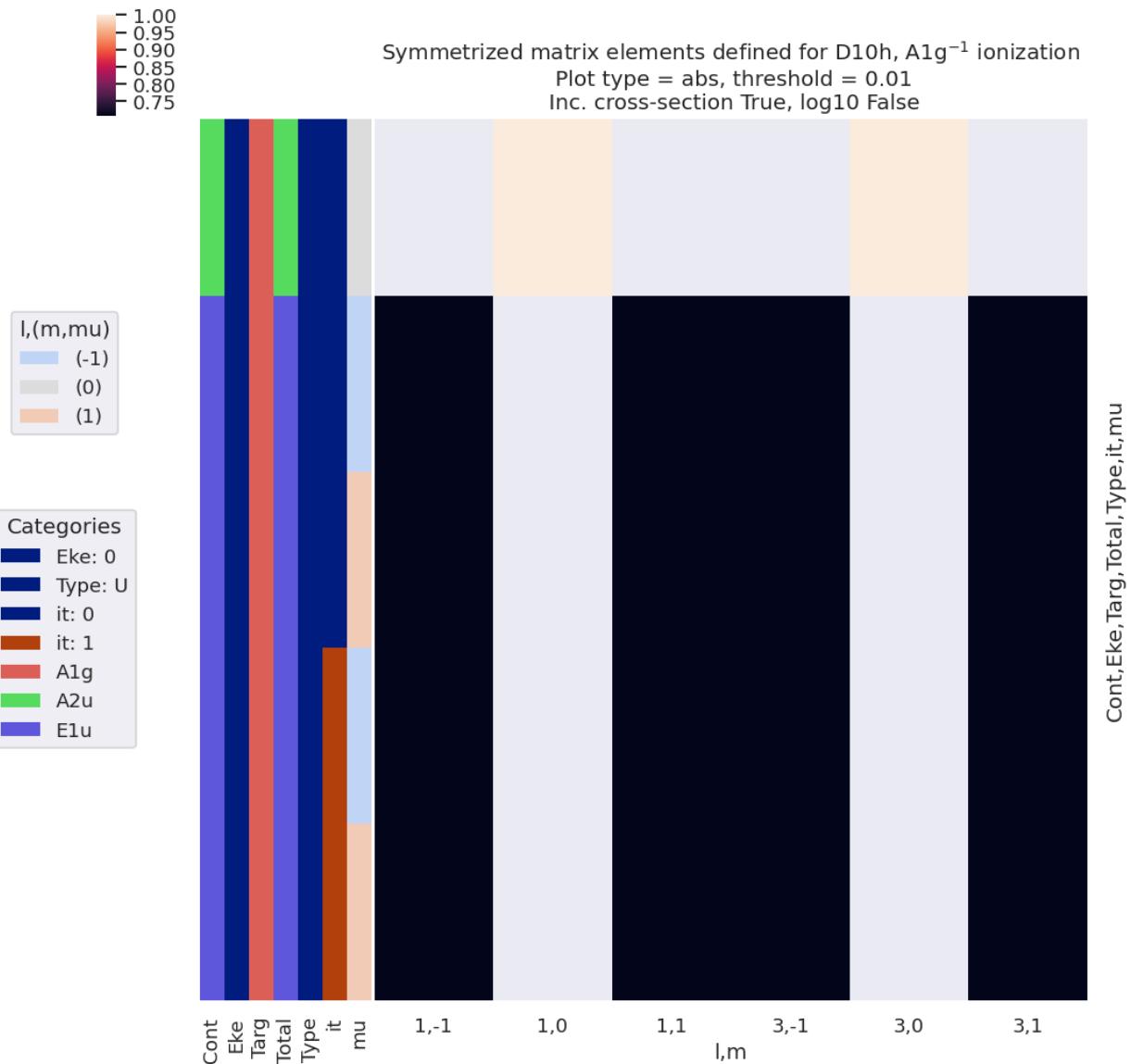


Fig. 9.4: Dipole-allowed continuum matrix elements for D_{10h} , A_{1g} ionization, arranged by (l, m) .

the number of unique parameters in an unsymmetrized case may be large, so care should also be taken to ensure that fit results are meaningful in such cases (e.g. by employing a sufficiently large dataset, and testing for reproducibility).

```
# Default matrix element relationship tests are set by symCheckDefns
from pemtk.fit._sym import symCheckDefns

symCheckDefns()

{'i': {'name': 'identity',
       'lam': <function pemtk.fit._sym.symCheckDefns.<locals>.<lambda>(x)>,
       'transform': False,
       'constraint': 'x'},
 'abs': {'name': 'abs',
        'lam': <function pemtk.fit._sym.symCheckDefns.<locals>.<lambda>(x)>,
        'transform': True,
        'constraint': 'm_x'},
 'phase': {'name': 'phase',
           'lam': <function pemtk.fit._sym.symCheckDefns.<locals>.<lambda>(x)>,
           'transform': True,
           'constraint': 'p_x'},
 'crot_p': {'name': 'Complex rotation +pi/2',
            'lam': <function pemtk.fit._sym.symCheckDefns.<locals>.<lambda>(x)>,
            'transform': False,
            'constraint': 'arctan2(sin(p_x+pi/2), cos(p_x+pi/2))'},
 'crot_m': {'name': 'Complex rotation -pi/2',
            'lam': <function pemtk.fit._sym.symCheckDefns.<locals>.<lambda>(x)>,
            'transform': False,
            'constraint': 'arctan2(sin(p_x-pi/2), cos(p_x-pi/2))'}}
```

Automated assignment from defined matrix elements

```
from pemtk.fit.fitClass import pemtkFit

# Example using data class (setup in init script)
data = pemtkFit()

# Set to new key in data class
dataKey = sym
data.data[dataKey] = {}

# Assign allowed matrix elements to fit object
dataType = 'matE'
# General case - just use complex coeffs directly
# data.data[dataKey][dataType] = symObj.coeffs[dataType]['b (comp)']

# Specific case - e.g. sum over 'h'
data.data[dataKey][dataType] = symObjA1g.coeffs['symAllowed']['XR']['b (comp)'].sum('h
    ↴')

# Propagate attrs
data.data[dataKey][dataType].attrs = symObjA1g.coeffs['symAllowed']['XR'].attrs

# Update selection with options
# E.g. Matrix element sub-selection
# data.selOpts['matE'] = {'thres': 0.01, 'inds': {'Type': 'U', 'Cont': 'A1'}}
data.selOpts['matE'] = {'thres': 0.01, 'inds': {'Type': 'U'}}
```

(continues on next page)

(continued from previous page)

```
data.setSubset(dataKey = sym, dataType = 'matE')    #, resetSelectors=True)  #_
↪Subselect from 'orb5' dataset, matrix elements

# And for the polarisation geometries...
# data.selOpts['pol'] = {'inds': {'Labels': 'z'}}
# data.setSubset(dataKey = 'pol', dataType = 'pol')
```

Subselected from dataset 'D10h', dataType 'matE': 72 from 540 points (13.33%)

```
# Set matrix elements to fitting parameters
# Running for the default case will attempt to automatically set the relations_
↪between
# matrix elements according to symmetry.
data.setMatEFit()
```

Modifying fitting basis parameters

A brief illustration of defining constraints is given below, for more details see the PEMtk documentation [20], particularly the [basic fitting guide](#). For more details on the base lmfit parameters class that is used here, see [lmfit library](#) [66, 67], particularly the documentation on [parameters](#) and [constraints](#).

Note: Full tabulations of the parameters available in HTML or notebook formats only.

```
# Set parameters with NO constraints set (except a reference phase)
data.setMatEFit(paramsCons = None)
```

```
# To add manual constraints
# Set param constraints as dict
# Any basic mathematical relations can be set here,
# see https://lmfit.github.io/lmfit-py/constraints.html
paramsCons = {}
paramsCons['m_A2u_0_A1g_A2u_1_0_0_0'] = '5*m_A2u_0_A1g_A2u_3_0_0_0'

# Missing settings will generate an error message
paramsCons['test'] = 'p_PU_SG_PU_3_1_n1_1'

# Init parameters with specified constraints
data.setMatEFit(paramsCons = paramsCons)
```

```
# Individual parameters can be addressed by name,
data.params['m_E1u_0_A1g_E1u_1_n1_n1_0']
```

```
<Parameter 'm_E1u_0_A1g_E1u_1_n1_n1_0', value=0.7071067811865475, bounds=[0.0001:5.
↪0]>
```

	name	value	stderr	vary	expr	init_value	min	max
0	m_A2u_0_A1g_A2u_1_0_0_0	1.000	None	True	None	1.000	1.000e-04	5.000
1	m_A2u_0_A1g_A2u_3_0_0_0	1.000	None	True	None	1.000	1.000e-04	5.000
2	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	None	True	None	0.707	1.000e-04	5.000
3	m_E1u_0_A1g_E1u_1_n1_n1_1	0.707	None	True	None	0.707	1.000e-04	5.000
4	m_E1u_0_A1g_E1u_1_n1_1_0	0.707	None	False	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	1.000e-04	5.000
5	m_E1u_0_A1g_E1u_1_n1_1_1	0.707	None	False	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	1.000e-04	5.000
6	m_E1u_0_A1g_E1u_1_1_n1_0	0.707	None	False	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	1.000e-04	5.000
7	m_E1u_0_A1g_E1u_1_1_n1_1	0.707	None	False	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	1.000e-04	5.000
8	m_E1u_0_A1g_E1u_1_1_1_0	0.707	None	False	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	1.000e-04	5.000
9	m_E1u_0_A1g_E1u_1_1_1_1	0.707	None	False	m_E1u_0_A1g_E1u_1_n1_n1_0	0.707	1.000e-04	5.000
10	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	None	True	None	0.707	1.000e-04	5.000
11	m_E1u_0_A1g_E1u_3_n1_n1_1	0.707	None	True	None	0.707	1.000e-04	5.000
12	m_E1u_0_A1g_E1u_3_n1_1_0	0.707	None	False	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	1.000e-04	5.000
13	m_E1u_0_A1g_E1u_3_n1_1_1	0.707	None	False	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	1.000e-04	5.000
14	m_E1u_0_A1g_E1u_3_1_n1_0	0.707	None	False	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	1.000e-04	5.000
15	m_E1u_0_A1g_E1u_3_1_n1_1	0.707	None	False	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	1.000e-04	5.000
16	m_E1u_0_A1g_E1u_3_1_1_0	0.707	None	False	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	1.000e-04	5.000
17	m_E1u_0_A1g_E1u_3_1_1_1	0.707	None	False	m_E1u_0_A1g_E1u_3_n1_n1_0	0.707	1.000e-04	5.000
18	p_A2u_0_A1g_A2u_1_0_0_0	0.000	None	False	None	0.000	-3.142e+00	3.142
19	p_A2u_0_A1g_A2u_3_0_0_0	0.000	None	True	None	0.000	-3.142e+00	3.142
20	p_E1u_0_A1g_E1u_1_n1_n1_0	0.000	None	True	None	0.000	-3.142e+00	3.142
21	p_E1u_0_A1g_E1u_1_n1_n1_1	3.142	None	True	None	3.142	-3.142e+00	3.142
22	p_E1u_0_A1g_E1u_1_n1_1_0	0.000	None	False	p_E1u_0_A1g_E1u_1_n1_n1_0	0.000	-3.142e+00	3.142
23	p_E1u_0_A1g_E1u_1_n1_1_1	3.142	None	False	p_E1u_0_A1g_E1u_1_n1_n1_1	3.142	-3.142e+00	3.142
24	p_E1u_0_A1g_E1u_1_1_n1_0	3.142	None	False	p_E1u_0_A1g_E1u_1_n1_n1_1	3.142	-3.142e+00	3.142
25	p_E1u_0_A1g_E1u_1_1_n1_1	3.142	None	False	p_E1u_0_A1g_E1u_1_n1_n1_1	3.142	-3.142e+00	3.142
26	p_E1u_0_A1g_E1u_1_1_1_0	3.142	None	False	p_E1u_0_A1g_E1u_1_n1_n1_1	3.142	-3.142e+00	3.142
27	p_E1u_0_A1g_E1u_1_1_1_1	3.142	None	False	p_E1u_0_A1g_E1u_1_n1_n1_1	3.142	-3.142e+00	3.142
28	p_E1u_0_A1g_E1u_3_n1_n1_0	0.000	None	True	None	0.000	-3.142e+00	3.142
29	p_E1u_0_A1g_E1u_3_n1_n1_1	3.142	None	True	None	3.142	-3.142e+00	3.142
30	p_E1u_0_A1g_E1u_3_n1_1_0	0.000	None	False	p_E1u_0_A1g_E1u_3_n1_n1_0	0.000	-3.142e+00	3.142
31	p_E1u_0_A1g_E1u_3_n1_1_1	3.142	None	False	p_E1u_0_A1g_E1u_3_n1_n1_1	3.142	-3.142e+00	3.142
32	p_E1u_0_A1g_E1u_3_1_n1_0	3.142	None	False	p_E1u_0_A1g_E1u_3_n1_n1_1	3.142	-3.142e+00	3.142
33	p_E1u_0_A1g_E1u_3_1_n1_1	3.142	None	False	p_E1u_0_A1g_E1u_3_n1_n1_1	3.142	-3.142e+00	3.142
34	p_E1u_0_A1g_E1u_3_1_1_0	3.142	None	False	p_E1u_0_A1g_E1u_3_n1_n1_1	3.142	-3.142e+00	3.142
35	p_E1u_0_A1g_E1u_3_1_1_1	3.142	None	False	p_E1u_0_A1g_E1u_3_n1_n1_1	3.142	-3.142e+00	3.142

Fig. 9.5: Fitting parameters as assigned for D_{10h} , A_{1g} ionization. Note the vary column, which defines the symmetry-unique values for fitting, whilst the expression column indicates the relationships of the non-unique values to the floated parameters.

```
# Properties can be modified directly...
data.params['m_A2u_0_A1g_A2u_3_0_0_0'].value = 1.777

# ...or by using lmfit's `set()` method.
data.params['m_E1u_0_A1g_E1u_1_n1_n1_0'].set(value = 1.36)
data.params['m_E1u_0_A1g_E1u_1_n1_n1_0'].set(vary = False)
data.params['m_E1u_0_A1g_E1u_1_n1_n1_0']
```

```
<Parameter 'm_E1u_0_A1g_E1u_1_n1_n1_0', value=1.36 (fixed), bounds=[0.0001:5.0]>
```

```
# The full set can always be checked via self.params
data.params
```

```
# The full mapping of parameter names and indexes is given in self.lmmu
data.lmmu
```

Manually setting fitting basis

To modify and/or set a basis set manually, the same functions can be used with different inputs and/or options. A few examples are given here, see the [PEMtk documentation \[20\]](#) for more information.

```
# Manual configuration of matrix elements
# Example using data class
dataManual = pemtkFit()

# Manual setting for matrix elements
# See API docs at https://epsproc.readthedocs.io/en/dev/modules/epsproc.util.setMatE.
#html
EPoints = 10
dataManual.setMatE(data = [[0, 0, *np.ones(EPoints)], [2, 0, *np.linspace(0, 1, EPoints)],
                           [4, 0, *np.linspace(0, 0.5, EPoints)]], dataNames=['l', 'm'])

# Matrix elements are set to Xarray and Pandas formats, under the 'matE' key
dataManual.data['matE']['matE'].pd
```

```
# To use manual settings for fitting, set `conformDims=True` to ensure ePSproc_
#labelling
dataManual.setMatE(data = [[0, 0, *np.ones(EPoints)], [2, 0, *np.linspace(0, 1, EPoints)],
                           [4, 0, *np.linspace(0, 0.5, EPoints)]], dataNames=['l', 'm'], conformDims=True)

# Then use as normal
dataManual.selOpts['matE'] = {'thres': 0.01, 'inds': {'Type': 'U', 'Eke': 1} }
dataManual.setSubset(dataKey = 'matE', dataType = 'matE')
dataManual.data[data.subKey]['matE']['it'] = [1] # In some cases NaN values may_
#need to be set for setMatEFit.
dataManual.setMatEFit()
```

9.4 Comparison with symmetry-defined and computational matrix elements

For comparison of a given symmetry-defined basis set with sample *ab initio* calculations using ePolyScat (ePS) [36, 37, 38, 39] calculations, results can be computed locally or pulled from the web. Some sample/test datasets can be found as part of the ePSproc repo, which includes the case study examples herein. Further ePolyScat (ePS) [36, 37, 38, 39] datasets are available from the ePSdata [48], and data can be pulled using the python ePSdata interface.

In the following, the test case above for N_2 $3\sigma_g^{-1}$ ionization is illustrated. Note that this comparison shows the results of a full *ab initio* computation of the matrix elements (Eq. (6.3)) versus the symmetry-allowed harmonics and associated $b_{hl\lambda}^{\Gamma\mu}$ parameters (Eq. (6.37)). In the former case, the $b_{hl\lambda}^{\Gamma\mu}$ are incorporated into the numerical results, but the full angular momentum selection rules and dipole integrals are also included; in the latter case the $b_{hl\lambda}^{\Gamma\mu}$ parameters serve to define the allowed matrix elements, and symmetry relations (e.g. phase, rotations and degeneracy), but *do not* include any other effects. Hence the comparison here indicates whether the symmetry-defined basis set is sufficient for a matrix element reconstruction, but it may contain terms which are zero in practice, or otherwise drop out from the complete photoionization treatment. Some conventions may also be different.

```
# Pull N2 data from ePSproc Github repo
dataName = 'n2Data'

# Set data dir
dataPath = Path(Path.cwd(), dataName)

# For pulling data from Github, a utility function is available
# This requires the repo subpath, and optionally branch
fDictMatE, fAllMatE = ep.util.io.getFilesFromGithub(subpath='data/photoionization/n2_
˓multiorb',
                                                 dataName=dataName, ref='dev') #_

˓Optional settings
```

```
Querying URL: https://api.github.com/repos/phockett/epsproc/contents/data/
˓photoionization/n2_multiorb?ref=dev
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
˓n2Data/n2_1pu_0.1-50.1eV_A2.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
˓n2Data/n2_3sg_0.1-50.1eV_A2.inp.out already exists
```

```
# Import data with PEMtk class
# For more details on ePSproc usage see
# https://epsproc.readthedocs.io/en/dev/demos/ePSproc_class_demo_161020.html

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

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

```
*** Job subset details
Key: subset
No 'job' info set for self.data[subset].
```

(continues on next page)

(continued from previous page)

```
*** Job orb6 details
Key: orb6
Dir /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/n2Data, 1_
  ↵file(s).
{   'batch': 'ePS n2, batch n2_1pu_0.1-50.1eV, orbital A2',
    'event': ' N2 A-state (1piu-1)',
    'orbE': -17.09691397835426,
    'orbLabel': '1piu-1'}

*** Job orb5 details
Key: orb5
Dir /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/n2Data, 1_
  ↵file(s).
{   'batch': 'ePS n2, batch n2_3sg_0.1-50.1eV, orbital A2',
    'event': ' N2 X-state (3sg-1)',
    'orbE': -17.34181645456815,
    'orbLabel': '3sg-1'}
```

```
# Format and display results from previous cell (hidden in some formats)
display_html(df1_styler._repr_html_() + df2_styler._repr_html_(), raw=True)
```

Here the basis sets are identical, aside from the difference in l_{max} .

Here the symmetry-defined basis has two degenerate continua, $it=0, 1$, with a phase rotation applied between them. For $it=0$ the $\pm m$ terms are anti-phase, whilst for $it=1$ they are in-phase (all negative). For the ePS basis, only the anti-phase component is present, and is further reduced to terms with m and mu of opposite sign. These differences are due to additional restrictions imposed by angular momentum selection rules, which are not included in the symmetry-defined case.

In general, the current mappings should be suitable for simulation and reconstruction, but care should be taken to:

1. Confirm symmetry and angular momentum relations for a given case.
2. Apply additional transformations if comparison with computational results is required.
3. Add degeneracy factors if required (otherwise will be subsumed into matrix element values).

GENERAL FIT SETUP AND NUMERICS

For the case studies in Chapter 11 - 13, the same basic setup and fitting routine is used in all cases, and this is outlined below. In general, this requires the steps outlined in Chpt. 9 and, for the case studies, configuration additionally requires ePolyScat (ePS) [36, 37, 38, 39] *ab initio radial matrix elements*, and *ADMs*, in order to define test datasets (Sect. 10.3). The N_2 case study is use as an example in this case, and Sect. 10.4 illustrates both setting test data, and running fits. Additionally, for the case studies herein, the setup routines are wrapped in a basic script, with configuration options for each case included, this is illustrated in Sect. 10.4.4.

For the case studies, all the sample data is available from the ePSproc [34] Github repo, and the examples below include steps for pulling the required data files. Note that further ePolyScat (ePS) [36, 37, 38, 39] datasets are available from the ePSdata [48], and data can be pulled using the python ePSdata interface.

10.1 Init and pulling data

Here the setup is mainly handled by some basic scripts, these follow the outline in the PEMtk documentation [20], see in particular the intro to fitting.

```
# Pull data files as required from Github, note the path here is required

# *** Method using epsproc.util.io.GetFilesFromGithub
# For pulling data from Github, a utility function is available
# This requires the repo subpath, and optionally branch
# The function will pull all files found in the repo path
from epsproc.util.io import getFilesFromGithub

# Set dataName (will be used as download subdir)
dataName = 'n2fitting'
# N2 matrix elements
fDictMatE, fAllMatE = getFilesFromGithub(subpath='data/photoionization/n2_multiorb',_
    ↴dataName=dataName)
# N2 alignment data
fDictADM, fAllMatADM = getFilesFromGithub(subpath='data/alignment', dataName=dataName)

# *** Alternative method: supply URLs directly for file downloader
# E.g. Pull N2 data from ePSproc Github repo

# URLs for test ePSproc datasets - n2
# For more datasets use ePSdata, see https://epsproc.readthedocs.io/en/dev/demos/
# ↴ePSdata_download_demo_300720.html
urls = {'n2PU':"https://github.com/phockett/ePSproc/blob/master/data/photoionization/
    ↴n2_multiorb/n2_1pu_0.1-50.1eV_A2.inp.out",
        'n2SU':"https://github.com/phockett/ePSproc/blob/master/data/photoionization/
    ↴n2_multiorb/n2_3sg_0.1-50.1eV_A2.inp.out",
```

(continues on next page)

(continued from previous page)

```
'n2ADMs':"https://github.com/phockett/ePSproc/blob/master/data/alignment/N2_
→ADM_VM_290816.mat",
'demoScript':"https://github.com/phockett/PEMtk/blob/master/demos/fitting/
→setup_fit_demo.py"}
```

from ePSproc.util.io **import** getFilesFromURLs
fList, fDict = getFilesFromURLs(urls, dataName=dataName)

```
Querying URL: https://api.github.com/repos/phockett/ePSproc/contents/data/
↳photoionization/n2_multiorb
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/n2_1pu_0.1-50.1eV_A2.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/n2_3sg_0.1-50.1eV_A2.inp.out already exists
Querying URL: https://api.github.com/repos/phockett/ePSproc/contents/data/alignment
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/N2 ADM_VM_290816.mat already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/n2_1pu_0.1-50.1eV_A2.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/n2_3sg_0.1-50.1eV_A2.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/N2 ADM_VM_290816.mat already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/setup_fit_demo.py already exists
```

10.2 Setup with options

Following the PEMtk documentation [20], the fitting workspace can be configured by setting:

1. A fitting basis set, either from computational matrix elements, from symmetry constraints, or manually. (See Chpt. 9 for more discussion.)
2. Data to fit. In the examples herein synthetic data will be created by adding noise to computational results.
3. *ADMs* to use for the fit. Again these may be from computational results, or set manually. If not specified these will default to an isotropic distribution, which may be appropriate in some cases.

In the following subsections each aspect of the configuration is illustrated.

```
# Initiation - a PEMtk fitting class object

# Set data dir
dataPath = Path.cwd(), dataName)

# Init class object
data = pemtkFit(fileBase = dataPath, verbose = 1)

# Read data files
data.scanFiles()
```

```
*** Job subset details
Key: subset
```

(continues on next page)

(continued from previous page)

```
No 'job' info set for self.data[subset].

*** Job orb6 details
Key: orb6
Dir /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/n2fitting, ↵
  ↵ 1 file(s).
{   'batch': 'ePS n2, batch n2_1pu_0.1-50.1eV, orbital A2',
    'event': ' N2 A-state (1piu-1)',
    'orbE': -17.09691397835426,
    'orbLabel': '1piu-1'}

*** Job orb5 details
Key: orb5
Dir /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/n2fitting, ↵
  ↵ 1 file(s).
{   'batch': 'ePS n2, batch n2_3sg_0.1-50.1eV, orbital A2',
    'event': ' N2 X-state (3sg-1)',
    'orbE': -17.34181645456815,
    'orbLabel': '3sg-1'}
```

10.2.1 Alignment distribution moments (ADMs)

The class `wraps ep.setADMs()` to set *ADMs* to the class data structure. This returns an isotropic distribution by default, or values can be set explicitly from a list. Note: if this is not set, the default value will be used, which is likely not very useful for the fit!

Values are set in `self.data['ADM']`, see Sect. 6.5 for more details on *ADMs* and molecular alignment. For the N_2 example case, the alignment data is as per the original experimental demonstration of the *bootstrap retrieval protocol* [1], and also available from the associated data repository [143].

```
# Default case - isotropic
data.setADMs()
# data.ADM['ADMX']
data.data['ADM']['ADM']
```

```
<xarray.DataArray 'ADM' (ADM: 1, t: 1)>
array([[1]])
Coordinates:
 * ADM      (ADM) MultiIndex
   - K      (ADM) int64 0
   - Q      (ADM) int64 0
   - S      (ADM) int64 0
 * t      (t) int64 0
Attributes:
  dataType:  ADM
  long_name: Axis distribution moments
  units:     arb
```

Note: Full tabulations of the parameters, and some plots, available in HTML or notebook formats only.

```
# Load time-dependent ADMs for N2 case
# These are in a Matlab/HDF5 file format
from scipy.io import loadmat
ADMdataFile = os.path.join(dataPath, 'N2 ADM VM_290816.mat')
ADMs = loadmat(ADMdataFile)

# Set tOffset for calcs, 3.76ps!!!
# This is because this is 2-pulse case, and will set t=0 to 2nd pulse
# (and matches defn. in N2 experimental paper)
# Marceau, C. et al. (2017) 'Molecular Frame Reconstruction Using Time-Domain
# Photoionization Interferometry', Physical Review Letters, 119(8), p. 083401.
# Available at: https://doi.org/10.1103/PhysRevLett.119.083401.
tOffset = -3.76
ADMs['time'] = ADMs['time'] + tOffset

data.setADMs(ADMs = ADMs['ADM'], t=ADMs['time'].squeeze(), KQSLabels = ADMs['ADMlist'],
            addS = True)
data.data['ADM']['ADM']
```

```
<xarray.DataArray 'ADM' (ADM: 4, t: 3691)>
array([[ 1.00000000e+00+0.00000000e+00j,  1.00000000e+00+0.00000000e+00j,
       1.00000000e+00+0.00000000e+00j, ...,
       1.00000000e+00+0.00000000e+00j,  1.00000000e+00+0.00000000e+00j,
       1.00000000e+00+0.00000000e+00j],
      [-2.26243113e-17+0.00000000e+00j,  2.43430608e-08+1.04125246e-20j,
       9.80188266e-08+6.89166168e-20j, ...,
       1.05433798e-01-1.62495135e-18j,  1.05433798e-01-1.62495135e-18j,
       1.05433798e-01-1.62495135e-18j],
      [ 1.55724057e-16+0.00000000e+00j, -3.37021111e-10-6.81416260e-20j,
       1.95424253e-10-3.10513374e-19j, ...,
       8.39913132e-02-5.12795441e-17j,  8.39913132e-02-5.12795441e-17j,
       8.39913132e-02-5.12795441e-17j],
      [-7.68430227e-16+0.00000000e+00j, -1.40177466e-11+1.04987400e-19j,
       6.33419102e-10+1.74747003e-18j, ...,
       3.78131657e-02+4.01318983e-16j,  3.78131657e-02+4.01318983e-16j,
       3.78131657e-02+4.01318983e-16j]])
```

Coordinates:

- * ADM (ADM) MultiIndex
- K (ADM) int64 0 2 4 6
- Q (ADM) int64 0 0 0 0
- S (ADM) int64 0 0 0 0
- * t (t) float64 -3.76 -3.76 -3.76 -3.759 -3.759 ... 10.1 10.1 10.1 10.1

Attributes:

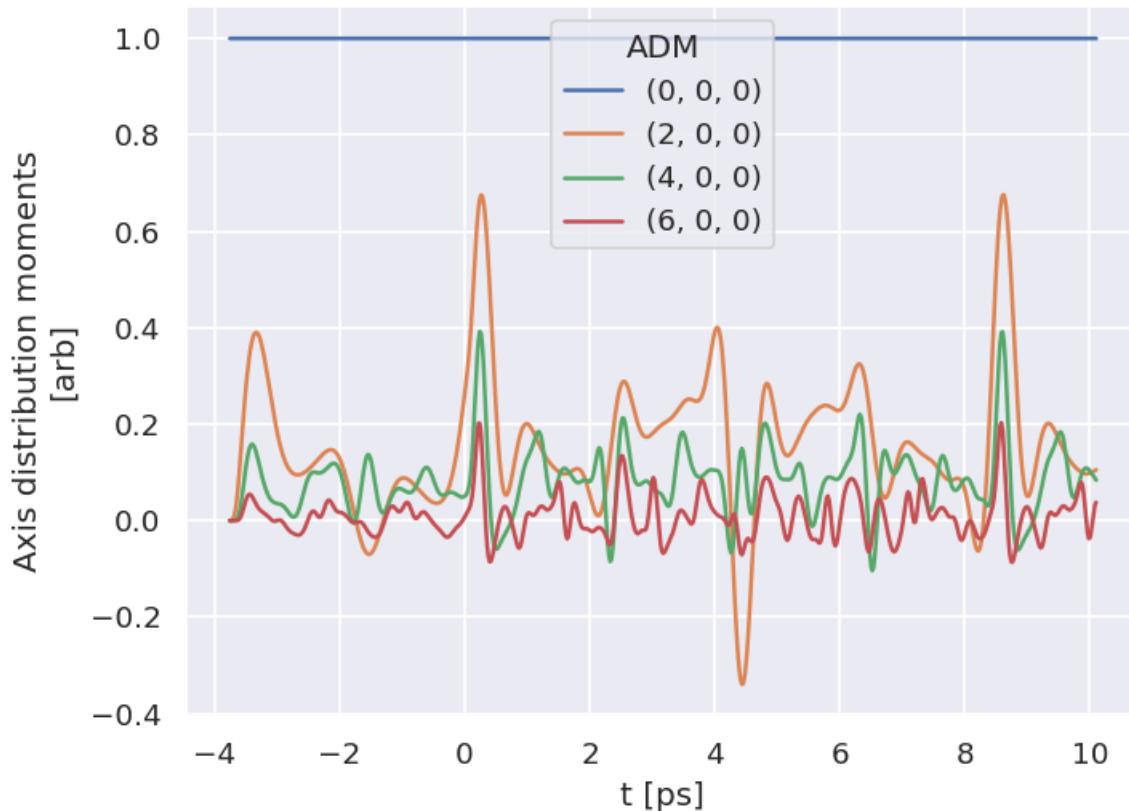
- dataType: ADM
- long_name: Axis distribution moments
- units: arb

```
# Manual plot with hvplot for full control and interactive plot
# NOTE: HTML version only.
key = 'ADM'
dataType='ADM'
data.data[key][dataType].unstack().real.hvplot.line(x='t').overlay(['K','Q','S'])
```

```
:NdOverlay  [S,Q,K]
:Curve     [t]    (ADM)
```

```
# A basic self.ADMplot routine is also available
%matplotlib inline
data.ADMplot(keys = 'ADM')
```

Dataset: ADM, ADM



10.2.2 Polarisation geometry/ies

This wraps `ep.setPolGeoms`. This defaults to (x,y,z) polarization geometries. Values are set in `self.data['pol']`.

Note: if this is not set, the default value will be used, which is likely not very useful for the fit!

```
data.setPolGeoms()
data.data['pol']['pol']
```

```
<xarray.DataArray (Labels: 3)>
array([quaternion(1, -0, 0, 0),
       quaternion(0.707106781186548, -0, 0.707106781186547, 0),
       quaternion(0.5, -0.5, 0.5, 0.5)], dtype=quaternion)
Coordinates:
  Euler      (Labels) object (0.0, 0.0, 0.0) ... (1.5707963267948966, 1.57079...
  * Labels    (Labels) <U32 'z' 'x' 'y'
Attributes:
  dataType: Euler
```

10.2.3 Subselect data

Currently handled in the class by setting `self.selOpts`, this allows for simple reuse of settings as required. Subselected data is set to `self.data['subset'][dataType]` by default (equivalently `self.data[self.subKey][dataType]`), and is the data the fitting routine will use.

```
# Settings for type subselection are in selOpts[dataType]

# E.g. Matrix element sub-selection
data.selOpts['matE'] = {'thres': 0.01, 'inds': {'Type':'L', 'Eke':1.1}}
data.setSubset(dataKey = 'orb5', dataType = 'matE') # Subselect from 'orb5' dataset, ↴matrix elements

# And for the polarisation geometries...
data.selOpts['pol'] = {'inds': {'Labels': 'z'}}
data.setSubset(dataKey = 'pol', dataType = 'pol')

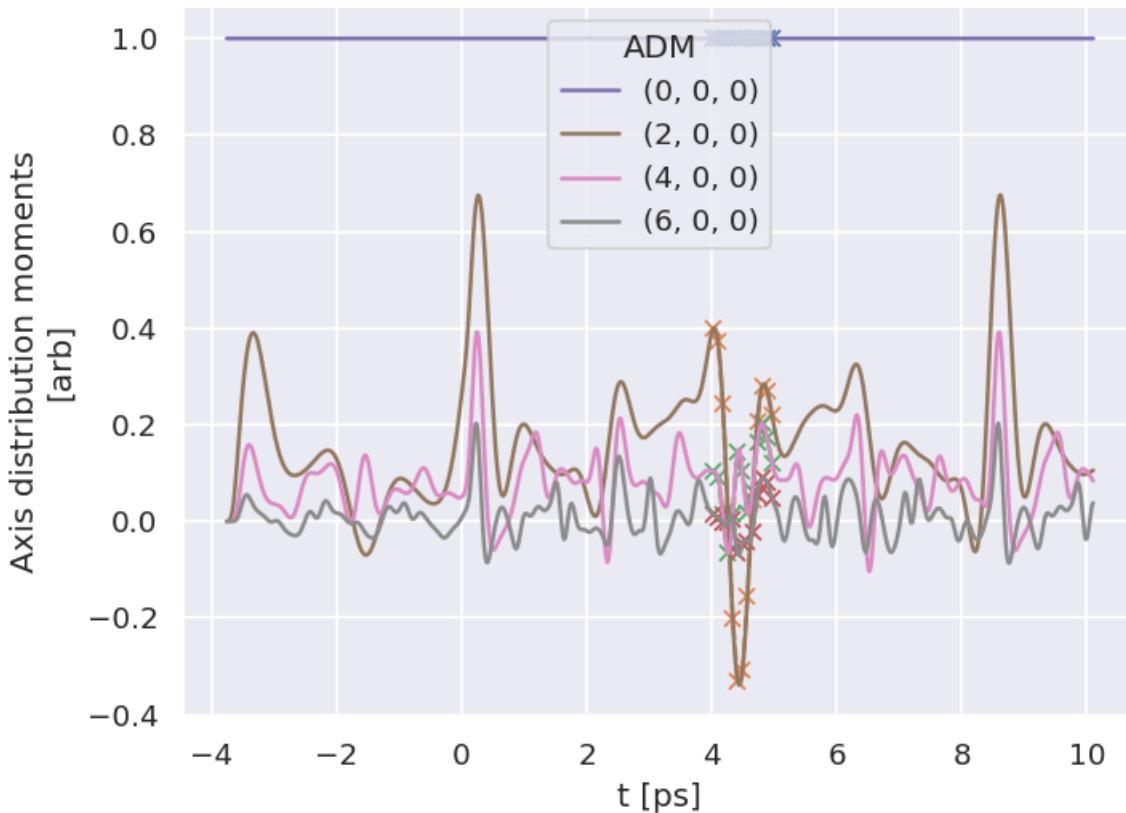
# And for the ADMs...
data.selOpts['ADM'] = {} #{'thres': 0.01, 'inds': {'Type':'L', 'Eke':1.1}}
data.setSubset(dataKey = 'ADM', dataType = 'ADM', sliceParams = {'t':[4, 5, 4]})
```

```
Subselected from dataset 'orb5', dataType 'matE': 36 from 11016 points (0.33%)
Subselected from dataset 'pol', dataType 'pol': 1 from 3 points (33.33%)
Subselected from dataset 'ADM', dataType 'ADM': 52 from 14764 points (0.35%)
```

```
# Note that the class uses data.subKey to reference the correct data internally
print(f'Data dict key: {data.subKey}')
print(f'Data dict contents: {data.data[data.subKey].keys() }')
```

```
Data dict key: subset
Data dict contents: dict_keys(['matE', 'pol', 'ADM'])
```

```
# Check subselected ADMs by plotting vs. full ADM data
# Plot from Xarray vs. full dataset
data.data['subset']['ADM'].real.squeeze().plot.line(x='t', marker = 'x', linestyle= ↴'dashed');
data.data['ADM']['ADM'].real.squeeze().plot.line(x='t');
```



10.3 Compute $\text{AF-}\beta_{LM}$ and simulate data

With all the components set, some observables can be calculated. For testing, this will also be used to simulate an experimental trace (with noise added).

For both basic computation, and fitting, the class method `self.afblmMatEfit()` can be used. This essentially wraps the main AF computational routine, `epsproc.afblmXprod()`, to compute $\text{AF-}\beta_{LM}$ s (for more details, see the [ePSproc method development docs](#) and [API docs](#)).

If called without reference data, the method returns computed $\text{AF-}\beta_{LM}$ s based on the input subsets already created, and also a set of (product) basis functions generated - as illustrated in Sect. 6.3, these can be examined to get a feel for the sensitivity of the geometric part of the problem, and will also be used as a basis in the fitting routine to limit repetitive computations.

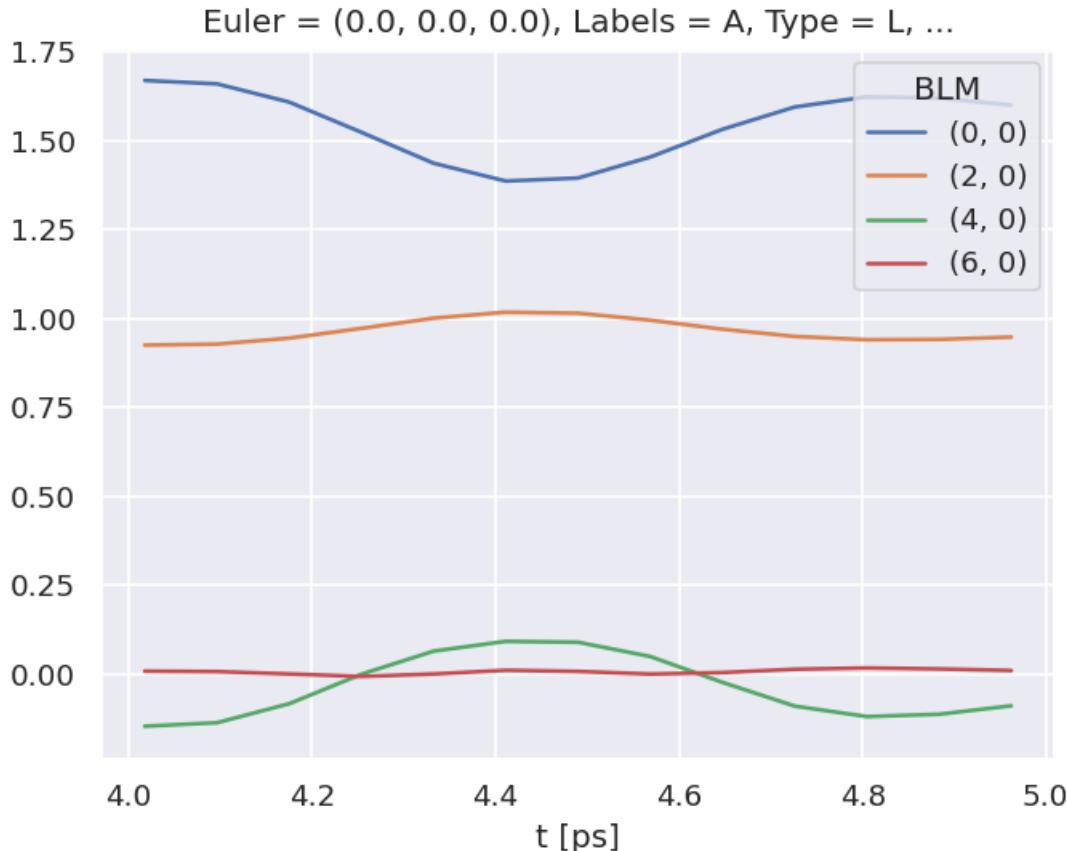
10.3.1 Compute $\text{AF-}\beta_{LM}$ s

```
# Compute with class method
# This uses all data as set to self.data['subset']
BetaNormX, basis = data.afblmMatEfit()
```

10.3.2 AF- β_{LM} s

The returned objects contain the β_{LM} parameters as an Xarray...

```
# Line-plot with Xarray/Matplotlib
# Note there is no filtering here, so this includes some invalid and null terms
BetaNormX.sel(Labels='A').real.squeeze().plot.line(x='t');
```



... and the basis sets as a dictionary. (See Sect. 6.3 for more details on the basis sets.)

```
basis.keys()
```

```
dict_keys(['BLMtableResort', 'polProd', 'phaseConvention', 'BLMRenorm'])
```

10.4 Fitting the data: configuration

As discussed in Chpt. 7, general non-linear fitting approaches are used for the *bootstrap retrieval protocol*. These are wrapped in the Photoelectron Metrology Toolkit [5] for *radial matrix elements* retrieval problems, as shown below. (And, as discussed in Sect. 5.3, make use of the `lmfit` library [66, 67] and `Scipy` [52] base routines.)

10.4.1 Set the data to fit

To use the values calculated above as the test data, it currently needs to be set as `self.data['subset']['AFBLM']` for fitting.

```
# Set computed results to main data structure

# Method 1: Set directly by manual assignment
# data.data['subset']['AFBLM'] = BetaNormX

# Method 2: Set to main data structure and subset using methods as above
# Set simulated data to master structure as "sim"
data.setData('sim', BetaNormX)
# Set to 'subset' to use for fitting.
data.setSubset('sim', 'AFBLM')
```

Subselected from dataset 'sim', dataType 'AFBLM': 52 from 52 points (100.00%)

```
# Set basis functions
data.basis = basis
```

10.4.2 Adding noise

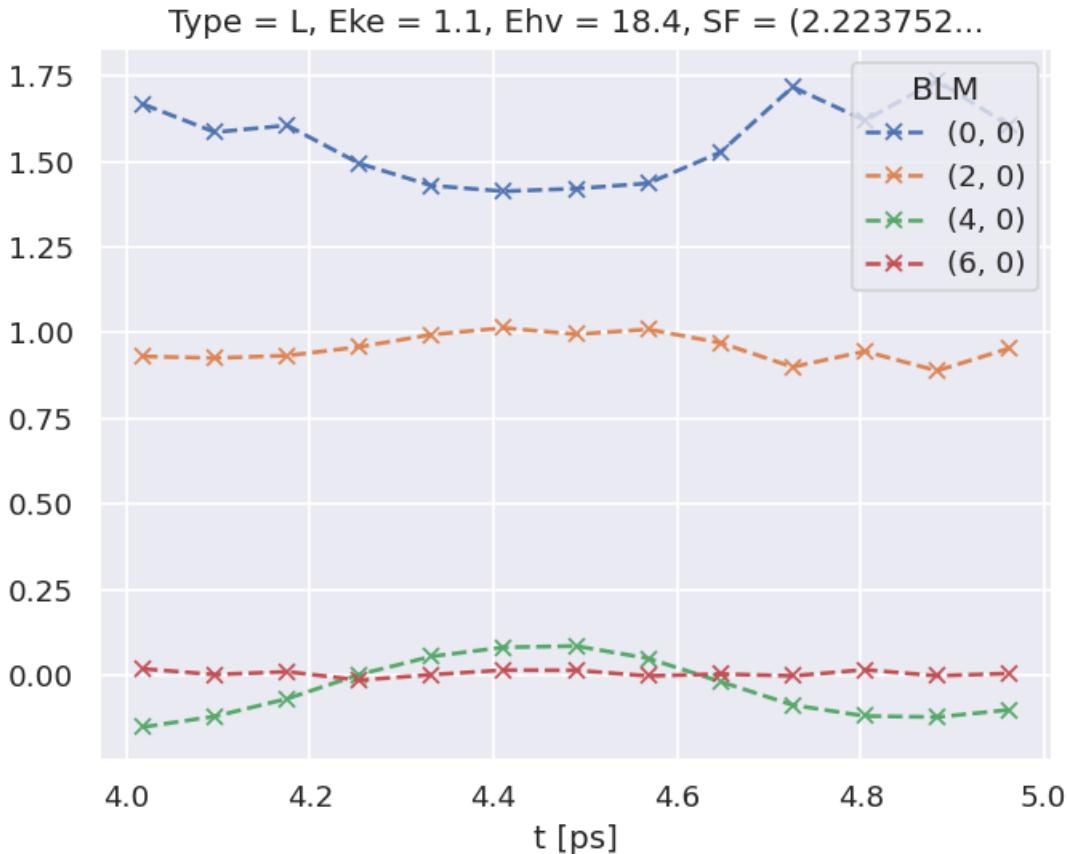
For a more realistic test of the *bootstrap retrieval protocol*, noise or other artifacts can be added to the data. Below is a routine for adding random (Gaussian) noise.

```
# Add noise with np.random.normal
# https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html
import numpy as np
mu, sigma = 0, 0.05 # Up to approx 10% noise (+/- 0.05)
# creating a noise with the same dimension as the dataset (2,2)
noise = np.random.normal(mu, sigma,
                         [data.data['subset']['AFBLM'].t.size,
                          data.data['subset']['AFBLM'].l.size])
# data.BLMfitPlot()

# Set noise in Xarray & scale by l
import xarray as xr
noiseXR = xr.ones_like(data.data['subset']['AFBLM']) * noise
# Scale by L? This prevents too much high-order noise
noiseXR = noiseXR.where(noiseXR.l<1, noiseXR/(noiseXR.l))
# Update data for fitting
data.data['subset']['AFBLM'] = data.data['subset']['AFBLM'] + noiseXR
# Remove non-zero m terms?
# This removes additional noise-only channels
data.data['subset']['AFBLM'] = data.data['subset']['AFBLM'].where(data.data['subset'][
    'AFBLM'].m == 0, 0)

# The BLMfitPlot() routine can be used to plot data and fit outputs
data.BLMfitPlot()
```

Dataset: subset, AFBLM



10.4.3 Setting up the fit parameters

As detailed in Chpt. 9, fitting requires a basis set and fit parameters. In this case, we can work from the existing matrix elements (as used for simulating data above) to speed up parameter creation, although in practice this may need to be approached *ab initio* or via symmetry - nonetheless, the method will be similar.

```
# Set matrix elements from ab initio results
data.setMatEFit(data.data['subset']['matE'])
```

This sets `self.params` from the matrix elements, which are a set of (real) parameters for lmfit, as a `Parameters` object.

Note that:

- The input matrix elements are converted to magnitude-phase form, hence there are twice the number as the input array, and labelled m or p accordingly, along with a name based on the full set of QNs/indexes set.
- One phase is set to `vary=False`, which defines a reference phase. This defaults to the first phase item.
- Min and max values are defined, by default the ranges are $1e^{-4} < \text{mag} < 5$, $-\pi < \text{phase} < \pi$.
- Relationships between the parameters are set by default, but can be set manually, or pass `paramsCons=None` to skip.

For further details, including modification of parameter settings, see Sect. 9.3.4 and the PEMtk documentation [20].

10.4.4 Quick setup with script

The steps demonstrated above are also wrapped in a helper script, although some steps may need to be re-run to change selection properties or ranges. For the case studies, there are specific details for each configured in the script, including source data locations and the selection criteria as used in each demonstration.

```
# Run general config script with dataPath set above
%run {dataPath/"setup_fit_demo.py"} -d {dataPath}
```

10.5 Fitting the data: Running fits

10.5.1 Single fit

With the parameters and data set, just call `self.fit()`! For more control, options to the `lmfit` library [66, 67] minimizer function can be set. Statistics and outputs are also handled by the `lmfit` library [66, 67], which includes uncertainty estimates and correlations in the fitted parameters.

```
# Run a fit
# data.randomizeParams() # Randomize input parameters if desired
# For method testing using known initial params is also_
# useful

# Run fit with defaults settings
data.fit()

# Additional keyword options can be pass, these are passed to the fitting routine.
# Args are passed to the lmfit minimizer, see https://lmfit.github.io/lmfit-py/
# fitting.html
# E.g. for scipy Least Squares, options can be found at
# https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares
# html#scipy.optimize.least_squares
# For example, pass convergence tolerances
# data.fit(ftol=1e-10, xtol=1e-10)
```

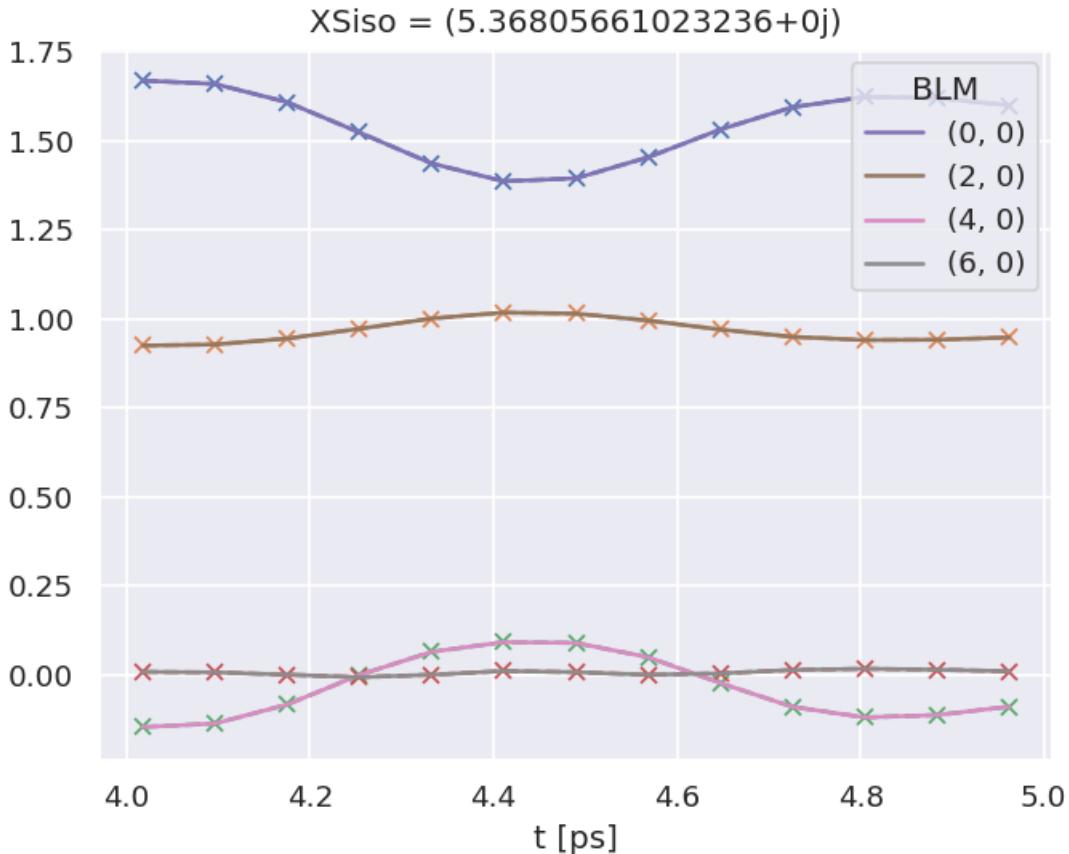
```
# Check fit outputs - self.result shows results from the last fit
data.result
```

```
# Fit data is also set to the master data structure with an integer key
data.data.keys()
```

```
dict_keys(['subset', 'orb6', 'orb5', 'ADM', 'pol', 'sim', 0])
```

```
# Plot results with data overlay
# data.BLMfitPlot(backend='hv') # Set backend='hv' for interactive plots
data.BLMfitPlot()
```

```
Dataset: subset, AFBLM
Dataset: 0, AFBLM
```



10.5.2 Extended execution methods, including parallel and batched execution

See the PEMtk documentation [20] for details, particularly the [batch runs demo page](#).

(1) serial execution

Either:

- Manually with a loop.
- With `self.multiFit()` method, although this is optimised for parallel execution (see below).

```
# Basic serial example with a loop
import time

start = time.time()

# Manual execution
for n in range(0,10):
    data.randomizeParams()
    data.fit()

end = time.time()
print((end - start)/60)

# Or run with self.multiFit(parallel = False)
# data.multiFit(nRange = [0,10], parallel = False)
```

```
4.699940152963003
```

```
# There are now 10 more fit results
data.data.keys()
```

```
dict_keys(['subset', 'orb6', 'orb5', 'ADM', 'pol', 'sim', 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

(b) parallel execution

A basic parallel fitting routine is implemented via the `self.multiFit()` method. This currently uses the `xyzpy` library [68] for quick parallelization, although there is some additional setup overhead in the currently implementation due to class init per fit batch. The default settings aims to set ~90% CPU usage, based on core-count.

```
# Multifit wrapper with range of fits specified
# Set 'num_workers' to override the default (~90% of available cores).
data.multiFit(nRange = [0,10], num_workers=20)
```

(c) Dump data

Various options are available. The most complete is to use Pickle (default case), which dumps the entire `self.data` structure to file, although this is not suggested for archival use. For details see the ePSproc documentation [35], particularly the [data structures demo page](#). For some data types HDF5 routines are available, and are demonstrated for post-processed fit data in the case studies (Chapt. 11 - Chapt. 13).

```
outStem = 'dataDump_N2' # Set for file save later
# Minimal case - timestamped filename
# data.writeFitData()

# Use ' fName' to supply a filename
# data.writeFitData(fName='N2_datadump')

# Use 'outStem' to define a filename which will be appended with a timestamp
# Set dataPath if desired, otherwise will use working dir
data.writeFitData(dataPath = dataPath, outStem=outStem)
```

```
PosixPath('/home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
n2fitting/dataDump_N2_310723_16-52-15.pickle')
```

10.5.3 Batch fit with sampling options

From the basic methods above, more sophisticated fitting strategies can be built. For example, the cell below implements batched fitting with Poission sampling of the data (for statistical bootstrapping).

```
# Batch fit with data weighting example
batchSize = 50

data.data['weights'] = {} # Use to log ref weights, will be overwritten otherwise

for n in np.arange(0,100,batchSize):
    print(f'Running batch [{n},{n+batchSize-1}]')
```

(continues on next page)

(continued from previous page)

```
# Reset weights
data.setWeights(wConfig = 'poission', keyExpt='sim')
data.setSubset('sim','weights')    # Set to 'subset' to use for fitting.

data.data['weights'][n] = data.data['sim']['weights'].copy()

# Run fit batch
data.multiFit(nRange = [n,n+batchSize-1], num_workers=20)

# Checkpoint - dump data to file
data.writeFitData(outStem=outStem)
```

CASE STUDY: GENERALISED BOOTSTRAPPING FOR A HOMONUCLEAR DIATOMIC SCATTERING SYSTEM, N_2 ($D_{\infty H}$)

In this chapter, the full code and analysis details of the case study for N_2 are given, including obtaining required data, running fits and analysis routines. For more details on the routines, see the [PEMtk documentation \[20\]](#); for the analysis see particularly the [fit fidelity](#) and [analysis](#) page, and [molecular frame analysis data processing](#) page (full analysis for Ref. [3], illustrating the N_2 case).

11.1 General setup

In the following code cells (see source notebooks for full details) the general setup routines (as per the outline in Chpt. 10) are executed via a configuration script with presets for the case studies herein.

Additionally, the routines will either run fits, or load existing data if available. Since fitting can be computationally demanding, it is, in general, recommended to approach large fitting problems carefully.

General note on fitting

Computational outputs in this chapter are significantly truncated in the PDF, and some simplified plots are used; see source notebooks (via [Quantum Metrology Vol. 3 \(Github repo\)](#)) or [Quantum Metrology Vol. 3 \(HTML version\)](#) for full details.

```
# Configure settings for case study

# Set case study by name
fitSystem='N2'
fitStem=f"fit_withNoise_orb5"

# Add noise?
addNoise = 'y'
mu, sigma = 0, 0.05 # Up to approx 10% noise (+/- 0.05)

# Batching - number of fits to run between data dumps
batchSize = 10

# Total fits to run
nMax = 10
```

```
# Pull data from web (N2 case)

from epsproc.util.io import getFilesFromGithub

# Set dataName (will be used as download subdir)
dataName = 'n2fitting'
# N2 matrix elements
fDictMatE, fAllMatE = getFilesFromGithub(subpath='data/photoionization/n2_multiorb',_
    ↴dataName=dataName)
# N2 alignment data
fDictADM, fAllMatADM = getFilesFromGithub(subpath='data/alignment', dataName=dataName)
```

```
# Fitting setup including data generation and parameter creation

# Set datapath,
dataPath = Path(Path.cwd(),dataName)

# Run general config script with dataPath set above
%run "../scripts/setup_fit_case-studies_270723.py" -d {dataPath} -c {fitSystem} -n
    ↴{addNoise} --sigma {sigma}
```

11.2 Load existing fit data or run fits

Note that running fits may be quite time-consuming and computationally intensive, depending on the size of the problem. The default case here will run a small batch for testing if there is no existing data found on the `dataPath`, otherwise the data is loaded for analysis.

```
# Look for existing Pickle files on path?
# dataFiles = list(dataPath.expanduser().glob('*.pickle'))
dataFiles = [Path(dataPath.expanduser(), 'N2_1199_fit_withNoise_orb5_280723_11-39-26.
    ↴pickle')] # Set reference dataset(s)

if not dataFiles:
    print("No data found, executing minimal fitting run...")

    # Run fit batch - single
    # data.multiFit(nRange = [n,n+batchSize-1], num_workers=batchSize)

    # Run fit batches with checkpoint files
    for n in np.arange(0,nMax,batchSize):
        print(f'*** Running batch [{n},{n+batchSize-1}], {dt.now().strftime("%d%m%y-
            ↴%H-%M-%S")}!')

        # Run fit batch
        data.multiFit(nRange = [n,n+batchSize-1], num_workers=batchSize)

        # Dump data so far
        data.writeFitData(outStem=f'{fitSystem}_{n+batchSize-1}_{fitStem}')

        print(f'Finished batch [{n},{n+batchSize-1}], {dt.now().strftime("%d%m%y_%H-
            ↴%M-%S")}')
        print(f'Written to file {fitSystem}_{n+batchSize-1}_{fitStem}'')
```

(continues on next page)

(continued from previous page)

```

else:
    dataFileIn = dataFiles[-1]      # Add index to select file, although loadFitData_
    ↪will concat multiple files          # Note that concat currently only works for fixed_
    ↪batch sizes however.
    print(f"Set dataFiles: {dataFileIn}")
    data.loadFitData(fList=dataFileIn, dataPath=dataPath)    #.expanduser())

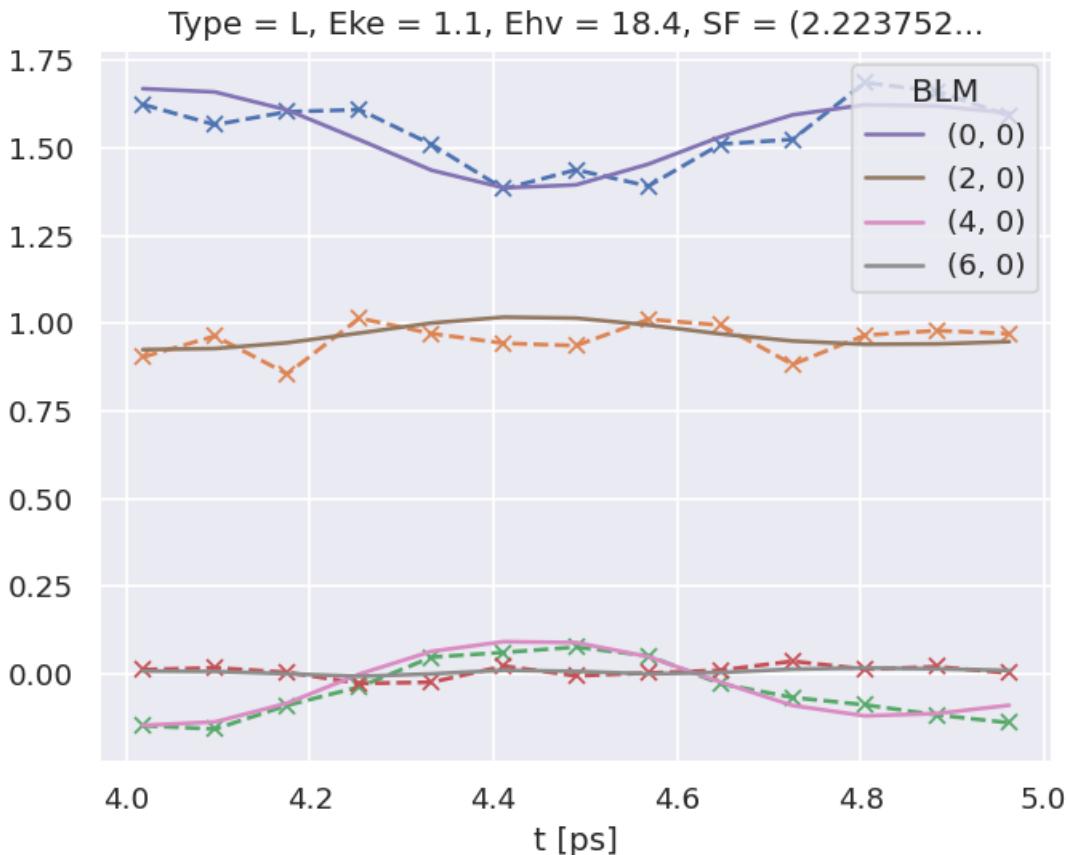
    data.BLMfitPlot(keys=['subset','sim'])

```

```

Set dataFiles: /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪n2fitting/N2_1199_fit_withNoise_orb5_280723_11-39-26.pickle
Read data from /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪n2fitting/N2_1199_fit_withNoise_orb5_280723_11-39-26.pickle with pickle.
Dataset: subset, AFBLM
Dataset: sim, AFBLM

```

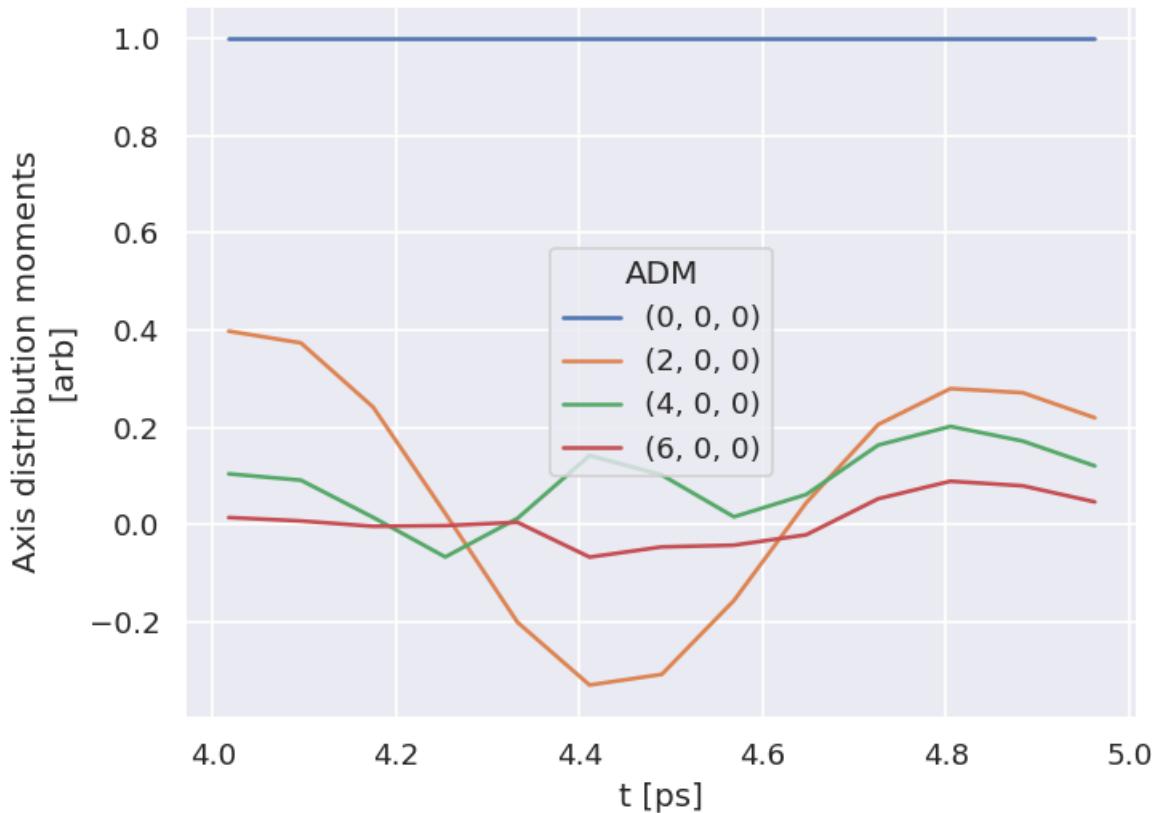


```

# Check ADMs
# Basic plotter
data.ADMplot(keys = 'subset')

```

```
Dataset: subset, ADM
```



```
# Fits appear as integer indexed items in the main data structure.
data.data.keys()
```

11.3 Post-processing and data overview

Post-processing involves aggregation of all the fit run results into a single data structure. This can then be analysed statistically and examined for best-fit results. In the statistical sense, this is essentially a search for candidate *radial matrix elements*, based on the assumption that some of the minima found in the χ^2 hyperspace will be the true results. Even if a clear global minima does not exist, searching for candidate *radial matrix elements* sets based on clustering of results and multiple local minima is still expected to lead to viable candidates provided that the information content of the dataset is sufficient. However, as discussed elsewhere (see Sect. 7.2), in some cases this may not be the case, and other limitations may apply (e.g. certain parameters may be undefined), or additional data required for unique determination of the *radial matrix elements*.

For more details on the analysis routines, see the PEMtk documentation [20], particularly the [fit fidelity](#) and [analysis](#) page, and [molecular frame analysis](#) data processing page (full analysis for Ref. [3], illustrating the N_2 case).

```
# General stats & post-processing to data tables
data.analyseFits()
```

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
  ↵with Pandas MultiIndex, but will try anyway.
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
  ↵with Pandas MultiIndex, but will try anyway.
```

(continues on next page)

(continued from previous page)

```
{ 'Fits': 1170,
  'Minima': { 'chisqr': 0.07137430989515783, 'redchi': 0.000479022214061462},
  'Stats': { 'chisqr': min 0.071
mean 0.084
median 0.071
max 2.594
std 0.089
var 0.008
Name: chisqr, dtype: float64,
      'redchi': min 4.790e-04
mean 5.652e-04
median 4.790e-04
max 1.741e-02
std 6.002e-04
var 3.602e-07
Name: redchi, dtype: float64},
  'Success': 1169}
```

```
# The BLMsetPlot routine will output aggregate fit results.
# Here the spread can be taken as a general indication of the uncertainty of
# the fitting, and indicate whether the fit is well-characterised/the information
# content of the data is sufficient.
data.BLMsetPlot(xDim = 't', thres=1e-6) # With xDim and thres set, for more control
# over outputs

# Glue plot for later
glue("N2-fitResultsBLM", data.data['plots']['BLMsetPlot'])
```

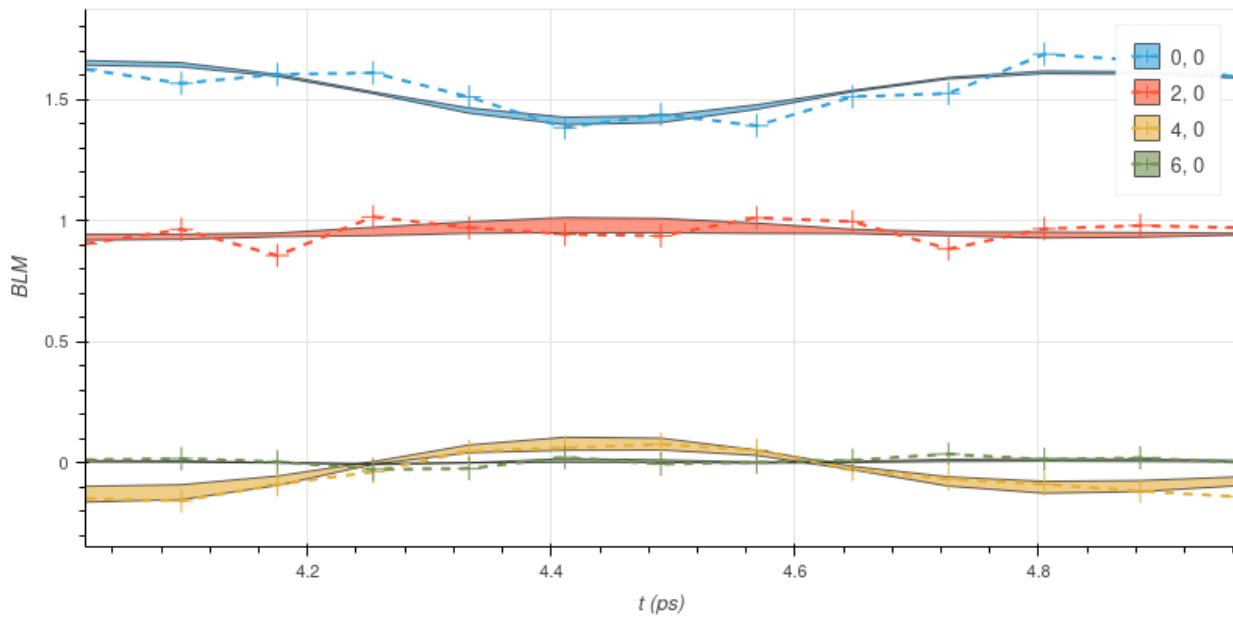


Fig. 11.1: Fit overview plot - $\beta_{L,M}(t)$. Here dashed lines with '+' markers indicates the input data, and bands indicate the mean fit results, where the width is the standard deviation in the fit model results. (See the PEMtk documentation [20] for details, particularly the analysis routines page.)

```
# Write aggregate datasets to HDF5 format
# This is more robust than Pickled data, but PEMtk currently only support output for_
# aggregate (post-processed) fit data.

data.processedToHDF5(dataPath = dataPath, outStem = dataFileIn.name, timeStamp=False)
```

```
Dumped self.data[fits][dfLong] to /home/jovyan/jake-home/buildTmp/_latest_build/
↳pdf/doc-source/part2/n2fitting/N2_1199_fit_withNoise_orb5_280723_11-39-26.pickle_
↳dfLong.pdHDF with Pandas .to_hdf() routine.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/N2_1199_fit_withNoise_orb5_280723_11-39-26.pickle_dfLong.pdHDF with_
↳pdHDF.
Dumped self.data[fits][AFxr] to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/
↳doc-source/part2/n2fitting/N2_1199_fit_withNoise_orb5_280723_11-39-26.pickle_
↳AFxr.pdHDF with Pandas .to_hdf() routine.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳n2fitting/N2_1199_fit_withNoise_orb5_280723_11-39-26.pickle_AFxer.pdHDF with_
↳pdHDF.
```

```
# Histogram fit results (reduced chi^2 vs. fit index)
# This may be quite slow for large datasets, setting limited ranges may help

# Use default auto binning
# data.fitHist()

# Example with range set
data.fitHist(thres=1.5e-3, bins=100)

# Glue plot for later
glue("N2-fitHist", data.data['plots']['fitHistPlot'])
```

Here bands in the χ^2 dimension can indicate groupings (local minima) are consistently found. Assuming each grouping is a viable fit candidate parameter set, these can then be explored in further detail.

11.4 Data exploration

The general aim in this procedure is to ascertain whether there was a good spread of parameters explored, and a single (or few sets) of best-fit results. There are a few procedures and helper methods for this...

11.4.1 View results

Single results sets can be viewed in the main data structure, indexed by #.

```
# Check keys
fitNumber = 2
data.data[fitNumber].keys()
```

```
dict_keys(['AFBLM', 'residual', 'results'])
```

Here `results` is an `lmFit` object, which includes final fit results and information, and `AFBLM` contains the model (fit) output.

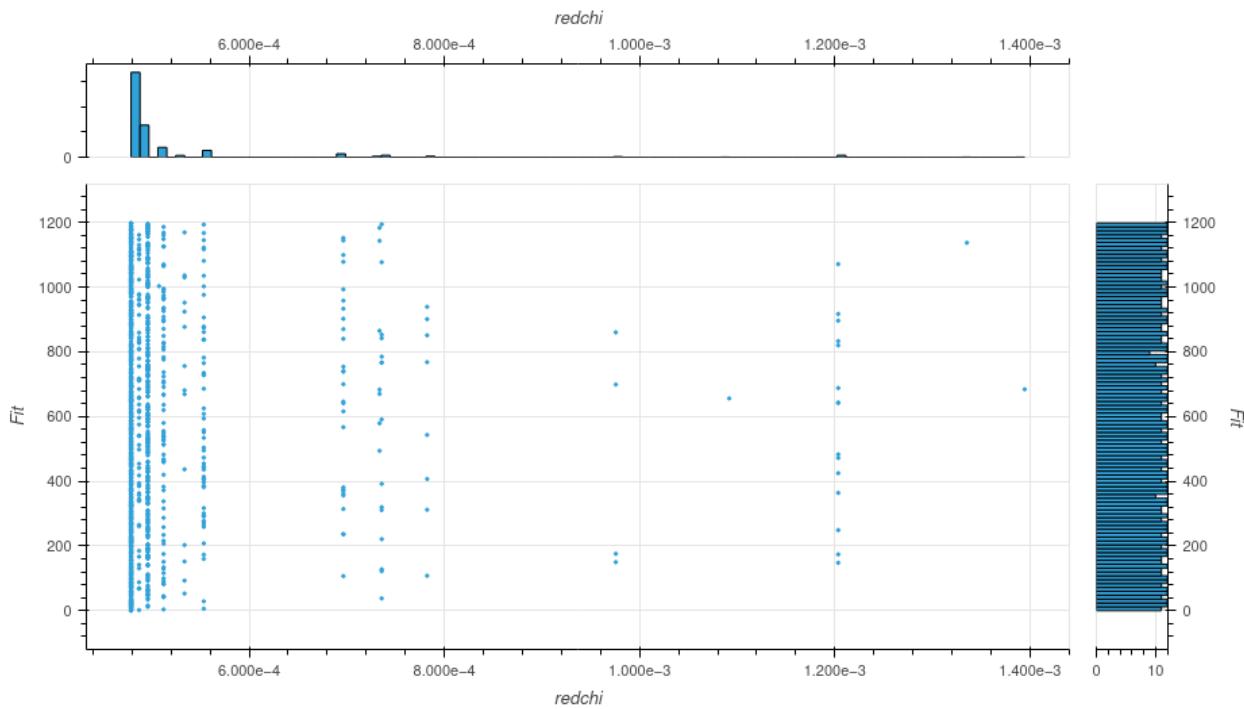


Fig. 11.2: Fit overview plot - χ^2 vs. fit index. Here bands indicate groupings (local minima) are consistently found.

An example is shown below. Of particular note here is which parameters have `vary=True` - these are included in the fitting - and if there is a column `expression`, which indicates any parameters defined to have specific relationships (see Chpt. 9). Any correlations found during fitting are also shown, which can also indicate parameters which are related (even if this is not predefined or known a priori).

```
# Show some results
data.data[fitNumber]['results']
```

11.5 Classify candidate sets

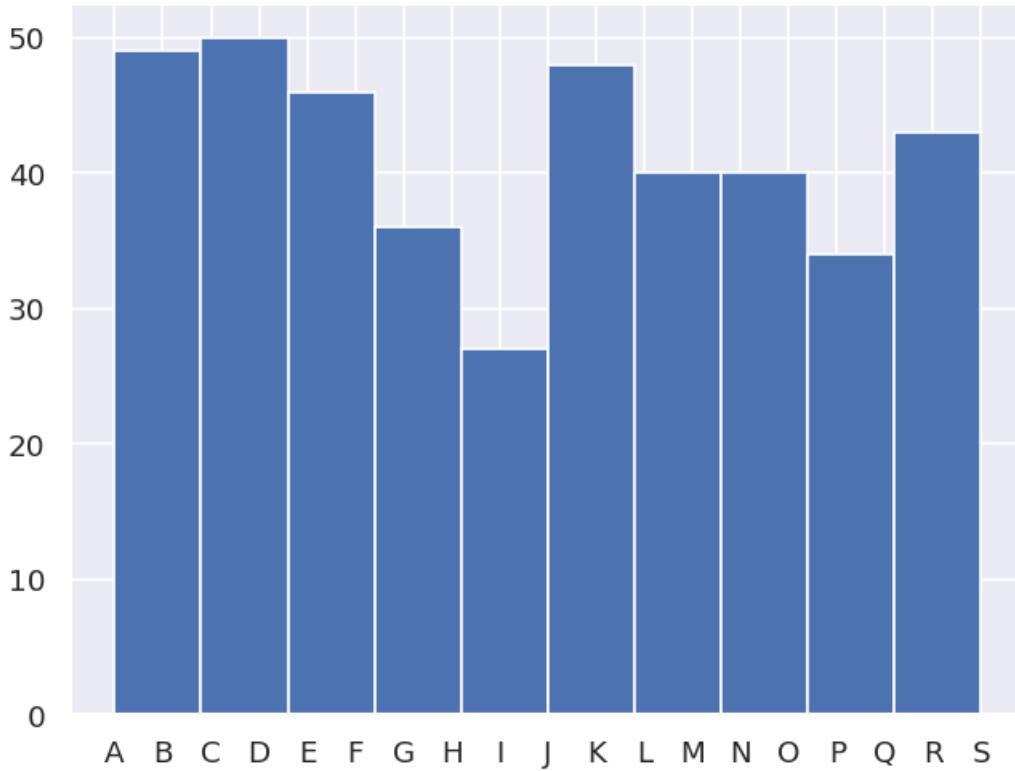
To probe the minima found, the `classifyFits` method can be used. This bins results into “candidate” groups, which can then be examined in detail.

```
# Run with defaults
# data.classifyFits()

# For more control, pass bins
# Here the minima is set at one end, and a %age range used for bins
minVal = data.fitsSummary['Stats']['redchi']['min']
binRangePC = 1e-8
data.classifyFits(bins = [minVal, minVal + binRangePC*minVal , 20])
```

redchiGroup	success				chisqr				redchi			
	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq
A	31	1	True	31	31.0	31.0	0.071	1.0	31.0	31.0	4.790e-04	1.0
B	18	1	True	18	18.0	18.0	0.071	1.0	18.0	18.0	4.790e-04	1.0
C	25	1	True	25	25.0	25.0	0.071	1.0	25.0	25.0	4.790e-04	1.0
D	25	1	True	25	25.0	25.0	0.071	1.0	25.0	25.0	4.790e-04	1.0
E	25	1	True	25	25.0	25.0	0.071	1.0	25.0	25.0	4.790e-04	1.0
F	21	1	True	21	21.0	21.0	0.071	1.0	21.0	21.0	4.790e-04	1.0
G	17	1	True	17	17.0	17.0	0.071	1.0	17.0	17.0	4.790e-04	1.0
H	19	1	True	19	19.0	19.0	0.071	1.0	19.0	19.0	4.790e-04	1.0
I	27	1	True	27	27.0	27.0	0.071	1.0	27.0	27.0	4.790e-04	1.0
J	21	1	True	21	21.0	21.0	0.071	1.0	21.0	21.0	4.790e-04	1.0
K	27	1	True	27	27.0	27.0	0.071	1.0	27.0	27.0	4.790e-04	1.0
L	22	1	True	22	22.0	22.0	0.071	1.0	22.0	22.0	4.790e-04	1.0
M	18	1	True	18	18.0	18.0	0.071	1.0	18.0	18.0	4.790e-04	1.0
N	18	1	True	18	18.0	18.0	0.071	1.0	18.0	18.0	4.790e-04	1.0
O	22	1	True	22	22.0	22.0	0.071	1.0	22.0	22.0	4.790e-04	1.0
P	18	1	True	18	18.0	18.0	0.071	1.0	18.0	18.0	4.790e-04	1.0
Q	16	1	True	16	16.0	16.0	0.071	1.0	16.0	16.0	4.790e-04	1.0
R	20	1	True	20	20.0	20.0	0.071	1.0	20.0	20.0	4.790e-04	1.0
S	23	1	True	23	23.0	23.0	0.071	1.0	23.0	23.0	4.790e-04	1.0

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
with Pandas MultiIndex, but will try anyway.
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
with Pandas MultiIndex, but will try anyway.
```



11.6 Explore candidate result sets

Drill-down on a candidate set of results, and examine values and spreads. For more details see [PEMtk documentation \[20\]](#), especially the [analysis routines page](#). (See also Sect. 5.3 for details on the plotting libraries implemented here.)

11.6.1 Raw results

Plot spreads in magnitude and phase parameters. Statistical plots are available for Seaborn and Holoviews backends, with some slightly different options.

```
# From the candidates, select a group for analysis
selGroup = 'A'

# paramPlot can be used to check the spread on each parameter.
# Plots use Seaborn or Holoviews/Bokeh
# Colour-mapping is controlled by the 'hue' parameter, additionally pass hRound for
# sig. fig control.
# The remap setting allows for short-hand labels as set in data.lmmu

paramType = 'm' # Set for (m)agnitude or (p)hase parameters
hRound = 14 # Set for cmapping, default may be too small (leads to all grey cmap on
# points)

data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
```

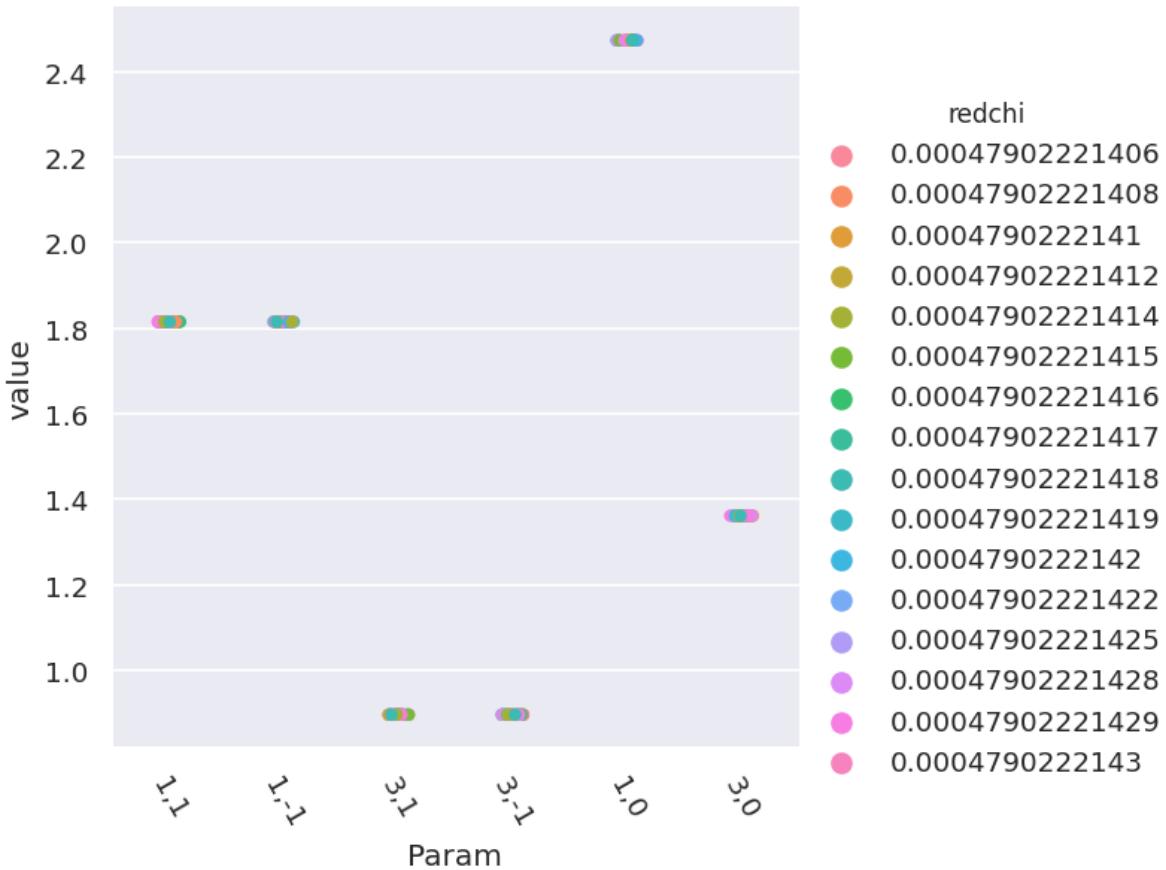
(continues on next page)

(continued from previous page)

```
backend=paramPlotBackend, hvType='violin',
returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
with Pandas MultiIndex, but will try anyway.

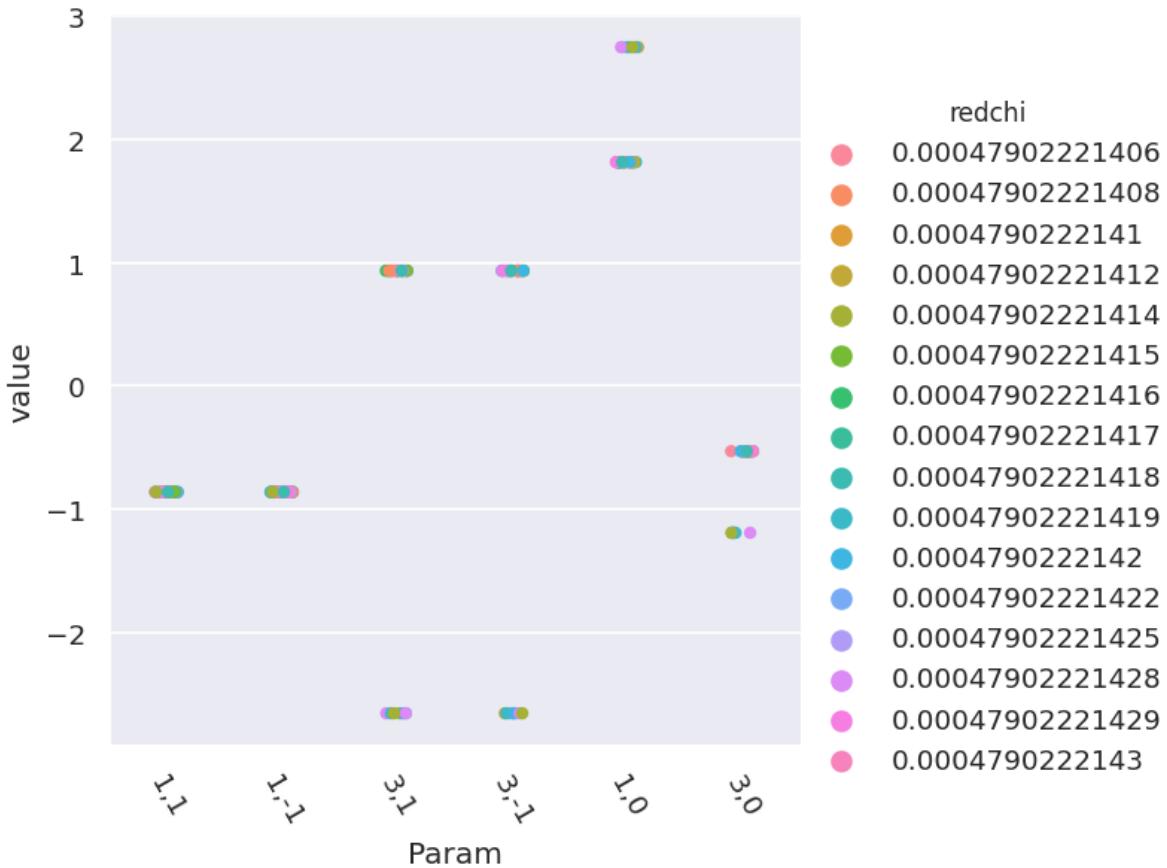
<seaborn.axisgrid.FacetGrid at 0x7f520827eb90>



```
paramType = 'p' # Set for (m)agnitude or (p)hase parameters
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
backend=paramPlotBackend, hvType='violin',
returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
with Pandas MultiIndex, but will try anyway.

<seaborn.axisgrid.FacetGrid at 0x7f520839a440>



11.6.2 Phases, phase shifts & corrections

Depending on how the fit was configured, phases may be defined in different ways. To set the phases relative to a specific parameter, and wrap to a specified range, use the `phaseCorrection()` method. This defaults to using the first parameter as a reference phase, and wraps to $-\pi : \pi$. The phase-corrected values are output to a new Type, 'pc', and a set of normalised magnitudes to 'n'. Additional settings can be passed for more control, as shown below.

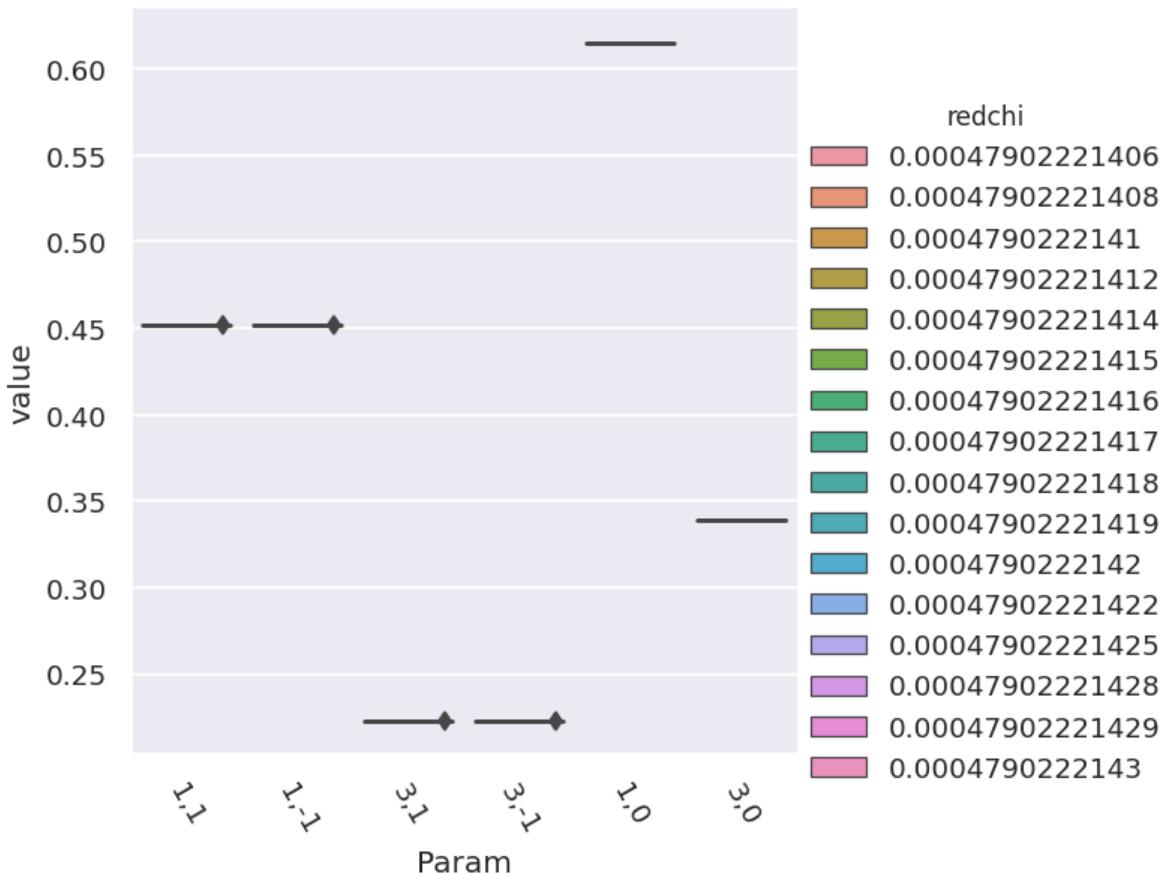
```
# Run phase correction routine
# Set absFlag=True for unsigned phases (mapped to 0:pi)
# Set useRef=False to set ref phase as 0, otherwise the reference value is set.
phaseCorrParams={'absFlag':True, 'useRef':False}
data.phaseCorrection(**phaseCorrParams)
```

Examine new data types...

```
paramType = 'n'
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
               backend=paramPlotBackend, hvType='violin', kind='box',
               returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index – checkDims may have issues with Pandas MultiIndex, but will try anyway.

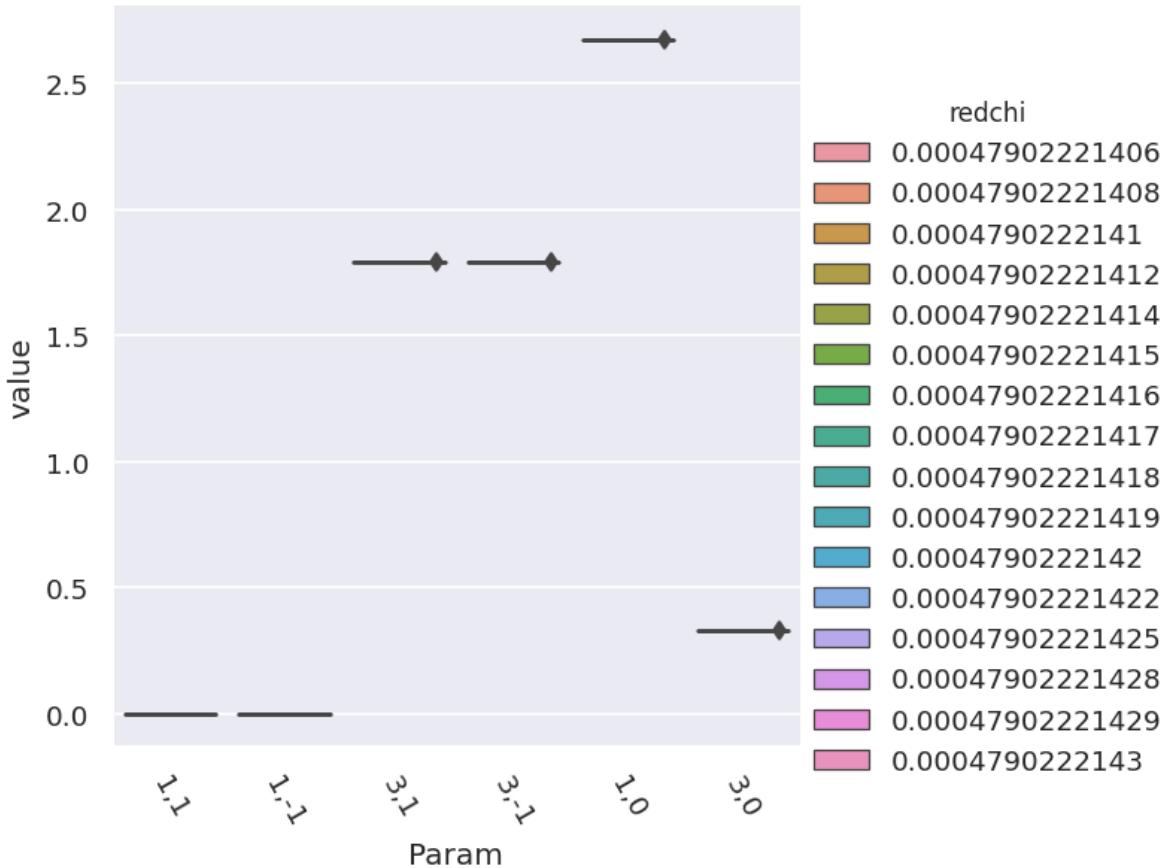
```
<seaborn.axisgrid.FacetGrid at 0x7f520810bf10>
```



```
paramType = 'pc'
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
               backend=paramPlotBackend, hvType='violin', kind='box',
               returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index – checkDims may have issues with Pandas MultiIndex, but will try anyway.

```
<seaborn.axisgrid.FacetGrid at 0x7f52082e38e0>
```



11.7 Parameter estimation & fidelity

For case studies, the fit results can be directly compared to the known input parameters. This should give a feel for how well the data defines the matrix elements (parameters) in this case. In general, probing the correlations and spread of results, and comparing to other (unfitted) results is required to estimate fidelity, see *Quantum Metrology* Vols. 1 & 2 [4, 9] for further discussion.

11.7.1 Best values and statistics

To get a final parameter set and associated statistics, based on a subset of the fit results, the `paramsReport()` method is available. If reference data is available, as for the case studies herein, the `paramsCompare()` method can also be used to compare with the reference case.

```
# Parameter summary
data.paramsReport(inds = {'redchiGroup': selGroup})
```

```
# Parameter comparison
# Note this uses phaseCorrParams as set previously for consistency
data.paramsCompare(phaseCorrParams=phaseCorrParams)
```

```
# Display above results With column name remapping to (l,m) labels only

# With Pandas functionality
data.paramsSummaryComp.rename(columns=data.lmmu['lmMap'])

# With utility method
# summaryRenamed = pemtk.fit._util.renameParams(data.paramsSummaryComp, data.lmmu[
#     ↪'lmMap'])
# summaryRenamed
```

Type	Source	Param dType	1,1	1,-1	3,1	3,-1	1,0	3,0
m	mean	num	1.815e+00	1.815e+00	8.959e-01	8.959e-01	2.473e+00	1.361e+00
	ref	num	1.785e+00	1.785e+00	8.029e-01	8.029e-01	2.686e+00	1.109e+00
	diff	%	1.662e+00	1.662e+00	1.038e+01	1.038e+01	8.617e+00	1.851e+01
		num	3.016e-02	3.016e-02	9.304e-02	9.304e-02	-2.131e-01	2.520e-01
	std	%	1.021e-03	1.021e-03	4.377e-03	4.377e-03	2.025e-03	6.889e-03
		num	1.853e-05	1.853e-05	3.921e-05	3.921e-05	5.007e-05	9.377e-05
	diff/std	%	1.628e+05	1.628e+05	2.373e+05	2.373e+05	4.256e+05	2.688e+05
	n	mean	4.514e-01	4.514e-01	2.229e-01	2.229e-01	6.152e-01	3.386e-01
	ref	num	4.447e-01	4.447e-01	2.001e-01	2.001e-01	6.693e-01	2.764e-01
n	diff	%	1.490e+00	1.490e+00	1.023e+01	1.023e+01	8.807e+00	1.837e+01
		num	6.725e-03	6.725e-03	2.279e-02	2.279e-02	-5.418e-02	6.221e-02
	std	%	1.023e-03	1.023e-03	4.375e-03	4.375e-03	2.021e-03	6.892e-03
		num	4.618e-06	4.618e-06	9.750e-06	9.750e-06	1.243e-05	2.334e-05
	diff/std	%	1.456e+05	1.456e+05	2.338e+05	2.338e+05	4.357e+05	2.666e+05
	p	mean	-8.610e-01	-8.610e-01	1.230e-01	1.230e-01	2.025e+00	-6.796e-01
	ref	num	-8.610e-01	-8.610e-01	-3.120e+00	-3.120e+00	2.611e+00	-7.868e-02
	diff	%	0.000e+00	0.000e+00	2.637e+03	2.637e+03	2.896e+01	8.842e+01
		num	0.000e+00	0.000e+00	3.243e+00	3.243e+00	-5.864e-01	-6.009e-01
p	std	%	0.000e+00	0.000e+00	1.240e+03	1.240e+03	1.957e+01	4.139e+01
		num	0.000e+00	0.000e+00	1.525e+00	1.525e+00	3.963e-01	2.813e-01
	diff/std	%	NaN	NaN	2.126e+02	2.126e+02	1.480e+02	2.136e+02
	pc	mean	0.000e+00	0.000e+00	1.794e+00	1.794e+00	2.675e+00	3.309e-01
	ref	num	0.000e+00	0.000e+00	2.259e+00	2.259e+00	2.811e+00	7.824e-01
	diff	%	NaN	NaN	2.591e+01	2.591e+01	5.067e+00	1.364e+02
		num	0.000e+00	0.000e+00	-4.649e-01	-4.649e-01	-1.356e-01	-4.515e-01
	std	%	NaN	NaN	1.074e-03	1.074e-03	1.812e-03	9.418e-03
		num	0.000e+00	0.000e+00	1.928e-05	1.928e-05	4.848e-05	3.116e-05
	diff/std	%	NaN	NaN	2.412e+06	2.412e+06	2.796e+05	1.449e+06

11.8 Using the reconstructed matrix elements

The results tables are accessible directly, and there are also methods to reformat the best fit results for use in further calculations.

```
# self.paramsSummary contains the results above as Pandas Dataframe, usual Pandas
#methods can be applied.
data.paramsSummary['data'].describe()
```

Param	PU_SG_PU_1_1_n1_1	PU_SG_PU_1_n1_1_1	PU_SG_PU_3_1_n1_1	PU_SG_PU_3_n1_1_1	SU_SG_SU_1_1_1
count	124.000	124.000	124.000	124.000	124.000
mean	0.351	0.351	0.759	0.759	0.759
std	0.972	0.972	1.008	1.008	1.008
min	-0.861	-0.861	-2.656	-2.656	-2.656
25%	-0.215	-0.215	0.223	0.223	0.223
50%	0.226	0.226	0.896	0.896	0.896
75%	0.792	0.792	1.149	1.149	1.149
max	1.815	1.815	1.794	1.794	1.794

```
# To set matrix elements from aggregate fit results, use `setAggMatE` for Pandas
data.setAggMatE(simpleForm = True)
data.data['agg']['matEpd']
```

Set reformatted aggregate data to self.data[agg][matEpd].

Cont	l	m	Type mu	m	n	p	pc	comp	compC	labels
PU	1	-1	1	1.815	0.451	-0.861	0.000	1.183-1.377j	0.451+0.000j	1,-1
		1	-1	1.815	0.451	-0.861	0.000	1.183-1.377j	0.451+0.000j	1,1
	3	-1	1	0.896	0.223	0.123	1.794	0.889+0.110j	-0.049+0.217j	3,-1
		1	-1	0.896	0.223	0.123	1.794	0.889+0.110j	-0.049+0.217j	3,1
SU	1	0	0	2.473	0.615	2.025	2.675	-1.085+2.222j	-0.550+0.277j	1,0
	3	0	0	1.361	0.339	-0.680	0.331	1.059-0.855j	0.320+0.110j	3,0

```
# To set matrix elements from aggregate fit results, use `aggToXR` for Xarray
# data.aggToXR(refKey = 'orb5', returnType = 'ds', conformDims=True) # use full ref
#dataset
data.aggToXR(refKey = 'subset', returnType = 'ds', conformDims=True) # Subselected
#mate
```

```
Added dim Targ
Added dim Total
Added dim Targ
Added dim Total
Set XR dataset for self.data['agg']['matE']
```

11.8.1 Density matrices

New (experimental) code for density matrix plots and comparison. See Sect. 6.4 for discussion. Code adapted from the PEMtk documentation [20] MF reconstruction page, original analysis for Ref. [3], illustrating the N_2 case. If the reconstruction is good, the differences (fidelity) should be on the order of the experimental noise level/reconstruction uncertainty, around 10% in the case studies herein; in general the values and patterns of the matrices can also indicate aspects of the retrieval that worked well, or areas where values are poorly defined/recovered from the given dataset.

11.8.2 Plot MF PADs

Routines as per https://pemtk.readthedocs.io/en/latest/topical_review_case_study/MFPAD_replotting_from_file_190722-dist.html - currently not working. Seems to be some difference in dim stacking/assignment now...? Might be Python/Xarray version change, or PEMtk/ePSproc implementation.

```
# Compute MFPADs for a range of cases

# Set Euler angs to include diagonal pol case
pRot = [0, 0, np.pi/2, 0]
tRot = [0, np.pi/2, np.pi/2, np.pi/4]
cRot = [0, 0, 0, 0]
labels = ['z','x','y', 'd']
eulerangs = np.array([labels, pRot, tRot, cRot]).T      # List form to use later, rows_
→per set of angles

# Should also use MFBLM function below instead of numeric version?
# Numeric version is handy for direct surface and difference case.
R = ep.setPolGeoms(eulerangs = eulerangs)

# Comparison and diff
pKey = [i for i in dataIn.data_vars if i != 'comp']    # List of arrays
dataTest.mfpadNumeric(keys=pKey, R = R)      # Compute MFPADs for each set of matrix_
→elements using numerical routine

dataTest.data['diff'] = {'TX': dataTest.data['subset']['TX'].sum('Sym')-dataTest.data[_
←'compC']['TX'].sum('Sym')}   # Add sum over sym to force matching dims
pKey.extend(['diff'])

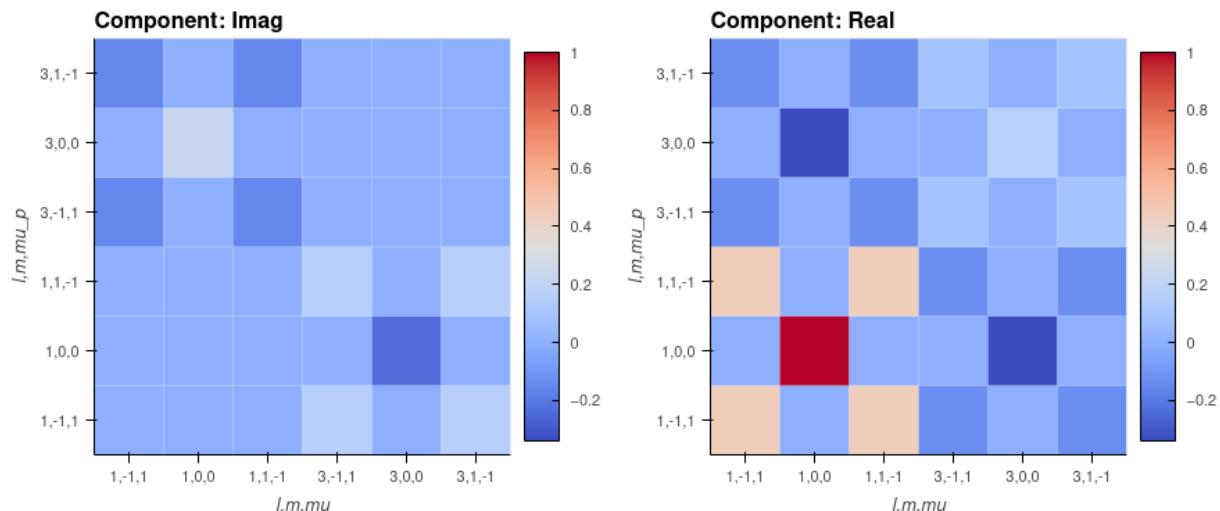
# Plot - all cases
# Now run in separate cells below for more stable output
# Erange=[1,2,1] # Set for a range of Ekes
# Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as_
→used for fitting)
# print(f"\n*** Plotting for keys = {pKey}, one per row ***\n") # Note plot labels_
→could do with some work!
# dataTest.padPlot(keys=pKey, Erange=Erange, backend='pl', returnFlag=True,_
→plotFlag=True) # Generate plotly polar surf plots for each dataset

# Change default plotting config, and then plot in separate cells below
ep.plot.hvPlotters.setPlotters(width=1000, height=500)
```

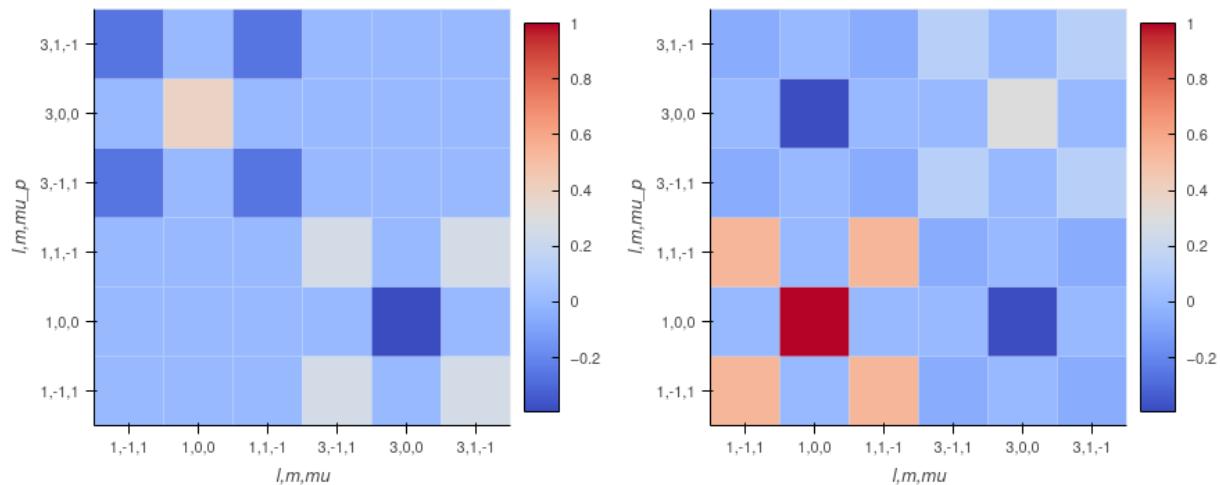
```
# Plot results from reconstructed matE
pKey = 'compC'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with_
→some work!
# dataTest.padPlot(keys=pKey, Erange=Erange, backend='pl', returnFlag=True,_
→plotFlag=True) # Generate plotly polar surf plots for each dataset
```

(continues on next page)

(a) Reference density matrix (unsigned phases)



(b) Reconstructed



(c) Difference

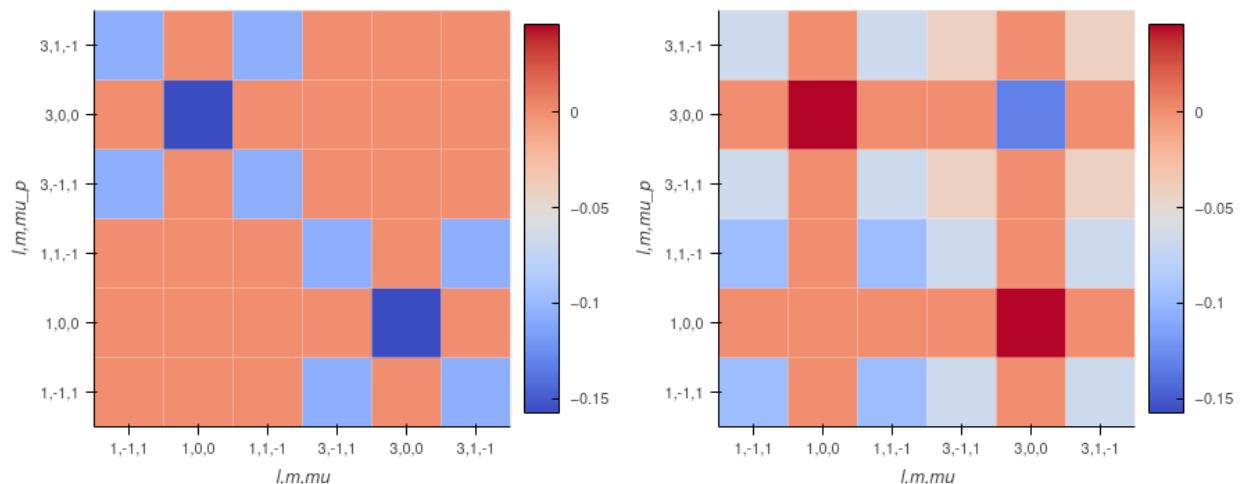


Fig. 11.3: Density matrix comparison - rows show (a) reference case (with signs of phases removed), (b) reconstructed case, (c) differences. Columns are (left) imaginary component, (right) real component. If the reconstruction is good, the differences (fidelity) should be on the order of the experimental noise level/reconstruction uncertainty, around 10% in the 11.8. Using the reconstructed matrix elements

(continued from previous page)

```
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("N2-compC", figObj)
```

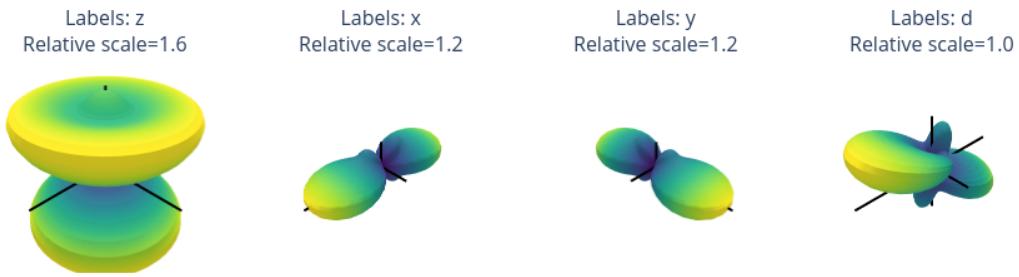


Fig. 11.4: *MF-PADs* computed from retrieved matrix elements for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis.

```
# Plot results from reference matE
pKey = 'subset'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("N2-ref", figObj)
```

```
# Plot normalised differences
pKey = 'diff'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset
```

(continues on next page)

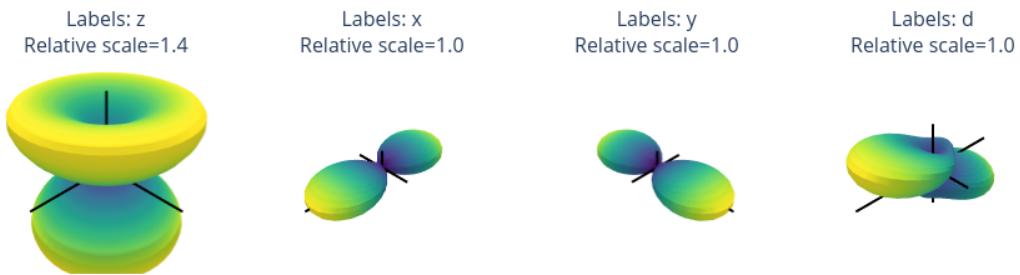


Fig. 11.5: *MF-PADs* computed from reference *ab initio* matrix elements for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis.

(continued from previous page)

```
# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("N2-diff", figObj)
```

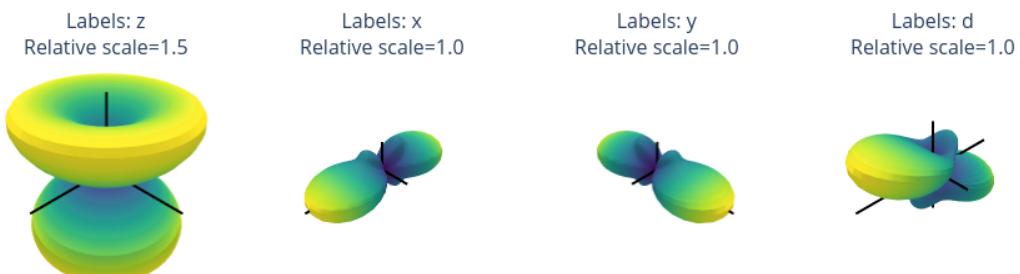


Fig. 11.6: *MF-PADs* differences between retrieved and reference cases for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis. Note diffs are normalised to emphasize the shape, but not magnitudes, of the differences - see the density matrix comparisons for a more rigorous fidelity analysis.

CASE STUDY: GENERALISED BOOTSTRAPPING FOR A LINEAR HETERONUCLEAR SCATTERING SYSTEM, *OCS* ($C_{\infty V}$)

In this chapter, the full code and analysis details of the case study for *OCS* are given, including obtaining required data, running fits and analysis routines. For more details on the routines, see the [PEMtk documentation \[20\]](#); for the analysis see particularly the [fit fidelity](#) and [analysis](#) page, and [molecular frame analysis data processing](#) page (full analysis for Ref. [3], illustrating the N_2 case).

12.1 General setup

In the following code cells (see source notebooks for full details) the general setup routines (as per the outline in Chpt. 10) are executed via a configuration script with presets for the case studies herein.

Additionally, the routines will either run fits, or load existing data if available. Since fitting can be computationally demanding, it is, in general, recommended to approach large fitting problems carefully.

General note on fitting

Computational outputs in this chapter are significantly truncated in the PDF, and some simplified plots are used; see source notebooks (via [Quantum Metrology Vol. 3 \(Github repo\)](#)) or [Quantum Metrology Vol. 3 \(HTML version\)](#) for full details.

```
# Configure settings for case study

# Set case study by name
fitSystem='OCS'
fitStem=f"fit_withNoise_orb13"

# Add noise?
addNoise = 'y'
mu, sigma = 0, 0.05 # Up to approx 10% noise (+/- 0.05)

# Batching - number of fits to run between data dumps
batchSize = 10

# Total fits to run
nMax = 10
```

```
# Pull data from web (OCS case)
from epsproc.util.io import getFilesFromGithub

# Set dataName (will be used as download subdir)
dataName = 'OCSfitting'
# OCS matrix elements
fDictMatE, fAllMatE = getFilesFromGithub(subpath='data/photoionization/OCS_multiorb',_
    ↳dataName=dataName, ref='dev')
# OCS alignment data
fDictADM, fAllADM = getFilesFromGithub(subpath='data/alignment/OCS ADMs_28K_VM_070722
    ↳', dataName=dataName, ref='dev')
```

```
Querying URL: https://api.github.com/repos/phockett/epsproc/contents/data/
    ↳photoionization/OCS_multiorb?ref=dev
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/OCS_survey.orb10_E0.1_2.0_30.1eV.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/OCS_survey.orb10_E1.1_2.0_31.1eV.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/OCS_survey.orb11_E0.1_2.0_30.1eV.inp.out already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/OCS_survey.orb11_E1.1_2.0_31.1eV.inp.out already exists
Querying URL: https://api.github.com/repos/phockett/epsproc/contents/data/
    ↳alignment/OCS ADMs_28K_VM_070722?ref=dev
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/A20_300fs_4p2TW_28K.dat already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/A40_300fs_4p2TW_28K.dat already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/A60_300fs_4p2TW_28K.dat already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/c2t_300fs_4p2TW_28K.dat already exists
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↳OCSfitting/time_300fs_4p2TW_28K.dat already exists
```

```
# Fitting setup including data generation and parameter creation

# Set datapath,
dataPath = Path(Path.cwd(),dataName)

# Run general config script with dataPath set above
%run "../scripts/setup_fit_case-studies_270723.py" -d {dataPath} -a {dataPath} -c
    ↳{fitSystem} -n {addNoise} --sigma {sigma} -a3D 'y'
```

12.2 Load existing fit data or run fits

Note that running fits may be quite time-consuming and computationally intensive, depending on the size of the size of the problem. The default case here will run a small batch for testing if there is no existing data found on the `dataPath`, otherwise the data is loaded for analysis.

```
# Look for existing Pickle files on path
dataFiles = list(dataPath.expanduser().glob('*.pickle'))
```

(continues on next page)

(continued from previous page)

```

if not dataFiles:
    print("No data found, executing minimal fitting run...")

    # Run fit batch - single
    # data.multiFit(nRange = [n,n+batchSize-1], num_workers=batchSize)

    # Run fit batches with checkpoint files
    for n in np.arange(0,nMax,batchSize):
        print(f'*** Running batch [{n},{n+batchSize-1}], {dt.now().strftime("%d%m%y_"
        ↪%H-%M-%S")}')

        # Run fit batch
        data.multiFit(nRange = [n,n+batchSize-1], num_workers=batchSize)

        # Dump data so far
        data.writeFitData(outStem=f'{fitSystem}_{n+batchSize-1}_{fitStem}')

        print(f'Finished batch [{n},{n+batchSize-1}], {dt.now().strftime("%d%m%y_%H-"
        ↪%M-%S")}')
        print(f'Written to file {fitSystem}_{n+batchSize-1}_{fitStem}')


else:
    dataFileIn = dataFiles[-1]    # Add index to select file, although loadFitData_
    ↪will concat multiple files
                                # Note that concat currently only works for fixed_
    ↪batch sizes however.
    print(f"Set dataFiles: {dataFileIn}")
    data.loadFitData(fList=dataFileIn, dataPath=dataPath)    #.expanduser())

    data.BLMfitPlot(keys=['subset','sim'])

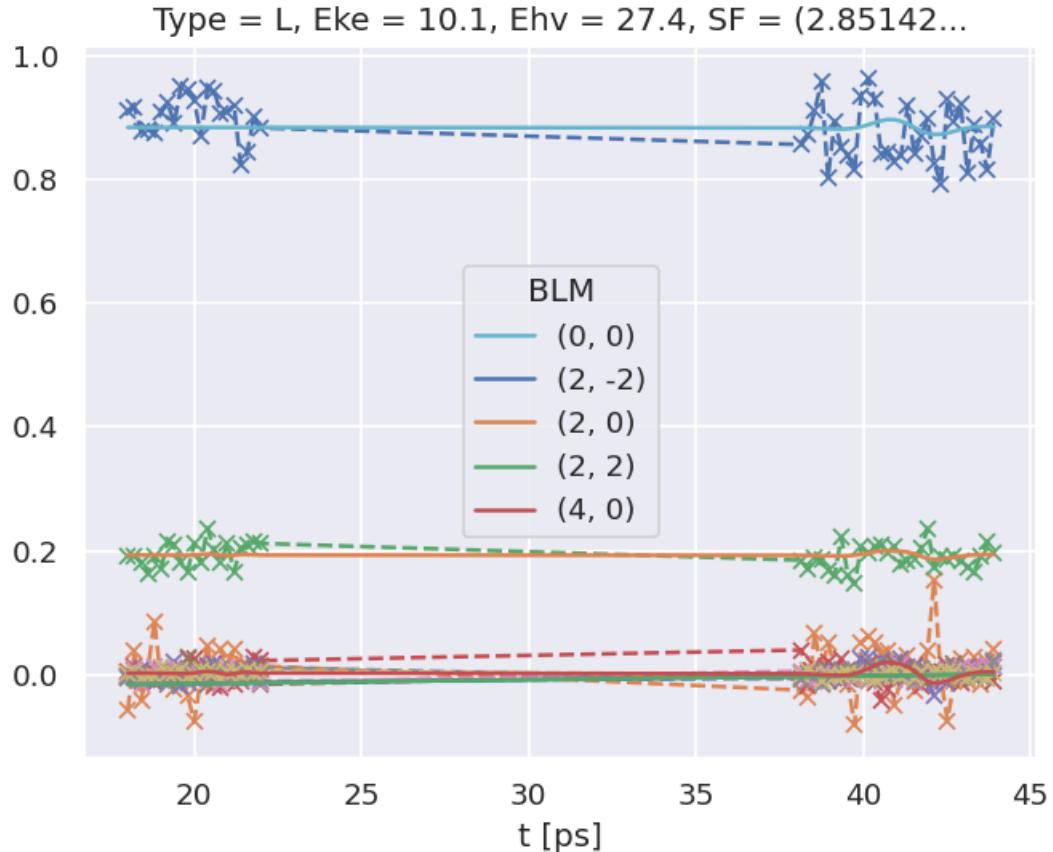
    # # Check ADMs
    # data.data['subset']['ADM'].unstack().where(data.data['subset']['ADM'].unstack()
    ↪K>0) \
        # .real.hvplot.line(x='t').overlay(['K','Q','S'])

```

```

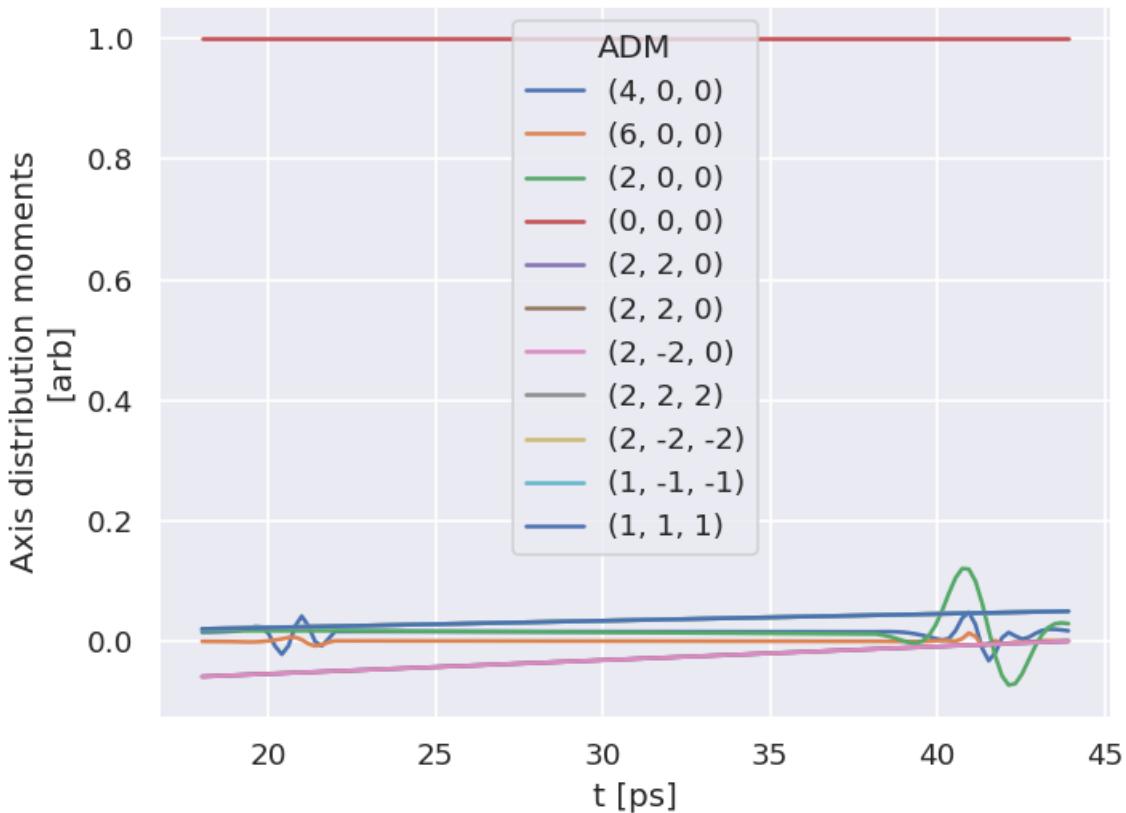
Set dataFiles: /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪OCSfitting/OCS_999_fit_3D-test_withNoise_orb13_200723_05-47-50.pickle
Read data from /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪OCSfitting/OCS_999_fit_3D-test_withNoise_orb13_200723_05-47-50.pickle with_
    ↪pickle.
Dataset: subset, AFBLM
Dataset: sim, AFBLM

```



```
# Check ADMs
# Basic plotter
data.ADMplot(keys = 'subset')
```

Dataset: subset, ADM



```
# Fits appear as integer indexed items in the main data structure.
data.data.keys()
```

12.3 Post-processing and data overview

Post-processing involves aggregation of all the fit run results into a single data structure. This can then be analysed statistically and examined for best-fit results. In the statistical sense, this is essentially a search for candidate *radial matrix elements*, based on the assumption that some of the minima found in the χ^2 hyperspace will be the true results. Even if a clear global minima does not exist, searching for candidate *radial matrix elements* sets based on clustering of results and multiple local minima is still expected to lead to viable candidates provided that the information content of the dataset is sufficient. However, as discussed elsewhere (see Sect. 7.2), in some cases this may not be the case, and other limitations may apply (e.g. certain parameters may be undefined), or additional data required for unique determination of the *radial matrix elements*.

For more details on the analysis routines, see the PEMtk documentation [20], particularly the [fit fidelity](#) and [analysis](#) page, and [molecular frame analysis](#) data processing page (full analysis for Ref. [3], illustrating the N_2 case).

```
# General stats & post-processing to data tables
data.analyseFits()
```

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
  ↵with Pandas MultiIndex, but will try anyway.
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
  ↵with Pandas MultiIndex, but will try anyway.
```

(continues on next page)

(continued from previous page)

```
{
  'Fits': 975,
  'Minima': {'chisqr': 0.2510827613766783, 'redchi': 0.00011080439601795158},
  'Stats': { 'chisqr': min 0.251
mean 0.253
median 0.251
max 2.304
std 0.066
var 0.004
Name: chisqr, dtype: float64,
      'redchi': min 1.108e-04
mean 1.117e-04
median 1.108e-04
max 1.017e-03
std 2.902e-05
var 8.420e-10
Name: redchi, dtype: float64},
  'Success': 971}
```

```
# The BLMsetPlot routine will output aggregate fit results.
# Here the spread can be taken as a general indication of the uncertainty of
# the fitting, and indicate whether the fit is well-characterised/the information
# content of the data is sufficient.
data.BLMsetPlot(xDim = 't', thres=1e-6) # With xDim and thres set, for more control
# over outputs

# Glue plot for later
glue("OCS-fitResultsBLM", data.data['plots']['BLMsetPlot'])
```

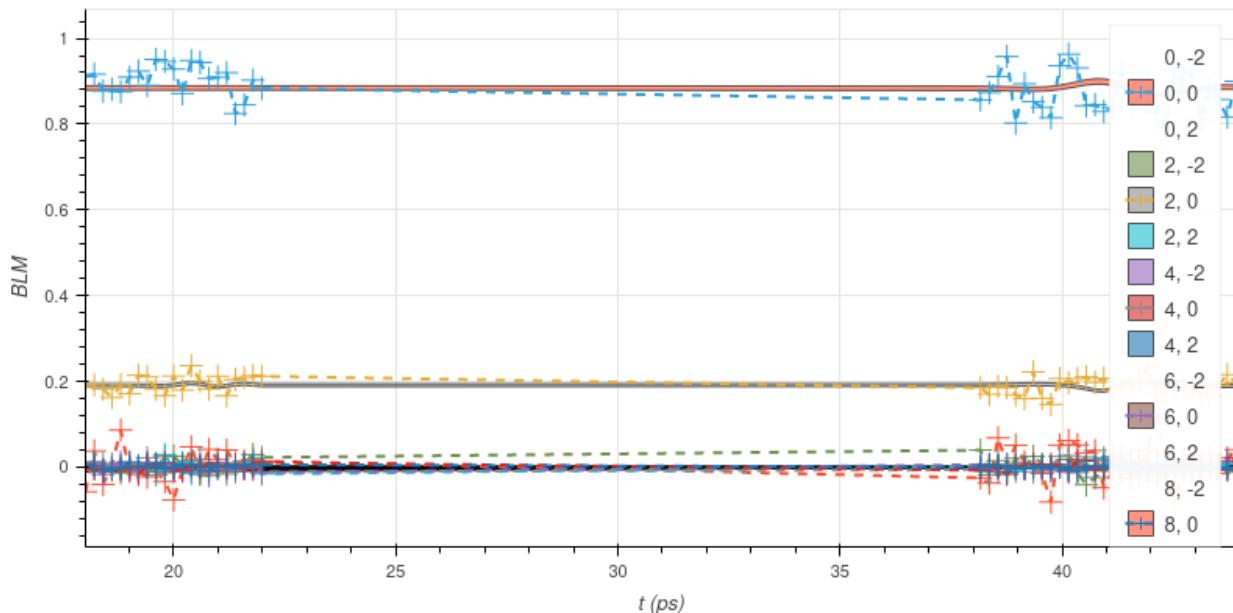


Fig. 12.1: Fit overview plot - $\beta_{L,M}(t)$. Here dashed lines with '+' markers indicates the input data, and bands indicate the mean fit results, where the width is the standard deviation in the fit model results. (See the PEMtk documentation [20] for details, particularly the analysis routines page.)

```
# Write aggregate datasets to HDF5 format
# This is more robust than Pickled data, but PEMtk currently only support output for_
# aggregate (post-processed) fit data.

data.processedToHDF5(dataPath = dataPath, outStem = dataFileIn.name, timeStamp=False)
```

```
Dumped self.data[fits][dfLong] to /home/jovyan/jake-home/buildTmp/_latest_build/
↳pdf/doc-source/part2/OCSfitting/OCS_999_fit_3D-test_withNoise_orb13_200723_05-47-
↳50.pickle_dfLong.pdHDF with Pandas .to_hdf() routine.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳OCSfitting/OCS_999_fit_3D-test_withNoise_orb13_200723_05-47-50.pickle_dfLong.
↳pdHDF with pdHDF.
Dumped self.data[fits][AFxr] to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/
↳doc-source/part2/OCSfitting/OCS_999_fit_3D-test_withNoise_orb13_200723_05-47-50.
↳pickle_AFxer.pdHDF with Pandas .to_hdf() routine.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↳OCSfitting/OCS_999_fit_3D-test_withNoise_orb13_200723_05-47-50.pickle_AFxer.pdHDF
↳with pdHDF.
```

```
# Histogram fit results (reduced chi^2 vs. fit index)
# This may be quite slow for large datasets, setting limited ranges may help

# Use default auto binning
# data.fitHist()

# Example with range set
data.fitHist(thres=1.11e-4, bins=100)

# Glue plot for later
glue("OCS-fitHist", data.data['plots']['fitHistPlot'])
```

Mask selected 974 results (from 975).

```
:AdjointLayout
:Scatter [redchi] (Fit)
:Histogram [Fit] (Fit_count)
:Histogram [redchi] (redchi_count)
```

Here bands in the χ^2 dimension can indicate groupings (local minima) are consistently found. Assuming each grouping is a viable fit candidate parameter set, these can then be explored in further detail.

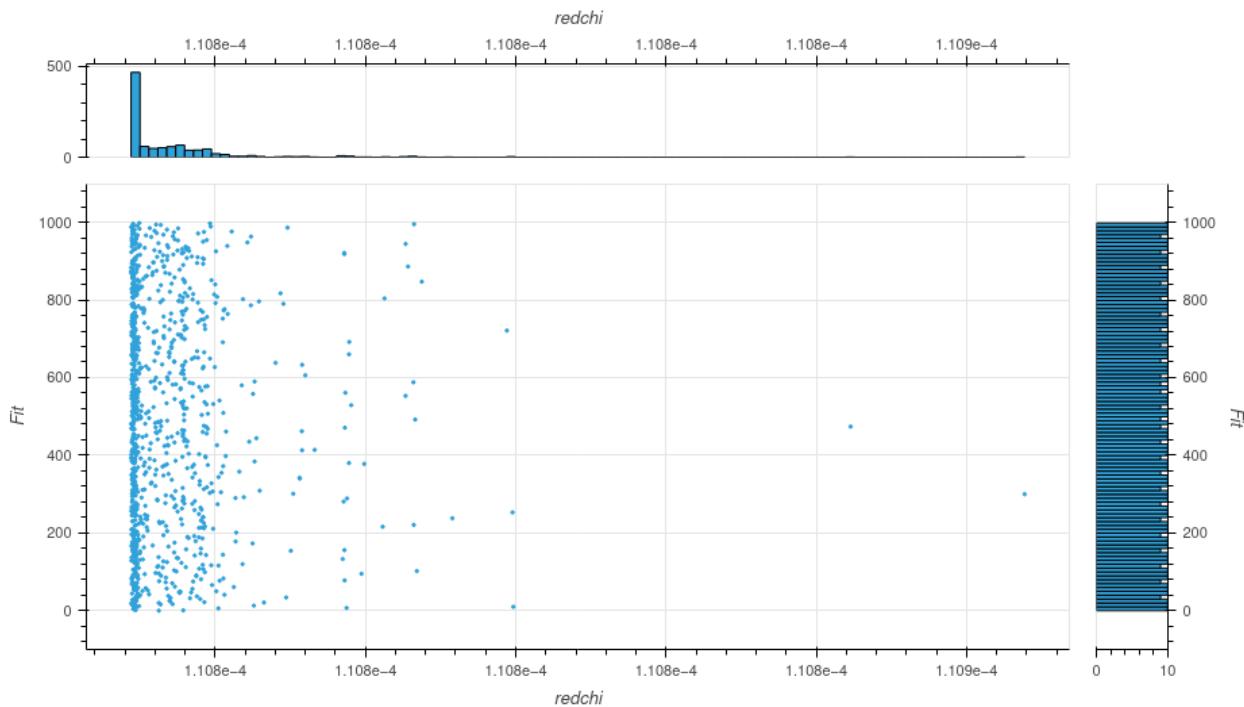


Fig. 12.2: Fit overview plot - χ^2 vs. fit index. Here bands indicate groupings (local minima) are consistently found.

12.4 Data exploration

The general aim in this procedure is to ascertain whether there was a good spread of parameters explored, and a single (or few sets) of best-fit results. There are a few procedures and helper methods for this...

12.4.1 View results

Single results sets can be viewed in the main data structure, indexed by #.

```
# Check keys
fitNumber = 2
data.data[fitNumber].keys()
```

```
dict_keys(['AFBLM', 'residual', 'results'])
```

Here `results` is an `ImFit` object, which includes final fit results and information, and `AFBLM` contains the model (fit) output.

An example is shown below. Of particular note here is which parameters have `vary=True` - these are included in the fitting - and if there is a column `expression`, which indicates any parameters defined to have specific relationships (see Chpt. 9). Any correlations found during fitting are also shown, which can also indicate parameters which are related (even if this is not predefined or known a priori).

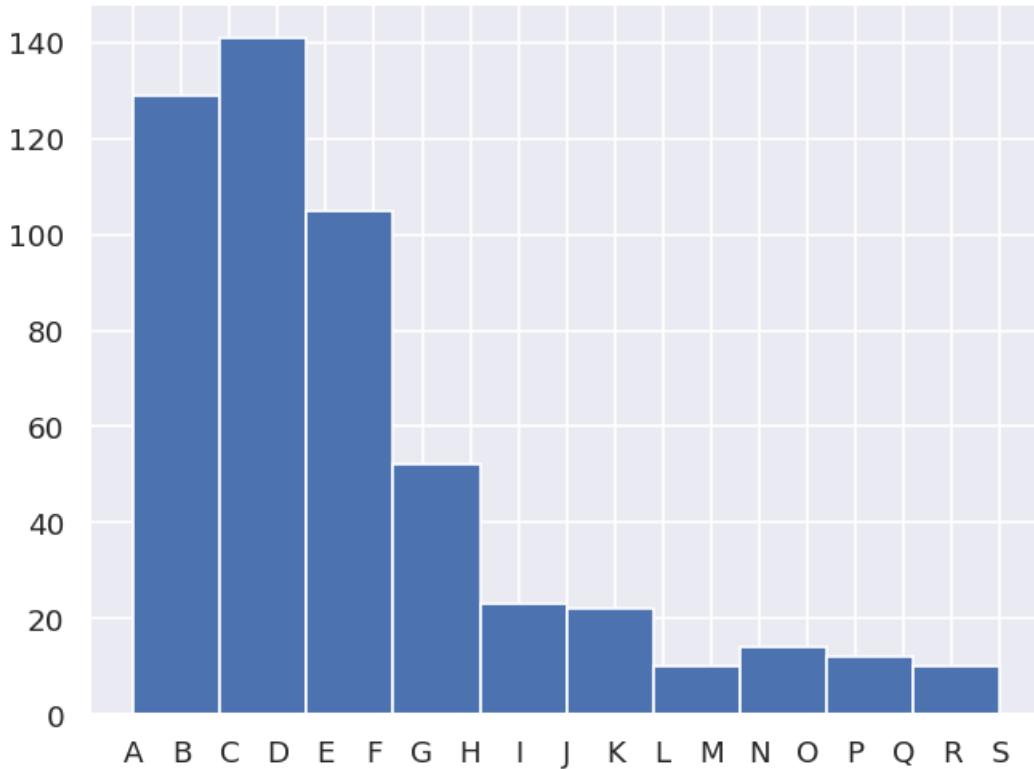
```
# Show some results
data.data[fitNumber]['results']
```

12.5 Classify candidate sets

16 Chapter 12. Case study: Generalised bootstrapping for a linear heteronuclear scattering system,
To probe the minima found, the `classifyFits` method can be used. This bins results into “candidate” groups, which can then be examined in detail.

redchiGroup	success				chisqr				redchi			
	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq
A	47	1	True	47	47.0	47.0	0.251	1.0	47.0	47.0	1.108e-04	1.0
B	82	1	True	82	82.0	82.0	0.251	1.0	82.0	82.0	1.108e-04	1.0
C	65	1	True	65	65.0	65.0	0.251	1.0	65.0	65.0	1.108e-04	1.0
D	76	1	True	76	76.0	76.0	0.251	1.0	76.0	76.0	1.108e-04	1.0
E	51	1	True	51	51.0	51.0	0.251	1.0	51.0	51.0	1.108e-04	1.0
F	54	1	True	54	54.0	54.0	0.251	1.0	54.0	54.0	1.108e-04	1.0
G	28	1	True	28	28.0	28.0	0.251	1.0	28.0	28.0	1.108e-04	1.0
H	24	1	True	24	24.0	24.0	0.251	1.0	24.0	24.0	1.108e-04	1.0
I	23	1	True	23	23.0	23.0	0.251	1.0	23.0	23.0	1.108e-04	1.0
J	10	1	True	10	10.0	10.0	0.251	1.0	10.0	10.0	1.108e-04	1.0
K	12	1	True	12	12.0	12.0	0.251	1.0	12.0	12.0	1.108e-04	1.0
L	7	1	True	7	7.0	7.0	0.251	1.0	7.0	7.0	1.108e-04	1.0
M	3	1	True	3	3.0	3.0	0.251	1.0	3.0	3.0	1.108e-04	1.0
N	7	1	True	7	7.0	7.0	0.251	1.0	7.0	7.0	1.108e-04	1.0
O	7	1	True	7	7.0	7.0	0.251	1.0	7.0	7.0	1.108e-04	1.0
P	7	1	True	7	7.0	7.0	0.251	1.0	7.0	7.0	1.108e-04	1.0
Q	5	1	True	5	5.0	5.0	0.251	1.0	5.0	5.0	1.108e-04	1.0
R	7	1	True	7	7.0	7.0	0.251	1.0	7.0	7.0	1.108e-04	1.0
S	3	1	True	3	3.0	3.0	0.251	1.0	3.0	3.0	1.108e-04	1.0

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.
```



12.6 Explore candidate result sets

Drill-down on a candidate set of results, and examine values and spreads. For more details see [PEMtk documentation \[20\]](#), especially the [analysis routines page](#). (See also [Sect. 5.3](#) for details on the plotting libraries implemented here.)

12.6.1 Raw results

Plot spreads in magnitude and phase parameters. Statistical plots are available for Seaborn and Holoviews backends, with some slightly different options.

```
# From the candidates, select a group for analysis
selGroup = 'A'

# paramPlot can be used to check the spread on each parameter.
# Plots use Seaborn or Holoviews/Bokeh
# Colour-mapping is controlled by the 'hue' parameter, additionally pass hRound for
# sig. fig control.
# The remap setting allows for short-hand labels as set in data.lmmu

paramType = 'm' # Set for (m)agnitude or (p)hase parameters
hRound = 14 # Set for cmapping, default may be too small (leads to all grey cmap on
# points)

data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
```

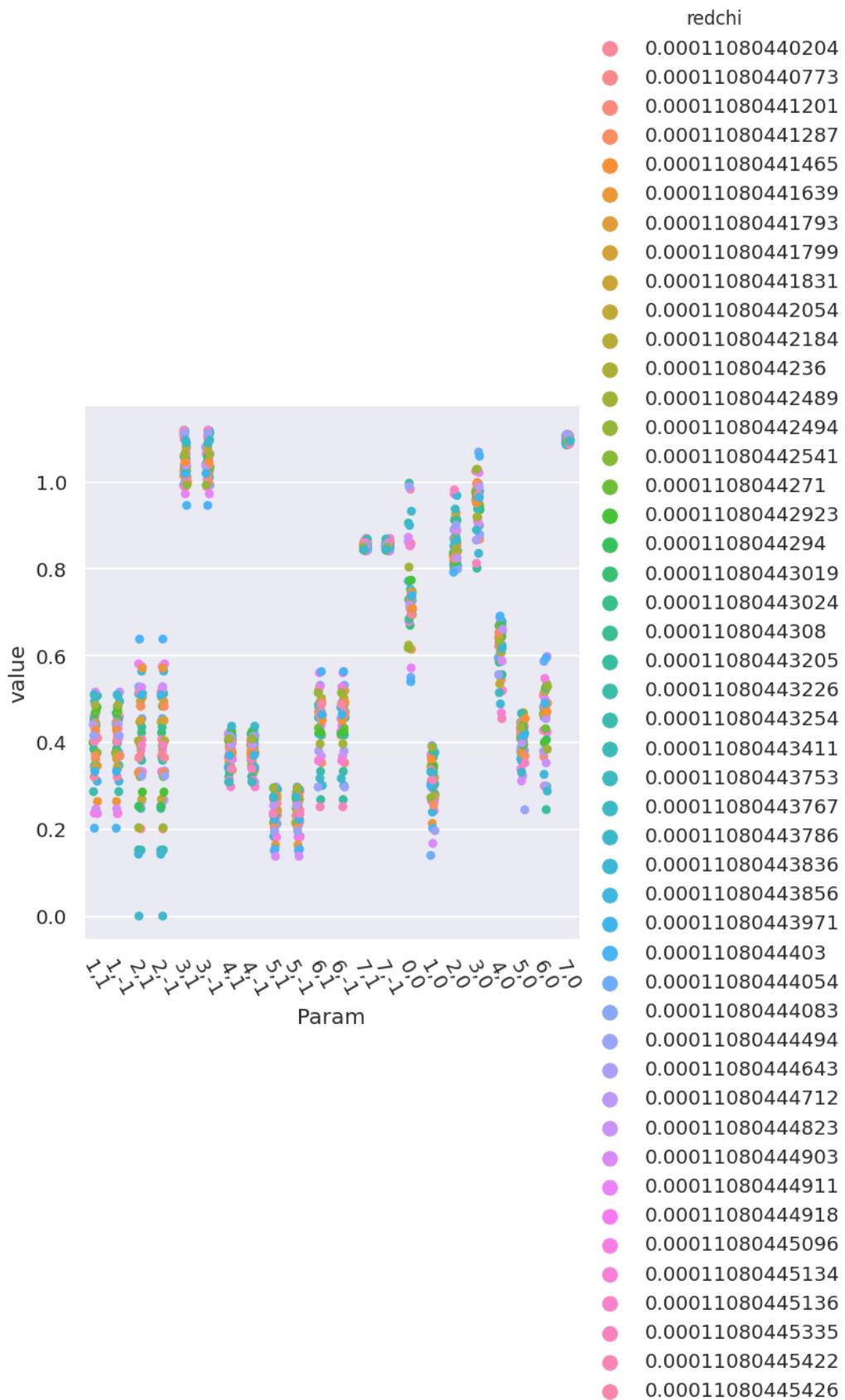
(continues on next page)

(continued from previous page)

```
backend=paramPlotBackend, hvType='violin',
returnFlag = True, hRound=hRound, remap = 'lmMap');
```

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
with Pandas MultiIndex, but will try anyway.
```

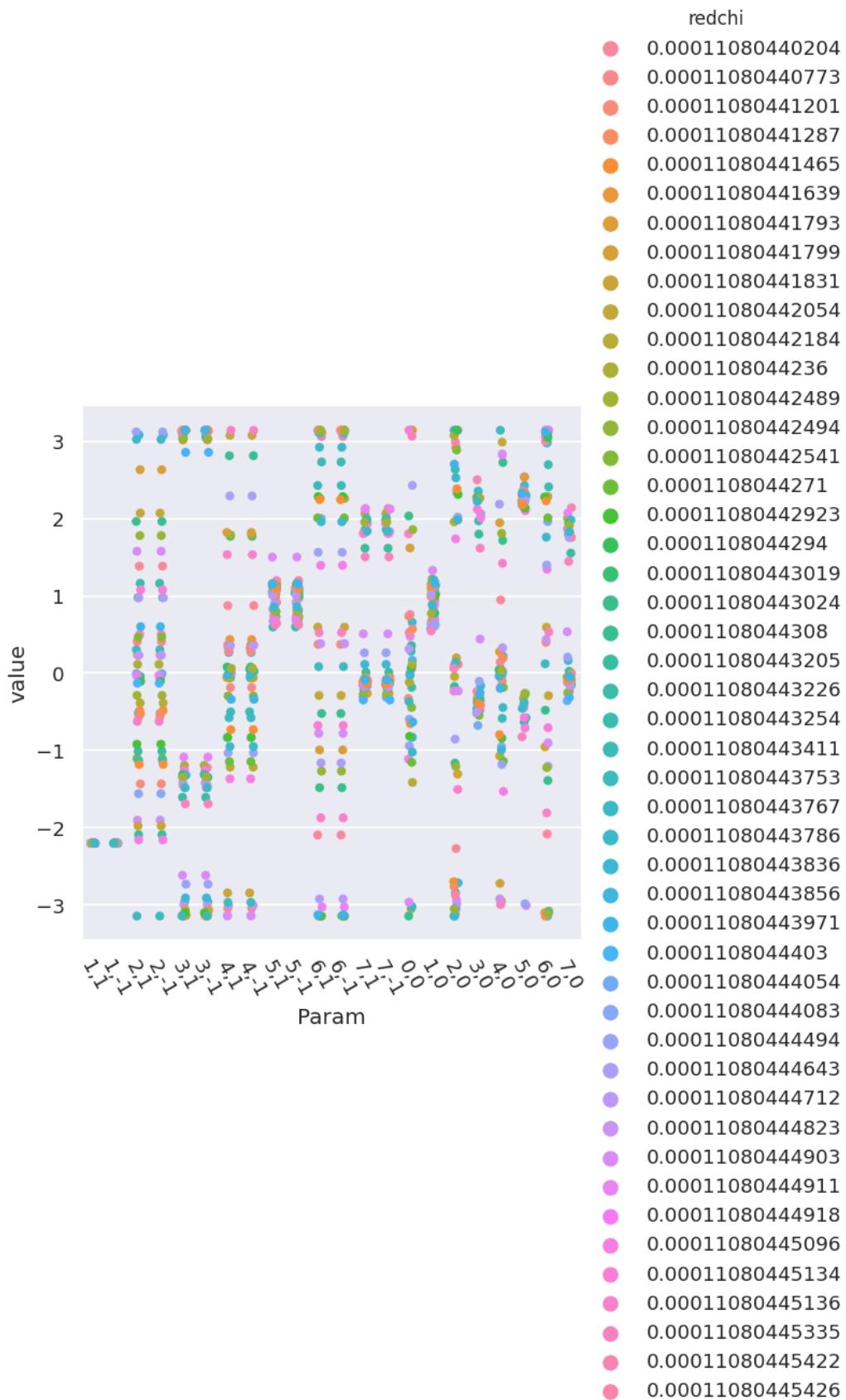
```
<seaborn.axisgrid.FacetGrid at 0x7f4704640d30>
```



```
paramType = 'p' # Set for (m)agnitude or (p)hase parameters
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
               backend=paramPlotBackend, hvType='violin',
               returnFlag = True, hRound=hRound, remap = 'lmMap');
```

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues
with Pandas MultiIndex, but will try anyway.
```

```
<seaborn.axisgrid.FacetGrid at 0x7f47042acf40>
```



12.6.2 Phases, phase shifts & corrections

Depending on how the fit was configured, phases may be defined in different ways. To set the phases relative to a specific parameter, and wrap to a specified range, use the `phaseCorrection()` method. This defaults to using the first parameter as a reference phase, and wraps to $-\pi : \pi$. The phase-corrected values are output to a new Type, ‘pc’, and a set of normalised magnitudes to ‘n’. Additional settings can be passed for more control, as shown below.

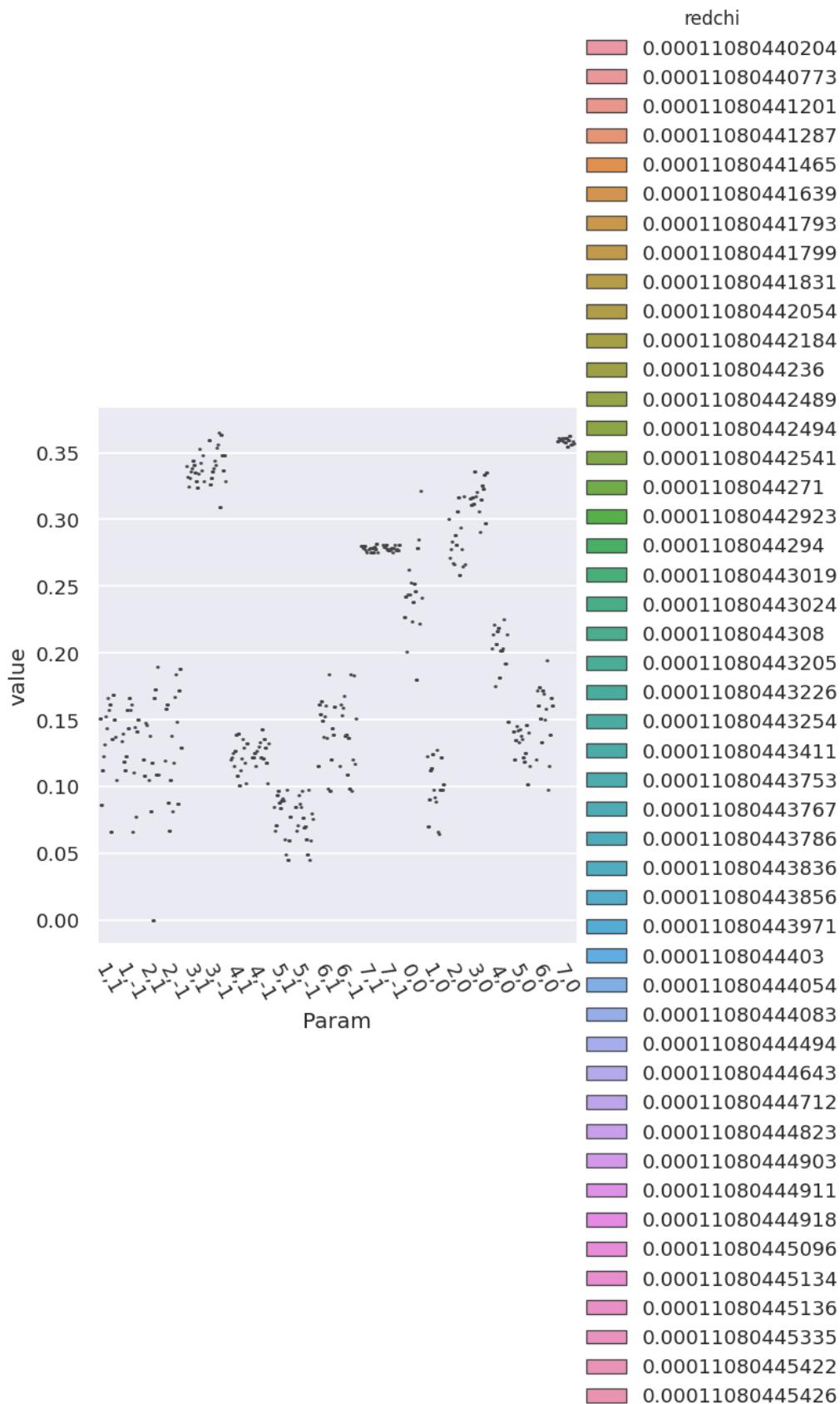
```
# Run phase correction routine
# Set absFlag=True for unsigned phases (mapped to 0:pi)
# Set useRef=False to set ref phase as 0, otherwise the reference value is set.
phaseCorrParams={'absFlag':True, 'useRef':False}
data.phaseCorrection(**phaseCorrParams)
```

Examine new data types...

```
paramType = 'n'
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
               backend=paramPlotBackend, hvType='violin', kind='box',
               returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.

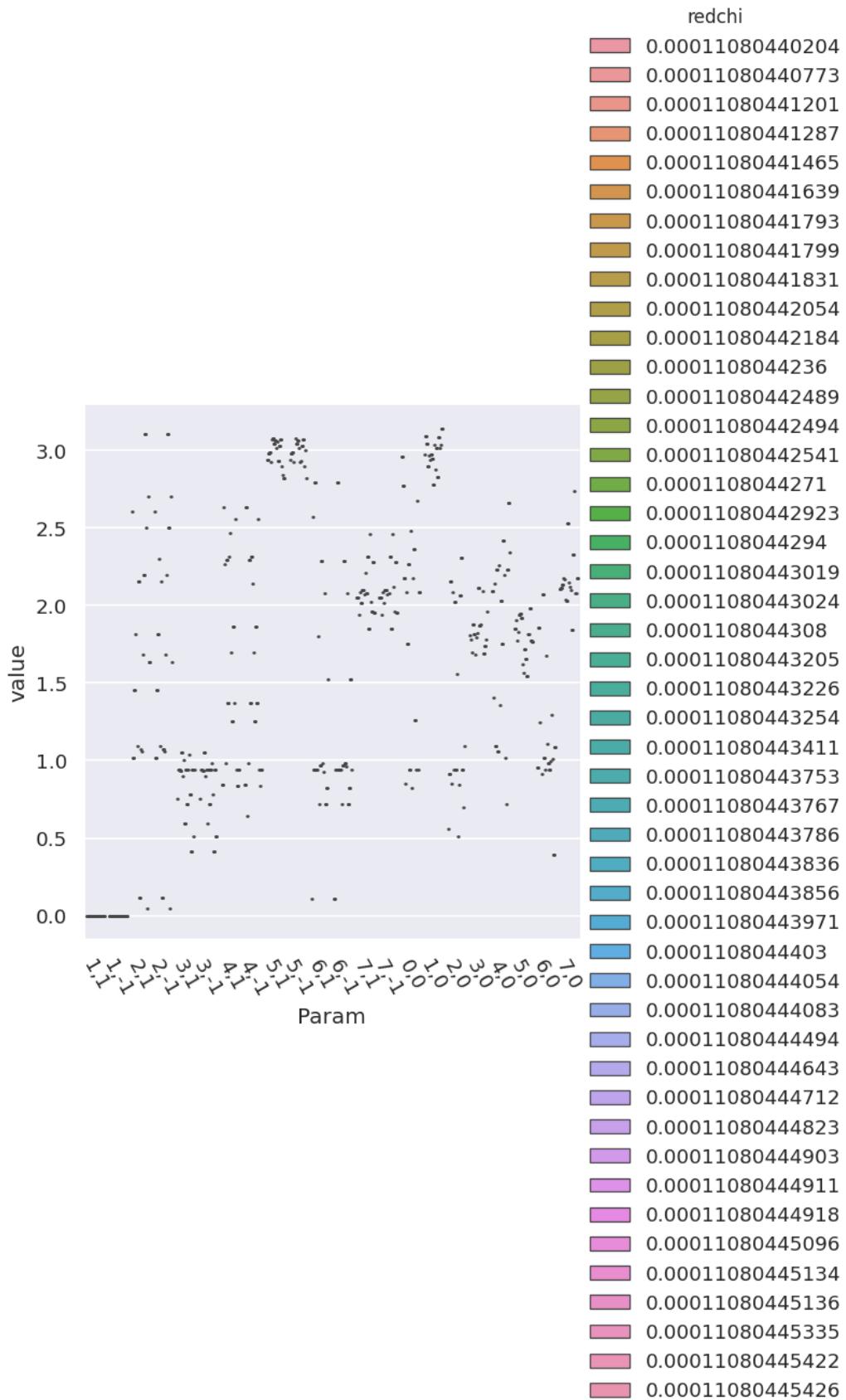
<seaborn.axisgrid.FacetGrid at 0x7f4704721a80>



```
paramType = 'pc'  
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',  
                backend=paramPlotBackend, hvType='violin', kind='box',  
                returnFlag = True, hRound=hRound, remap = 'lmMap');
```

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues  
with Pandas MultiIndex, but will try anyway.
```

```
<seaborn.axisgrid.FacetGrid at 0x7f47045dae60>
```



12.7 Parameter estimation & fidelity

For case studies, the fit results can be directly compared to the known input parameters. This should give a feel for how well the data defines the matrix elements (parameters) in this case. In general, probing the correlations and spread of results, and comparing to other (unfitted) results is required to estimate fidelity, see *Quantum Metrology* Vols. 1 & 2 [4, 9] for further discussion.

12.7.1 Best values and statistics

To get a final parameter set and associated statistics, based on a subset of the fit results, the `paramsReport()` method is available. If reference data is available, as for the case studies herein, the `paramsCompare()` method can also be used to compare with the reference case.

```
# Parameter summary
data.paramsReport(inds = {'redchiGroup':selGroup})
```



```
# Parameter comparison
# Note this uses phaseCorrParams as set previously for consistency
data.paramsCompare(phaseCorrParams=phaseCorrParams)
```



```
# Display above results With column name remapping to (l,m) labels only

# With Pandas functionality
data.paramsSummaryComp.rename(columns=data.lmmu['lmMap'])

# With utility method
# summaryRenamed = pemtk.fit._util.renameParams(data.paramsSummaryComp, data.lmmu[
#   ↪'lmMap'])
# summaryRenamed
```

Type	Source	Param dType	1,1	1,-1	2,1	2,-1	3,1	3,-1	4,1	4,-1
m	mean	num	0.395	0.395	0.384	0.384	1.041	1.041	0.378	0.378
	ref	num	0.299	0.299	0.735	0.735	0.966	0.966	0.875	0.875
	diff	%	24.383	24.383	91.513	91.513	7.275	7.275	131.508	131.508
		num	0.096	0.096	-0.351	-0.351	0.076	0.076	-0.497	-0.497
	std	%	19.952	19.952	35.782	35.782	3.929	3.929	9.266	9.266
		num	0.079	0.079	0.137	0.137	0.041	0.041	0.035	0.035
	diff/std	%	122.212	122.212	255.754	255.754	185.143	185.143	1419.210	1419.210
	n	mean	0.129	0.129	0.125	0.125	0.340	0.340	0.123	0.123
	ref	num	0.097	0.097	0.240	0.240	0.315	0.315	0.286	0.286
	diff	%	24.386	24.386	91.506	91.506	7.278	7.278	131.499	131.499
p		num	0.031	0.031	-0.115	-0.115	0.025	0.025	-0.162	-0.162
	std	%	19.952	19.952	35.782	35.782	3.929	3.929	9.266	9.266
		num	0.026	0.026	0.045	0.045	0.013	0.013	0.011	0.011
	diff/std	%	122.226	122.226	255.734	255.734	185.231	185.231	1419.117	1419.117
	mean	num	-2.200	-2.200	0.091	0.091	-0.054	-0.054	-0.139	-0.139
	ref	num	-2.200	-2.200	-1.410	-1.410	-2.100	-2.100	-0.800	-0.800
	diff	%	0.000	0.000	1646.083	1646.083	3757.652	3757.652	475.859	475.859
		num	0.000	0.000	1.501	1.501	2.046	2.046	0.661	0.661
	std	%	0.000	0.000	1572.473	1572.473	4976.382	4976.382	1110.494	1110.494
		num	0.000	0.000	1.434	1.434	2.709	2.709	1.542	1.542
pc	diff/std	%	NaN	NaN	104.681	104.681	75.510	75.510	42.851	42.851
	mean	num	0.000	0.000	1.796	1.796	0.888	0.888	1.816	1.816
	ref	num	0.000	0.000	0.790	0.790	0.100	0.100	1.401	1.401
	diff	%	NaN	NaN	56.010	56.010	88.748	88.748	22.861	22.861
		num	0.000	0.000	1.006	1.006	0.788	0.788	0.415	0.415
	std	%	NaN	NaN	46.193	46.193	17.279	17.279	36.044	36.044
		num	0.000	0.000	0.830	0.830	0.153	0.153	0.654	0.654
diff/std	%	NaN	NaN	121.253	121.253	513.612	513.612	63.425	63.425	676.0

12.8 Using the reconstructed matrix elements

The results tables are accessible directly, and there are also methods to reformat the best fit results for use in further calculations.

```
# self.paramsSummary contains the results above as Pandas Dataframe, usual Pandas
methods can be applied.
data.paramsSummary['data'].describe()
```

Param	P_S_P_1_1_n1_1	P_S_P_1_n1_1_1	P_S_P_2_1_n1_1	P_S_P_2_n1_1_1	P_S_P_3_1_n1_1	P_S_P_3_n1_1_1
count	188.000	188.000	188.000	188.000	188.000	188.000
mean	-0.419	-0.419	0.599	0.599	0.554	0.554
std	1.042	1.042	1.083	1.083	1.416	1.416
min	-2.200	-2.200	-3.142	-3.142	-3.142	-3.142
25%	-0.550	-0.550	0.113	0.113	0.337	0.337
50%	0.033	0.033	0.322	0.322	0.928	0.928
75%	0.177	0.177	1.063	1.063	1.039	1.039
max	0.516	0.516	3.121	3.121	3.142	3.142

```
# To set matrix elements from aggregate fit results, use `setAggMatE` for Pandas
data.setAggMatE(simpleForm = True)
data.data['agg']['matEpd']
```

Set reformatted aggregate data to self.data[agg][matEpd].

Cont	1	Type		m	n	p	pc	comp	compC	labels
		m	mu							
P	1	-1	1	0.395	0.129	-2.200	0.000	-0.232-0.319j	0.129+0.000j	1,-1
		1	-1	0.395	0.129	-2.200	0.000	-0.232-0.319j	0.129+0.000j	1,1
	2	-1	1	0.384	0.125	0.091	1.796	0.382+0.035j	-0.028+0.122j	2,-1
		1	-1	0.384	0.125	0.091	1.796	0.382+0.035j	-0.028+0.122j	2,1
	3	-1	1	1.041	0.340	-0.054	0.888	1.040-0.057j	0.215+0.264j	3,-1
		1	-1	1.041	0.340	-0.054	0.888	1.040-0.057j	0.215+0.264j	3,1
	4	-1	1	0.378	0.123	-0.139	1.816	0.374-0.052j	-0.030+0.120j	4,-1
		1	-1	0.378	0.123	-0.139	1.816	0.374-0.052j	-0.030+0.120j	4,1
	5	-1	1	0.239	0.078	0.979	2.979	0.133+0.198j	-0.077+0.013j	5,-1
		1	-1	0.239	0.078	0.979	2.979	0.133+0.198j	-0.077+0.013j	5,1
	6	-1	1	0.439	0.143	0.515	1.374	0.382+0.216j	0.028+0.141j	6,-1
		1	-1	0.439	0.143	0.515	1.374	0.382+0.216j	0.028+0.141j	6,1
	7	-1	1	0.853	0.278	0.490	2.125	0.753+0.401j	-0.147+0.237j	7,-1
		1	-1	0.853	0.278	0.490	2.125	0.753+0.401j	-0.147+0.237j	7,1
S	0	0	0	0.752	0.246	0.161	1.826	0.742+0.121j	-0.062+0.238j	0,0
	1	0	0	0.305	0.100	0.966	2.965	0.173+0.251j	-0.098+0.017j	1,0
	2	0	0	0.860	0.281	0.177	1.348	0.847+0.151j	0.062+0.274j	2,0
	3	0	0	0.954	0.312	0.380	1.875	0.886+0.354j	-0.093+0.297j	3,0
	4	0	0	0.615	0.201	0.095	1.810	0.612+0.058j	-0.048+0.195j	4,0
	5	0	0	0.402	0.131	1.237	1.739	0.132+0.380j	-0.022+0.129j	5,0
	6	0	0	0.455	0.149	0.763	1.402	0.329+0.314j	0.025+0.146j	6,0
	7	0	0	1.098	0.359	0.494	2.156	0.967+0.521j	-0.198+0.299j	7,0

```
# To set matrix elements from aggregate fit results, use `aggToXR` for Xarray
# data.aggToXR(refKey = 'orb5', returnType = 'ds', conformDims=True)    # use full ref_
# dataset
data.aggToXR(refKey = 'subset', returnType = 'ds', conformDims=True)      # Subselected_
# mate
```

```

Added dim Total
Added dim Targ
Added dim Total
Added dim Targ
Set XR dataset for self.data['agg']['matE']

```

12.8.1 Density matrices

New (experimental) code for density matrix plots and comparison. See Sect. 6.4 for discussion. Code adapted from the PEMtk documentation [20] MF reconstruction page, original analysis for Ref. [3], illustrating the N_2 case. If the reconstruction is good, the differences (fidelity) should be on the order of the experimental noise level/reconstruction uncertainty, around 10% in the case studies herein; in general the values and patterns of the matrices can also indicate aspects of the retrieval that worked well, or areas where values are poorly defined/recovered from the given dataset.

12.8.2 Plot MF PADs

Routines as per https://pemtk.readthedocs.io/en/latest/topical_review_case_study/MFPAD_replotting_from_file_190722-dist.html - currently not working. Seems to be some difference in dim stacking/assignment now...? Might be Python/Xarray version change, or PEMtk/ePSproc implementation.

```

# Compute MFPADs for a range of cases

# Set Euler angs to include diagonal pol case
pRot = [0, 0, np.pi/2, 0]
tRot = [0, np.pi/2, np.pi/2, np.pi/4]
cRot = [0, 0, 0, 0]
labels = ['z','x','y', 'd']
eulerangs = np.array([labels, pRot, tRot, cRot]).T      # List form to use later, rows
# per set of angles

# Should also use MFBLM function below instead of numeric version?
# Numeric version is handy for direct surface and difference case.
R = ep.setPolGeoms(eulerangs = eulerangs)

# Comparison and diff
pKey = [i for i in dataIn.data_vars if i!='comp']    # List of arrays
dataTest.mfpadNumeric(keys=pKey, R = R)    # Compute MFPADs for each set of matrix
# elements using numerical routine

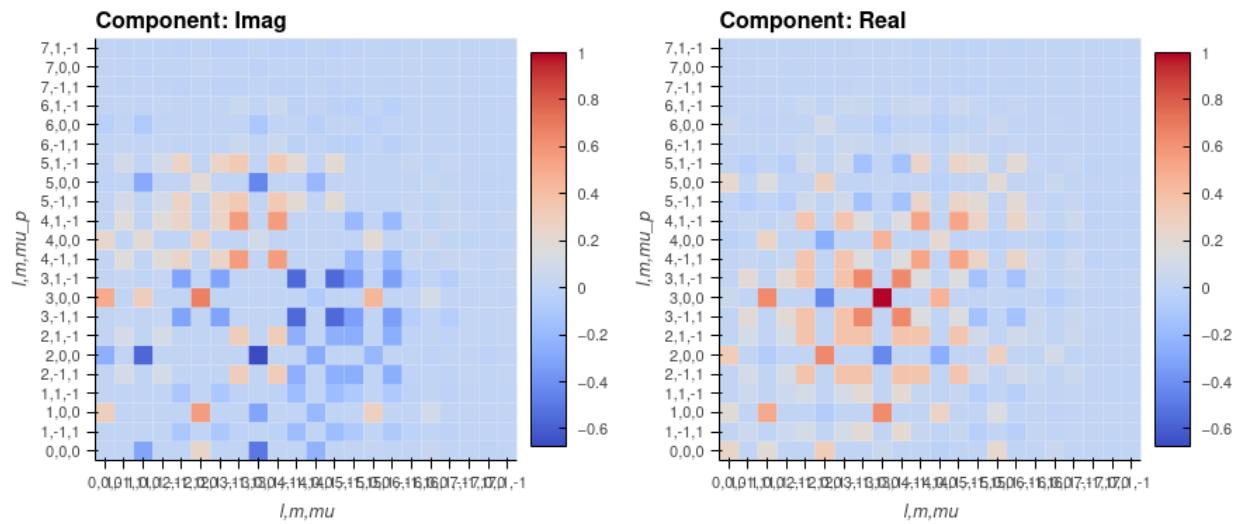
dataTest.data['diff'] = {'TX': dataTest.data['subset']['TX'].sum('Sym')-dataTest.data[
# 'compC']['TX'].sum('Sym')}    # Add sum over sym to force matching dims
pKey.extend(['diff'])

# Plot - all cases
# Now run in separate cells below for more stable output
# Erange=[1,2,1]    # Set for a range of Ekes
# Eplot = {'Eke':data.selOpt['matE']['inds']['Eke']}    # Plot for selected Eke (as
# used for fitting)
# print(f"\n*** Plotting for keys = {pKey}, one per row ***\n")    # Note plot labels
# could do with some work!
# dataTest.padPlot(keys=pKey, Erange=Erange, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset

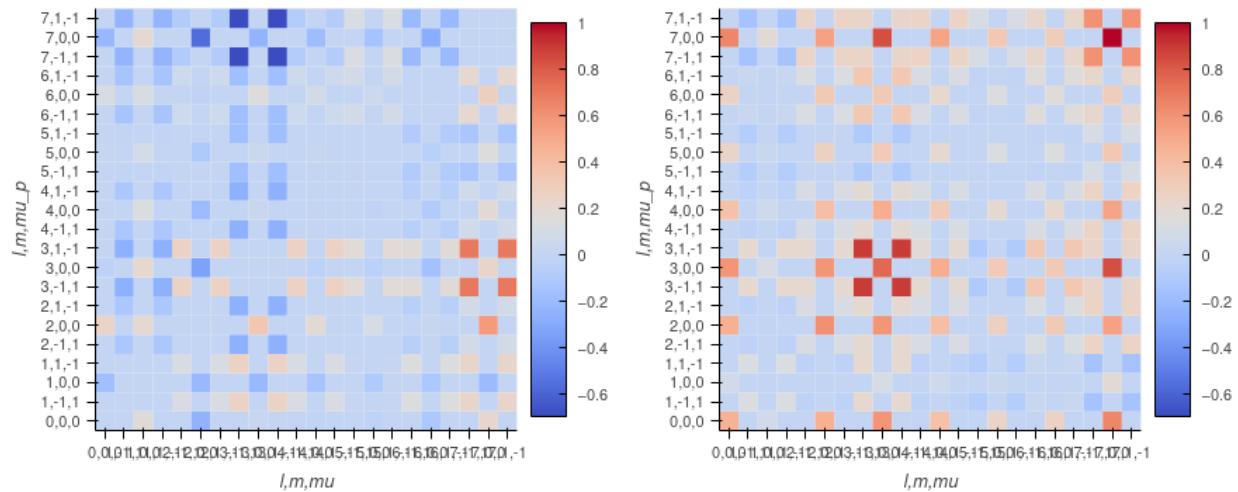
```

(continues on next page)

(a) Reference density matrix (unsigned phases)



(b) Reconstructed



(c) Difference

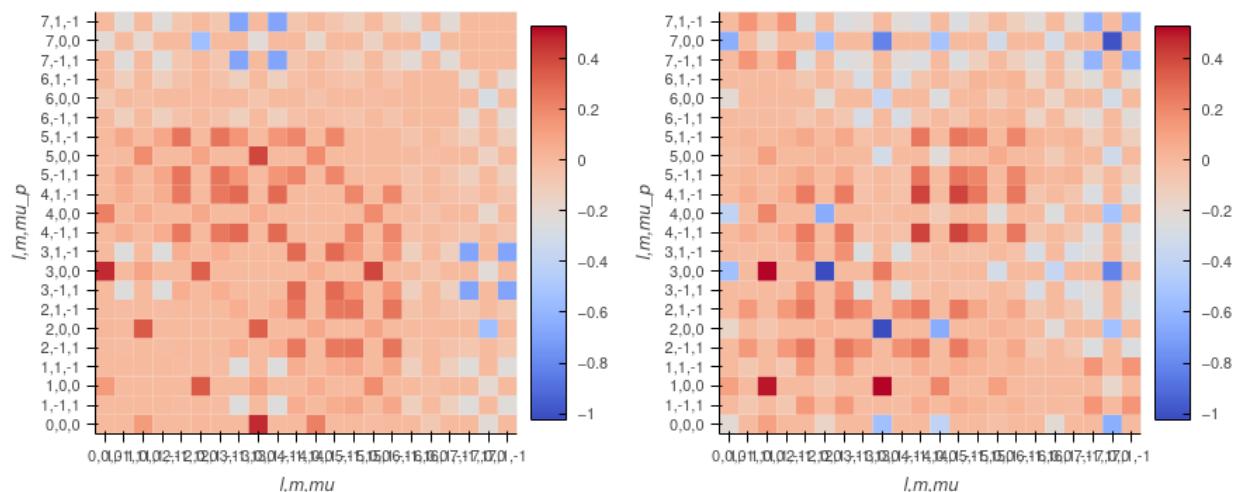


Fig. 12.3: Density matrix comparison - rows show (a) reference case (with signs of phases removed), (b) reconstructed case, (c) differences. Columns are (left) imaginary component, (right) real component. If the reconstruction is good, the differences (fidelity) should be on the order of the experimental noise level/reconstruction uncertainty, around 10% in the **12.8. Using the reconstructed matrix elements** case studies herein. **173**

(continued from previous page)

```
# Change default plotting config, and then plot in separate cells below
ep.plot.hvPlotters.setPlotters(width=1000, height=500)

# Plot results from reconstructed matE
pKey = 'compC'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("OCS-compC", figObj)
```

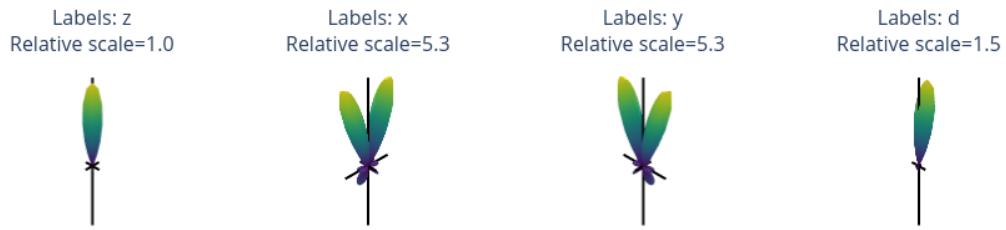


Fig. 12.4: **MF-PADs** computed from retrieved matrix elements for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis.

```
# Plot results from reference matE
pKey = 'subset'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("OCS-ref", figObj)
```



Fig. 12.5: *MF-PADs* computed from reference *ab initio* matrix elements for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis.

```
# Plot normalised differences
pKey = 'diff'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erange, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("OCS-diff", figObj)
```



Fig. 12.6: *MF-PADs* differences between retrieved and reference cases for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis. Note diffs are normalised to emphasize the shape, but not magnitudes, of the differences - see the density matrix comparisons for a more rigorous fidelity analysis.

CHAPTER
THIRTEEN

CASE STUDY: GENERALISED BOOTSTRAPPING FOR A GENERAL ASYMMETRIC TOP SCATTERING SYSTEM, C_2H_4 (D_{2H})

In this chapter, the full code and analysis details of the case study for C_2H_4 are given, including obtaining required data, running fits and analysis routines. For more details on the routines, see the [PEMtk documentation \[20\]](#); for the analysis see particularly the [fit fidelity](#) and [analysis](#) page, and [molecular frame analysis data processing](#) page (full analysis for Ref. [3], illustrating the N_2 case).

13.1 General setup

In the following code cells (see source notebooks for full details) the general setup routines (as per the outline in Chpt. 10) are executed via a configuration script with presets for the case studies herein.

Additionally, the routines will either run fits, or load existing data if available. Since fitting can be computationally demanding, it is, in general, recommended to approach large fitting problems carefully.

General note on fitting

Computational outputs in this chapter are significantly truncated in the PDF, and some simplified plots are used; see source notebooks (via [Quantum Metrology Vol. 3 \(Github repo\)](#)) or [Quantum Metrology Vol. 3 \(HTML version\)](#) for full details.

```
# Configure settings for case study

# Set case study by name
fitSystem='C2H4'
fitStem=f"fit_3D-test_withNoise_orb8"

# Add noise?
addNoise = 'y'
mu, sigma = 0, 0.05 # Up to approx 10% noise (+/- 0.05)

# Batching - number of fits to run between data dumps
batchSize = 10

# Total fits to run
nMax = 10
```

```
# Pull data from web (C2H4 case)
from epsproc.util.io import getFilesFromGithub

# Set dataName (will be used as download subdir)
dataName = 'C2H4fitting'
# C2H4 matrix elements
GHbranch='3d-AFPAD-dev'
fDictMatE, fAllMatE = getFilesFromGithub(subpath='data/photoionization/C2H4',_
    ↪dataName=dataName, ref=GHbranch)
# C2H4 alignment data
fDictADM, fAllADM = getFilesFromGithub(subpath='data/alignment/C2H4 ADMs_8TW_120fs_VM
    ↪', dataName=dataName, ref=GHbranch)
```

```
Querying URL: https://api.github.com/repos/phockett/epsproc/contents/data/
    ↪photoionization/C2H4?ref=3d-AFPAD-dev
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪C2H4fitting/C2H4_1.0-100.0eV_orb8_B3u.inp.out already exists
Querying URL: https://api.github.com/repos/phockett/epsproc/contents/data/
    ↪alignment/C2H4 ADMs_8TW_120fs_VM?ref=3d-AFPAD-dev
Local file /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪C2H4fitting/ADMs_8TW_120fs_5K.mat already exists
```

```
# Fitting setup including data generation and parameter creation

# Set datapath,
dataPath = Path(Path.cwd(),dataName)
ADMfile = 'ADMs_8TW_120fs_5K.mat'

# Run general config script with dataPath set above
%run "../scripts/setup_fit_case-studies_270723.py" -d {dataPath} -a {ADMfile} -c
    ↪{fitSystem} -n {addNoise} --sigma {sigma}
```

13.2 Load existing fit data or run fits

Note that running fits may be quite time-consuming and computationally intensive, depending on the size of the size of the problem. The default case here will run a small batch for testing if there is no existing data found on the `dataPath`, otherwise the data is loaded for analysis.

```
# Look for existing Pickle files on path
dataFiles = list(dataPath.expanduser().glob('*.pickle'))

if not dataFiles:
    print("No data found, executing minimal fitting run...")

    # Run fit batch - single
    # data.multiFit(nRange = [n,n+batchSize-1], num_workers=batchSize)

    # Run fit batches with checkpoint files
    for n in np.arange(0,nMax,batchSize):
        print(f'*** Running batch [{n},{n+batchSize-1}], {dt.now().strftime("%a %Y-%H-%M-%S")}' )
```

(continues on next page)

(continued from previous page)

```

# Run fit batch
data.multiFit(nRange = [n,n+batchSize-1], num_workers=batchSize)

# Dump data so far
data.writeFitData(outStem=f"fitSystem_{n+batchSize-1}_{fitStem}")

print(f'Finished batch [{n},{n+batchSize-1}], {dt.now().strftime("%d%m%y_%H-%M-%S")}')
print(f'Written to file {fitSystem}_{n+batchSize-1}_{fitStem}')


else:
    dataFileIn = dataFiles[-1]    # Add index to select file, although loadFitData_
    ↪will concat multiple files                                # Note that concat currently only works for fixed_
    ↪batch sizes however.
    print(f"Set dataFiles: {dataFileIn}")
    data.loadFitData(fList=dataFileIn, dataPath=dataPath)      #.expanduser()

    data.BLMfitPlot(keys=['subset', 'sim'])

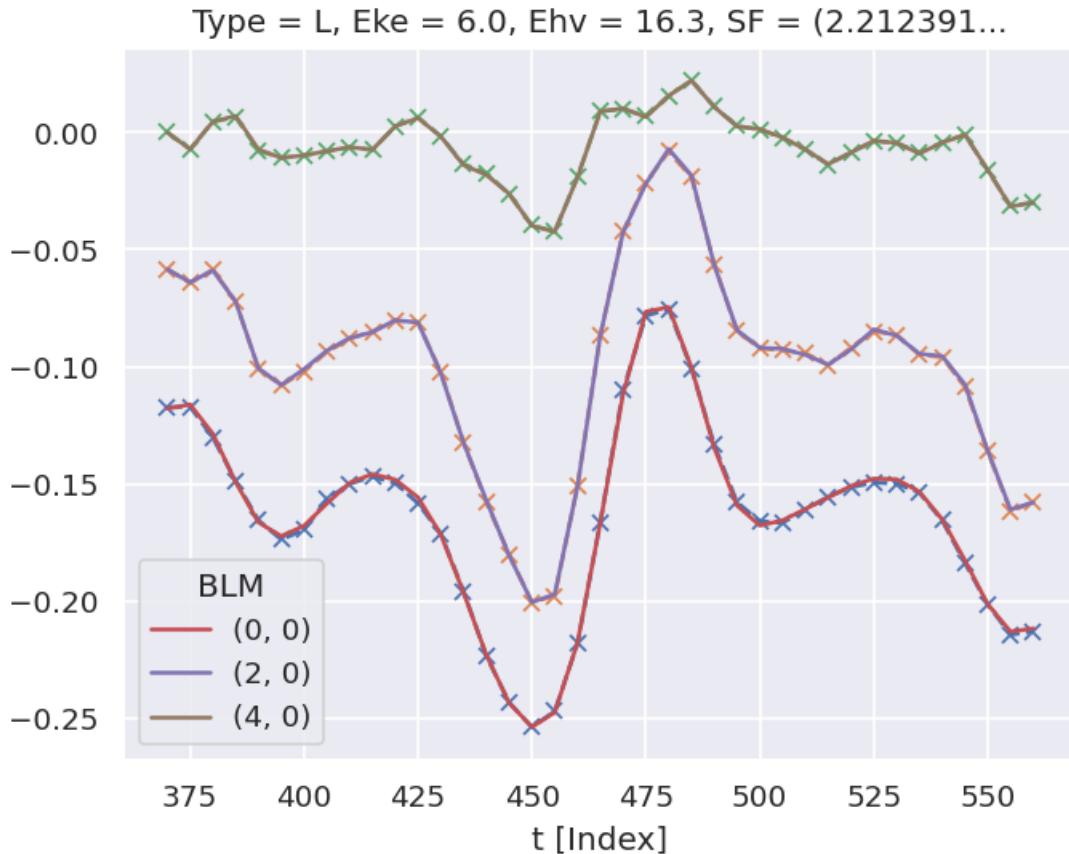
    # # Check ADMs
    # data.data['subset']['ADM'].unstack().where(data.data['subset']['ADM'].unstack()
    ↪K>0) \
    #     .real.hvplot.line(x='t').overlay(['K', 'Q', 'S'])

```

```

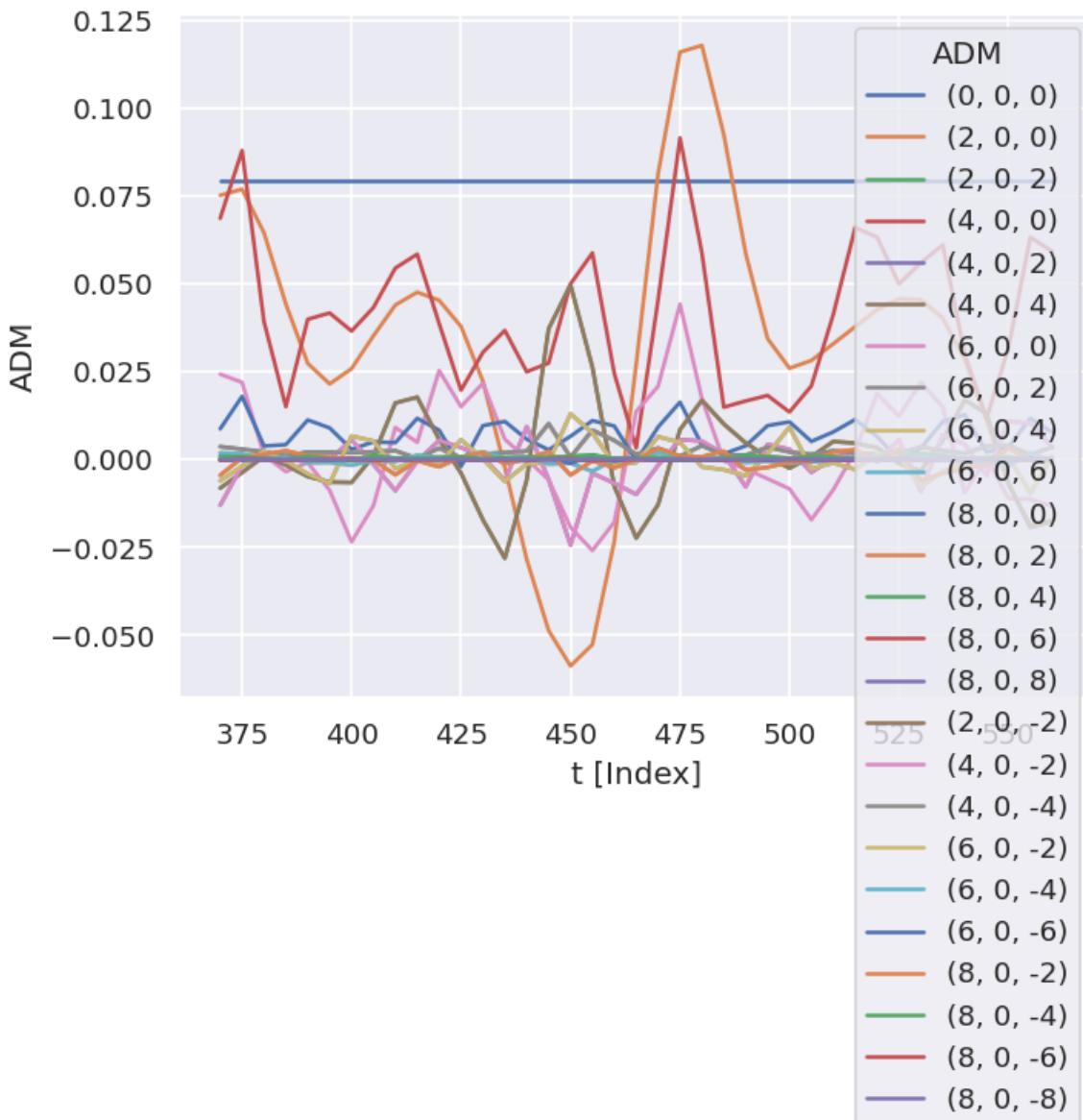
Set dataFiles: /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪C2H4fitting/C2H4_399_fit_3D-test_withNoise_orb8_270723_13-45-22.pickle
Read data from /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
    ↪C2H4fitting/C2H4_399_fit_3D-test_withNoise_orb8_270723_13-45-22.pickle with_
    ↪pickle.
Dataset: subset, AFBLM
Dataset: sim, AFBLM

```



```
# Check ADMs
# Basic plotter
data.ADMplot(keys = 'subset')
```

Dataset: subset, ADM



```
# Fits appear as integer indexed items in the main data structure.
data.data.keys()
```

13.3 Post-processing and data overview

Post-processing involves aggregation of all the fit run results into a single data structure. This can then be analysed statistically and examined for best-fit results. In the statistical sense, this is essentially a search for candidate *radial matrix elements*, based on the assumption that some of the minima found in the χ^2 hyperspace will be the true results. Even if a clear global minima does not exist, searching for candidate *radial matrix elements* sets based on clustering of results and multiple local minima is still expected to lead to viable candidates provided that the information content of the dataset is sufficient. However, as discussed elsewhere (see Sect. 7.2), in some cases this may not be the case, and other limitations may apply (e.g. certain parameters may be undefined), or additional data required for unique determination of the *radial matrix elements*.

For more details on the analysis routines, see the [PEMtk documentation \[20\]](#), particularly the [fit fidelity and analysis page](#), and [molecular frame analysis data processing page](#) (full analysis for Ref. [3], illustrating the N_2 case).

```
# General stats & post-processing to data tables
data.analyseFits()

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues ↴
  ↪with Pandas MultiIndex, but will try anyway.
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues ↴
  ↪with Pandas MultiIndex, but will try anyway.
{'Fits': 390,
 'Minima': {'chisqr': 5.901473330378771e-05, 'redchi': 8.901166410827708e-08},
 'Stats': { 'chisqr': min      5.901e-05
mean      2.760e-04
median    6.259e-05
max      3.133e-02
std       1.935e-03
var       3.746e-06
Name: chisqr, dtype: float64,
      'redchi': min      8.901e-08
mean      4.164e-07
median    9.440e-08
max      4.726e-05
std       2.919e-06
var       8.521e-12
Name: redchi, dtype: float64},
 'Success': 390}
```

```
# The BLMsetPlot routine will output aggregate fit results.
# Here the spread can be taken as a general indication of the uncertainty of
# the fitting, and indicate whether the fit is well-characterised/the information
# content of the data is sufficient.
data.BLMsetPlot(xDim = 't', thres=1e-6) # With xDim and thres set, for more control ↴
  ↪over outputs

# Glue plot for later
glue("C2H4-fitResultsBLM", data.data['plots']['BLMsetPlot'])
```

```
# Write aggregate datasets to HDF5 format
# This is more robust than Pickled data, but PEMtk currently only support output for ↴
  ↪aggregate (post-processed) fit data.

data.processedToHDF5(dataPath = dataPath, outStem = dataFileIn.name, timeStamp=False)
```

```
Dumped self.data[fits][dfLong] to /home/jovyan/jake-home/buildTmp/_latest_build/
  ↪pdf/doc-source/part2/C2H4fitting/C2H4_399_fit_3D-test_withNoise_orb8_270723_13-
  ↪45-22.pickle_dfLong.pdHDF with Pandas .to_hdf() routine.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
  ↪C2H4fitting/C2H4_399_fit_3D-test_withNoise_orb8_270723_13-45-22.pickle_dfLong.
  ↪pdHDF with pdHDF.
Dumped self.data[fits][AFxr] to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/
  ↪doc-source/part2/C2H4fitting/C2H4_399_fit_3D-test_withNoise_orb8_270723_13-45-22-
  ↪pickle_AFxr.pdHDF with Pandas .to_hdf() routine.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
  ↪C2H4fitting/C2H4_399_fit_3D-test_withNoise_orb8_270723_13-45-22.pickle_AFxr.
  ↪pdHDF with pdHDF.
```

(continues on next page)

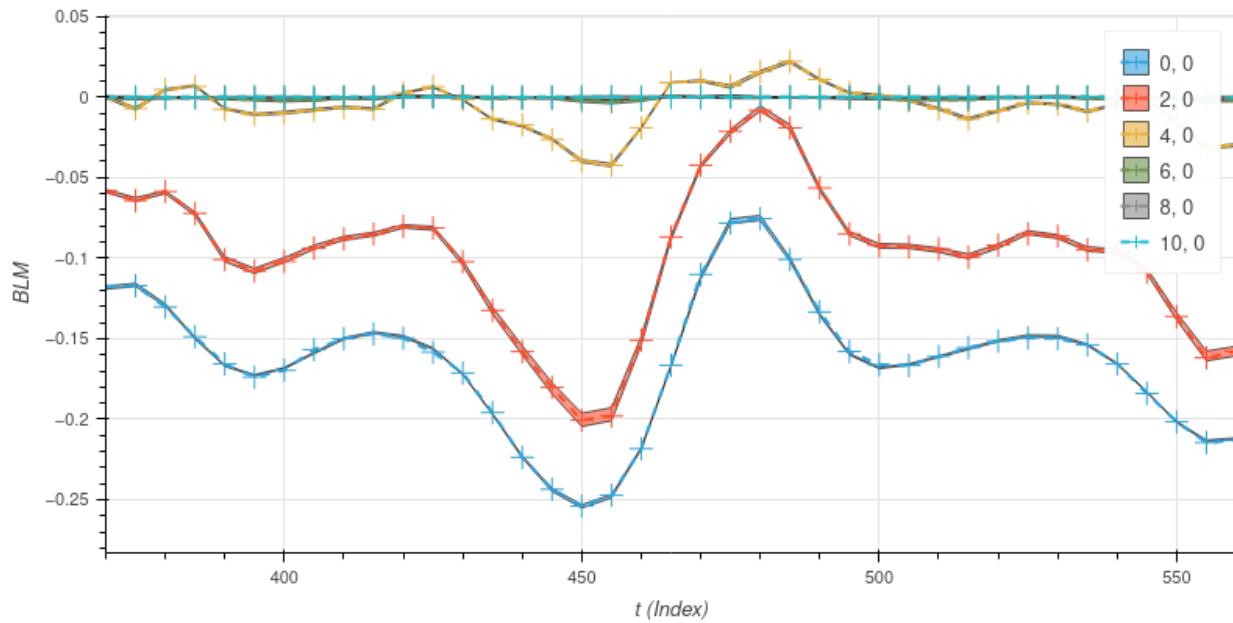


Fig. 13.1: Fit overview plot - $\beta_{L,M}(t)$. Here dashed lines with '+' markers indicates the input data, and bands indicate the mean fit results, where the width is the standard deviation in the fit model results. (See the PEMtk documentation [20] for details, particularly the analysis routines page.)

(continued from previous page)

```
# Histogram fit results (reduced chi^2 vs. fit index)
# This may be quite slow for large datasets, setting limited ranges may help

# Use default auto binning
# data.fitHist()

# Example with range set
data.fitHist(thres=3e-7, bins=100)

# Glue plot for later
glue("C2H4-fitHist", data.data['plots']['fitHistPlot'])
```

```
Mask selected 361 results (from 390).
```

```
:AdjointLayout
:Scatter [redchi] (Fit)
:Histogram [Fit] (Fit_count)
:Histogram [redchi] (redchi_count)
```

Here bands in the χ^2 dimension can indicate groupings (local minima) are consistently found. Assuming each grouping is a viable fit candidate parameter set, these can then be explored in further detail.

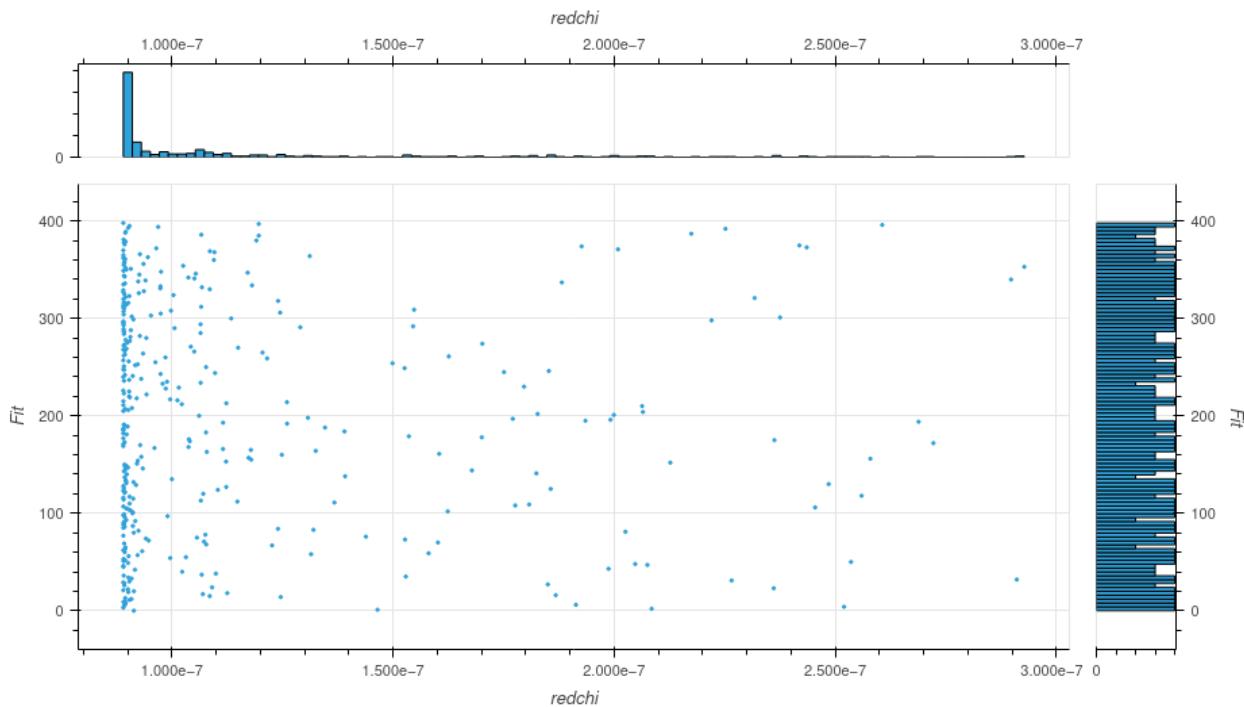


Fig. 13.2: Fit overview plot - χ^2 vs. fit index. Here bands indicate groupings (local minima) are consistently found.

13.4 Data exploration

The general aim in this procedure is to ascertain whether there was a good spread of parameters explored, and a single (or few sets) of best-fit results. There are a few procedures and helper methods for this...

13.4.1 View results

Single results sets can be viewed in the main data structure, indexed by #.

```
# Check keys
fitNumber = 2
data.data[fitNumber].keys()

dict_keys(['AFBLM', 'residual', 'results'])
```

Here `results` is an `ImFit` object, which includes final fit results and information, and `AFBLM` contains the model (fit) output.

An example is shown below. Of particular note here is which parameters have `vary=True` - these are included in the fitting - and if there is a column `expression`, which indicates any parameters defined to have specific relationships (see Chpt. 9). Any correlations found during fitting are also shown, which can also indicate parameters which are related (even if this is not predefined or known a priori).

```
# Show some results
data.data[fitNumber]['results']
```

13.5 Classify candidate sets

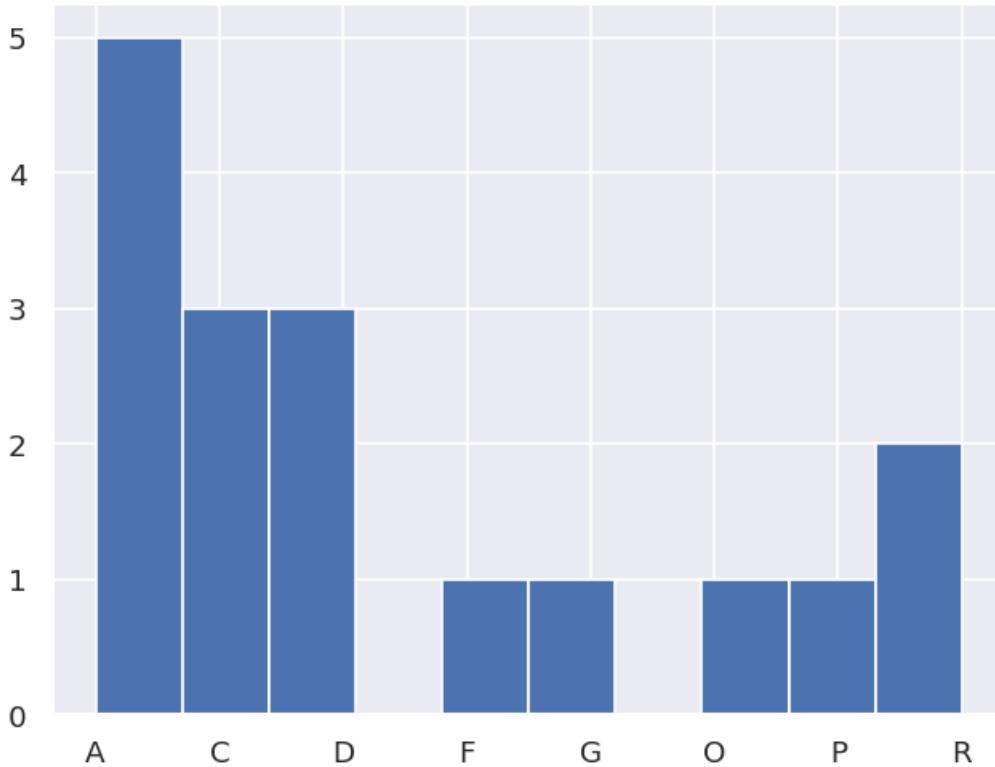
To probe the minima found, the `classifyFits` method can be used. This bins results into “candidate” groups, which can then be examined in detail.

```
# Run with defaults
# data.classifyFits()

# For more control, pass bins
# Here the minima is set at one end, and a %age range used for bins
minVal = data.fitsSummary['Stats']['redchi']['min']
binRangePC = 1e-5
data.classifyFits(bins = [minVal, minVal + binRangePC*minVal , 20])
```

redchiGroup	success			chisqr			redchi					
	count	unique	top	freq	count	unique	top	freq	count	unique	top	freq
A	5	1	True	5	5.0	5.0	0.0	1.0	5.0	5.0	0.0	1.0
B	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
C	3	1	True	3	3.0	3.0	0.0	1.0	3.0	3.0	0.0	1.0
D	3	1	True	3	3.0	3.0	0.0	1.0	3.0	3.0	0.0	1.0
E	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
F	1	1	True	1	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0
G	1	1	True	1	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0
H	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
I	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
J	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
K	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
L	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
M	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
N	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
O	1	1	True	1	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0
P	1	1	True	1	1.0	1.0	0.0	1.0	1.0	1.0	0.0	1.0
Q	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN
R	2	1	True	2	2.0	2.0	0.0	1.0	2.0	2.0	0.0	1.0
S	0	0	NaN	NaN	0	0	NaN	NaN	0	0	NaN	NaN

```
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues ↴
    ↴with Pandas MultiIndex, but will try anyway.
*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues ↴
    ↴with Pandas MultiIndex, but will try anyway.
```



13.6 Explore candidate result sets

Drill-down on a candidate set of results, and examine values and spreads. For more details see [PEMtk documentation \[20\]](#), especially the [analysis routines page](#). (See also [Sect. 5.3](#) for details on the plotting libraries implemented here.)

13.6.1 Raw results

Plot spreads in magnitude and phase parameters. Statistical plots are available for Seaborn and Holoviews backends, with some slightly different options.

```
# From the candidates, select a group for analysis
selGroup = 'A'

# paramPlot can be used to check the spread on each parameter.
# Plots use Seaborn or Holoviews/Bokeh
# Colour-mapping is controlled by the 'hue' parameter, additionally pass hRound for
# sig. fig control.
# The remap setting allows for short-hand labels as set in data.lmmu

paramType = 'm' # Set for (m)agnitude or (p)hase parameters
hRound = 14 # Set for cmapping, default may be too small (leads to all grey cmap on
# points)

data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
```

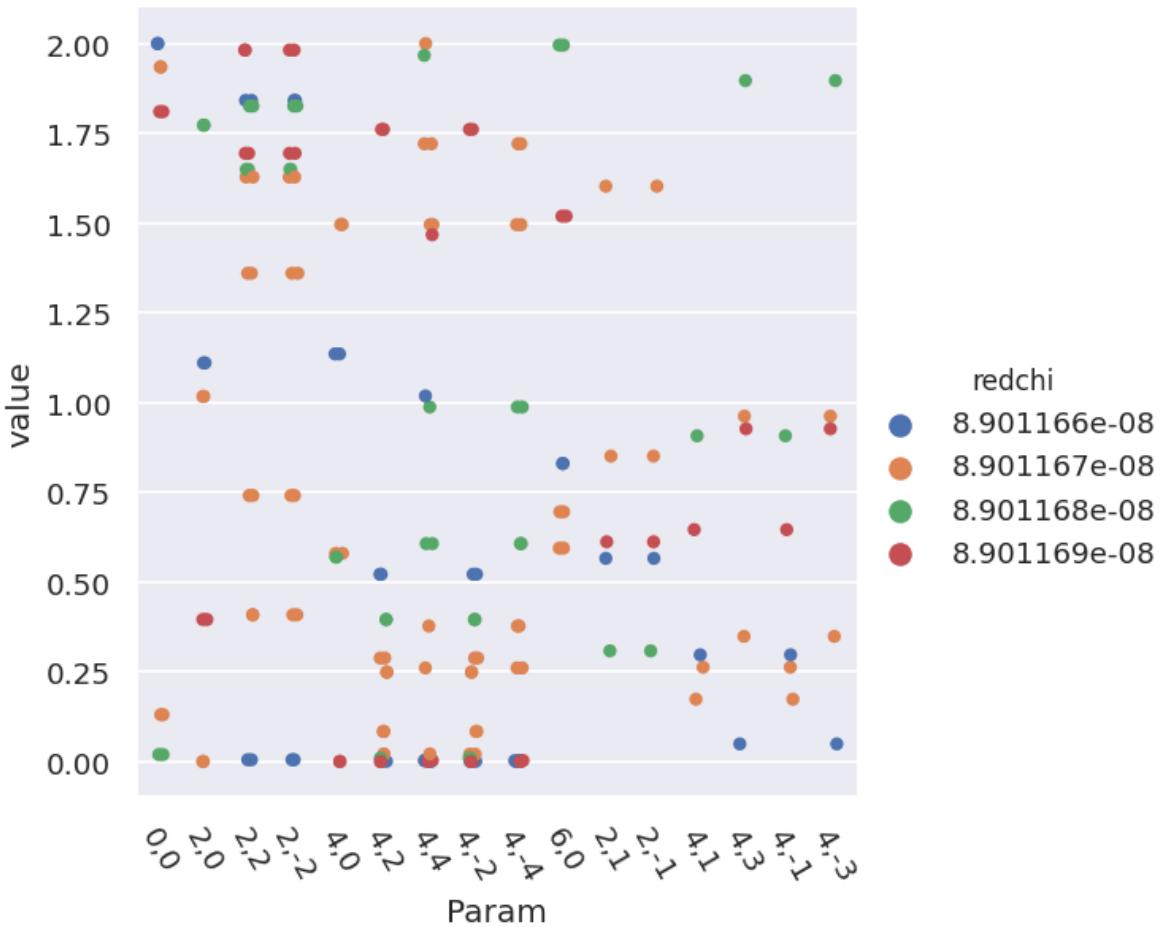
(continues on next page)

(continued from previous page)

```
backend=paramPlotBackend, hvType='violin',
returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.

```
<seaborn.axisgrid.FacetGrid at 0x7f884465c640>
```

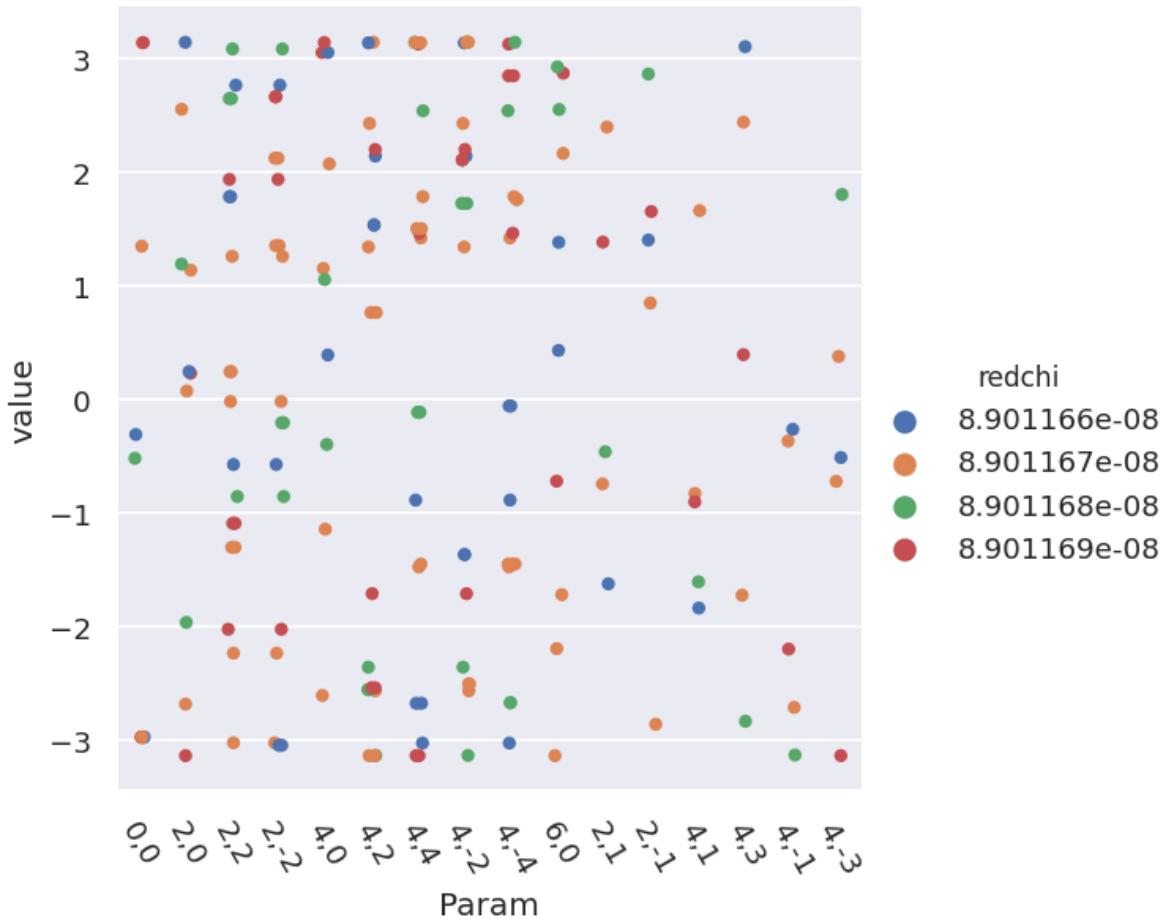


```
paramType = 'p' # Set for (m)agnitude or (p)hase parameters
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',  

    backend=paramPlotBackend, hvType='violin',
    returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.

```
<seaborn.axisgrid.FacetGrid at 0x7f884951ed70>
```



13.6.2 Phases, phase shifts & corrections

Depending on how the fit was configured, phases may be defined in different ways. To set the phases relative to a specific parameter, and wrap to a specified range, use the `phaseCorrection()` method. This defaults to using the first parameter as a reference phase, and wraps to $-\pi : \pi$. The phase-corrected values are output to a new Type, ‘pc’, and a set of normalised magnitudes to ‘n’. Additional settings can be passed for more control, as shown below.

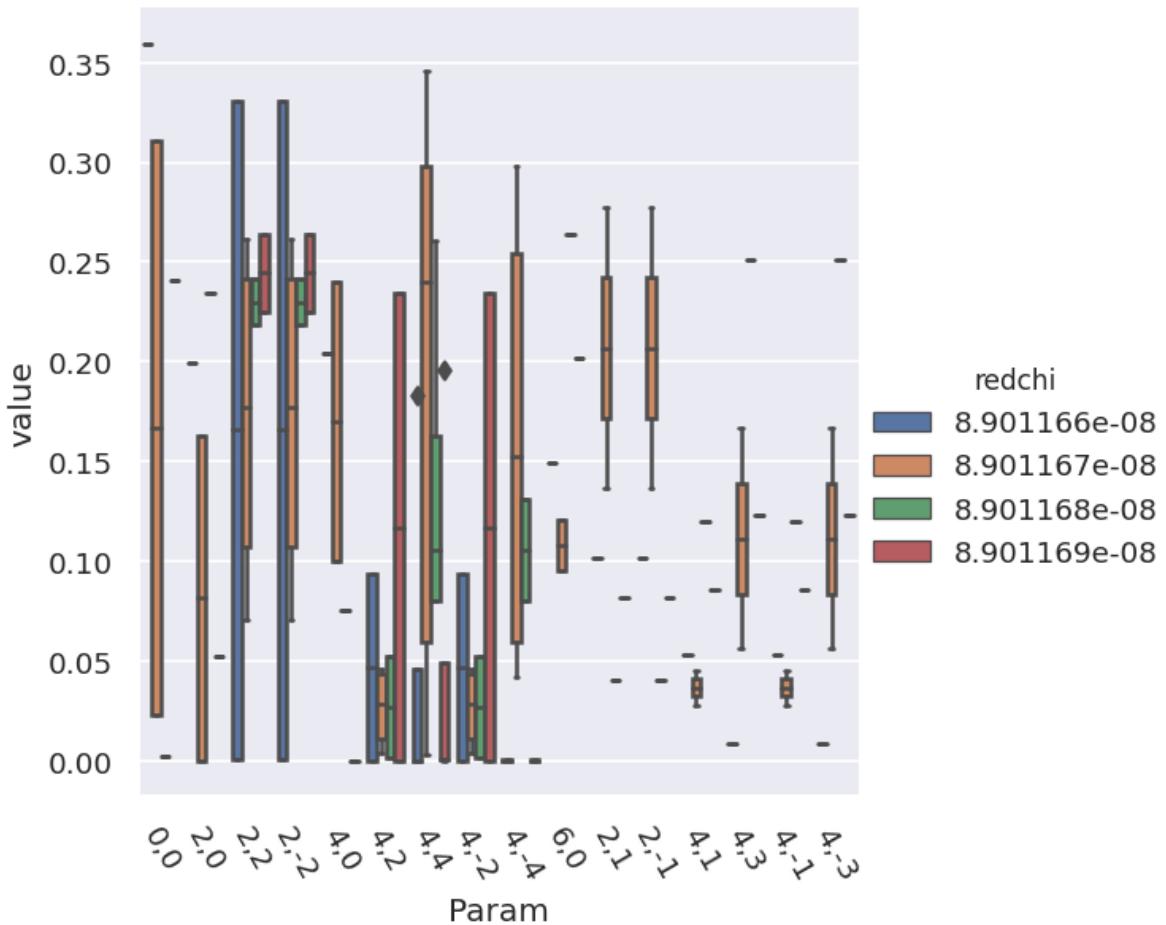
```
# Run phase correction routine
# Set absFlag=True for unsigned phases (mapped to 0:pi)
# Set useRef=False to set ref phase as 0, otherwise the reference value is set.
phaseCorrParams={'absFlag':True, 'useRef':False}
data.phaseCorrection(**phaseCorrParams)
```

Examine new data types...

```
paramType = 'n'
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
               backend=paramPlotBackend, hvType='violin', kind='box',
               returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.

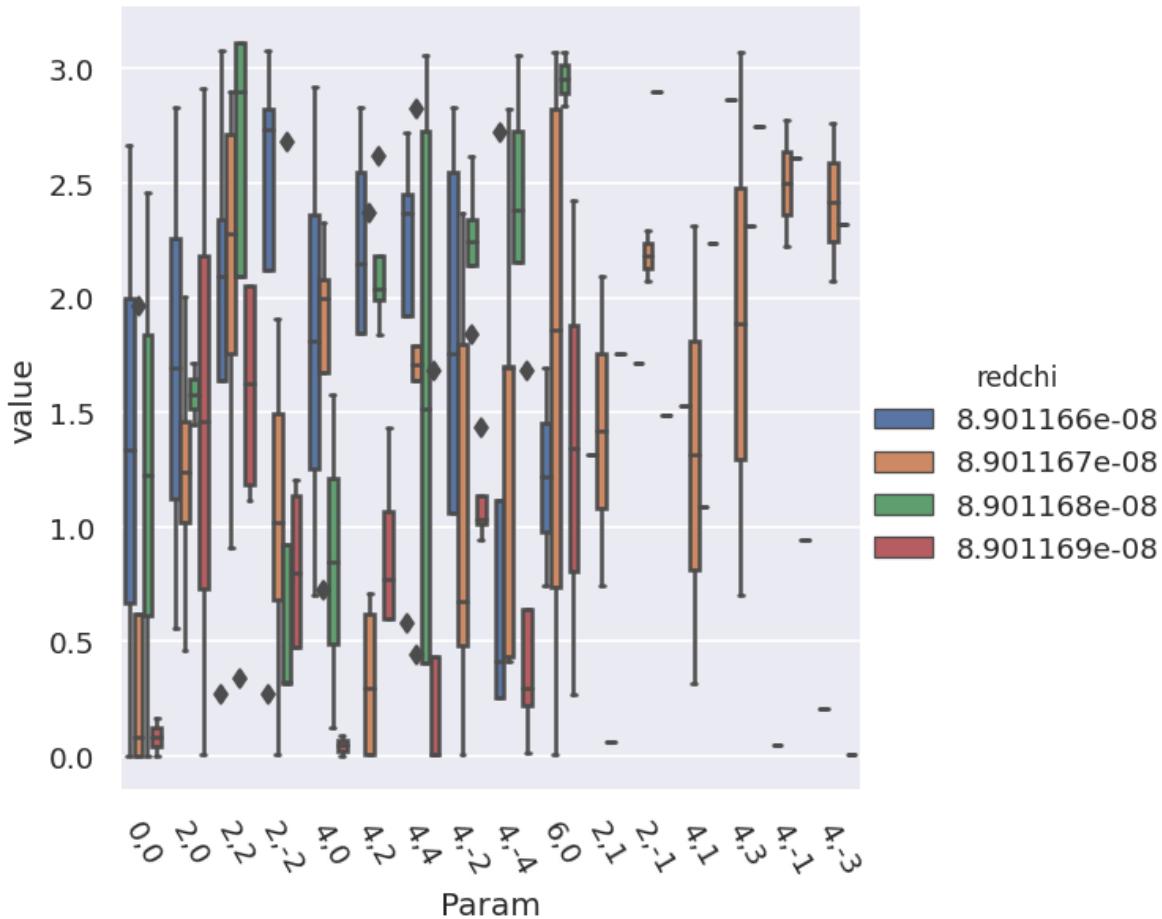
<seaborn.axisgrid.FacetGrid at 0x7f884465c640>



```
paramType = 'pc'
data.paramPlot(selectors={'Type':paramType, 'redchiGroup':selGroup}, hue = 'redchi',
               backend=paramPlotBackend, hvType='violin', kind='box',
               returnFlag = True, hRound=hRound, remap = 'lmMap');
```

*** Warning: found MultiIndex for DataFrame data.index - checkDims may have issues with Pandas MultiIndex, but will try anyway.

<seaborn.axisgrid.FacetGrid at 0x7f884440d930>



13.7 Parameter estimation & fidelity

For case studies, the fit results can be directly compared to the known input parameters. This should give a feel for how well the data defines the matrix elements (parameters) in this case. In general, probing the correlations and spread of results, and comparing to other (unfitted) results is required to estimate fidelity, see *Quantum Metrology* Vols. 1 & 2 [4, 9] for further discussion.

13.7.1 Best values and statistics

To get a final parameter set and associated statistics, based on a subset of the fit results, the `paramsReport()` method is available. If reference data is available, as for the case studies herein, the `paramsCompare()` method can also be used to compare with the reference case.

```
# Parameter summary
data.paramsReport(inds = {'redchiGroup':selGroup})
```

```
# Parameter comparison
# Note this uses phaseCorrParams as set previously for consistency
data.paramsCompare(phaseCorrParams=phaseCorrParams)
```

```

# Display above results With column name remapping to (l,m) labels only

# With Pandas functionality
data.paramsSummaryComp.rename(columns=data.lmmu['lmMap'])

# With utility method
# summaryRenamed = pemtk.fit._util.renameParams(data.paramsSummaryComp, data.lmmu[
#     ↪'lmMap'])
# summaryRenamed

```

Type	Source	Param dType	0,0	0,0	2,0	2,0	2,2	2,2	2,-2	2,-2	4,0
m	mean	num	1.179	1.179	0.859	0.859	1.635	1.635	1.635	1.635	0.756
	ref	num	1.366	1.366	1.299	1.299	1.386	1.386	1.386	1.386	0.400
	diff	%	15.900	15.900	51.143	51.143	15.198	15.198	15.198	15.198	47.063
		num	-0.187	-0.187	-0.439	-0.439	0.248	0.248	0.248	0.248	0.356
	std	%	85.738	85.738	79.694	79.694	10.684	10.684	10.684	10.684	76.209
		num	1.011	1.011	0.685	0.685	0.175	0.175	0.175	0.175	0.576
	diff/std	%	18.545	18.545	64.173	64.173	142.256	142.256	142.256	142.256	61.755
	n	mean	0.187	0.187	0.130	0.130	0.254	0.254	0.254	0.254	0.124
	ref	num	0.268	0.268	0.255	0.255	0.272	0.272	0.272	0.272	0.079
	diff	%	43.240	43.240	95.963	95.963	6.979	6.979	6.979	6.979	36.651
p		num	-0.081	-0.081	-0.125	-0.125	-0.018	-0.018	-0.018	-0.018	0.045
	std	%	88.166	88.166	76.703	76.703	18.057	18.057	18.057	18.057	78.802
		num	0.165	0.165	0.100	0.100	0.046	0.046	0.046	0.046	0.098
	diff/std	%	49.044	49.044	125.111	125.111	38.647	38.647	38.647	38.647	46.511
	mean	num	1.357	-2.978	1.648	-1.497	-0.357	0.413	-0.357	0.413	0.792
	ref	num	0.163	-2.978	-2.385	0.756	0.802	-2.340	0.802	-2.340	2.494
	diff	%	87.971	0.000	244.712	150.525	324.612	666.272	324.612	666.272	214.888
		num	1.194	0.000	4.033	-2.253	-1.159	2.753	-1.159	2.753	-1.702
	std	%	131.140	0.000	71.415	104.662	599.642	567.657	599.642	567.657	219.427
		num	1.780	0.000	1.177	1.567	2.140	2.346	2.140	2.346	1.738
pc	diff/std	%	67.082	NaN	342.663	143.820	54.134	117.372	54.134	117.372	97.931
	mean	num	0.000	1.483	2.132	0.747	1.267	1.682	1.267	1.682	0.727
	ref	num	0.000	3.142	2.548	0.593	0.639	2.503	0.639	2.503	2.331
	diff	%	NaN	111.866	19.526	20.602	49.590	48.847	49.590	48.847	220.734
		num	0.000	-1.659	-0.416	0.154	0.628	-0.821	0.628	-0.821	-1.604
	std	%	NaN	82.832	34.350	80.295	70.160	60.024	70.160	60.024	106.441
		num	0.000	1.228	0.732	0.600	0.889	1.009	0.889	1.009	0.774
diff/std	%	NaN	135.051	56.845	25.658	70.681	81.378	70.681	81.378	81.378	207.378

13.8 Using the reconstructed matrix elements

The results tables are accessible directly, and there are also methods to reformat the best fit results for use in further calculations.

```
# self.paramsSummary contains the results above as Pandas Dataframe, usual Pandas
#methods can be applied.
data.paramsSummary['data'].describe()
```

Param	AG_B3U_B3U_0_0_1_1	AG_B3U_B3U_0_0_n1_1	AG_B3U_B3U_2_0_1_1	AG_B3U_B3U_2_0_n1_1	AG
count	20.000	20.000	2.000e+01	20.000	20.000
mean	0.681	-0.032	1.192e+00	0.060	
std	1.122	1.956	1.058e+00	1.275	
min	-0.523	-2.978	1.742e-05	-3.142	
25%	0.000	-0.743	2.197e-01	0.002	
50%	0.077	0.167	1.122e+00	0.217	
75%	1.461	1.841	1.830e+00	0.670	
max	3.139	2.665	3.141e+00	1.773	

```
# To set matrix elements from aggregate fit results, use `setAggMatE` for Pandas
data.setAggMatE(simpleForm = True)
data.data['agg']['matEpd']
```

Set reformatted aggregate data to self.data[agg][matEpd].

Cont	1	m	Type mu	m	n	p	pc	comp	compC	labels
AG	0	0	-1	1.179	0.187	1.357	0.000	0.250+1.152j	0.187+0.000j	0,0
			1	1.179	0.187	-2.978	1.483	-1.163-0.192j	0.016+0.186j	0,0
	2	0	-1	0.859	0.130	1.648	2.132	-0.066+0.857j	-0.069+0.110j	2,0
			1	0.859	0.130	-1.497	0.747	0.063-0.857j	0.095+0.088j	2,0
		-2	-1	1.635	0.254	-0.357	1.267	1.532-0.571j	0.076+0.243j	2,-2
			1	1.635	0.254	0.413	1.682	1.497+0.656j	-0.028+0.253j	2,-2
	2	-1	1.635	0.254	-0.357	1.267	1.532-0.571j	0.076+0.243j	2,2	
			1	1.635	0.254	0.413	1.682	1.497+0.656j	-0.028+0.253j	2,2
	4	0	-1	0.756	0.124	0.792	0.727	0.531+0.538j	0.093+0.082j	4,0
			1	0.756	0.124	1.156	1.762	0.305+0.692j	-0.024+0.122j	4,0
		-2	-1	0.539	0.075	0.366	1.662	0.503+0.193j	-0.007+0.075j	4,-2
			1	0.539	0.075	0.552	1.378	0.459+0.283j	0.014+0.074j	4,-2
	2	-1	0.765	0.124	-0.060	2.062	0.764-0.046j	-0.058+0.109j	4,2	
			1	0.765	0.124	1.382	1.406	0.144+0.752j	0.020+0.122j	4,2
		-4	-1	0.539	0.075	0.366	1.662	0.503+0.193j	-0.007+0.075j	4,-4
			1	0.539	0.075	0.552	1.378	0.459+0.283j	0.014+0.074j	4,-4
	4	-1	0.765	0.124	-0.060	2.062	0.764-0.046j	-0.058+0.109j	4,4	
			1	0.765	0.124	1.382	1.406	0.144+0.752j	0.020+0.122j	4,4
	6	0	-1	1.127	0.166	0.196	1.431	1.105+0.220j	0.023+0.165j	6,0
			1	1.127	0.166	0.708	2.133	0.856+0.733j	-0.089+0.141j	6,0
B1G	2	-2	-1	0.993	0.139	0.453	2.561	0.892+0.435j	-0.116+0.076j	2,-2
			1	0.993	0.139	0.453	2.561	0.892+0.435j	-0.116+0.076j	2,-2
		2	-1	0.993	0.139	0.574	0.910	0.833+0.539j	0.085+0.110j	2,2
			1	0.993	0.139	0.574	0.910	0.833+0.539j	0.085+0.110j	2,2
	4	-2	-1	0.127	0.022	-1.190	1.015	0.047-0.118j	0.012+0.019j	4,-2
			1	0.127	0.022	-1.190	1.015	0.047-0.118j	0.012+0.019j	4,-2
		2	-1	0.326	0.048	-0.260	1.241	0.315-0.084j	0.015+0.045j	4,2
			1	1.295	0.198	-0.260	1.241	1.251-0.333j	0.064+0.187j	4,2
		-4	-1	0.127	0.022	0.618	1.354	0.104+0.074j	0.005+0.022j	4,-4
			1	0.127	0.022	0.618	1.354	0.104+0.074j	0.005+0.022j	4,-4
	4	-1	0.326	0.048	0.082	0.960	0.325+0.027j	0.027+0.039j	4,4	
			1	0.326	0.048	0.082	0.960	0.325+0.027j	0.027+0.039j	4,4
B2G	2	-1	0	0.788	0.128	0.187	1.193	0.774+0.147j	0.047+0.119j	2,-1
		1	0	0.788	0.128	0.778	2.093	0.561+0.553j	-0.064+0.111j	2,1
	4	-1	0	0.457	0.067	-0.706	1.496	0.348-0.297j	0.005+0.066j	4,-1
		1	0	0.837	0.121	0.273	2.341	0.806+0.225j	-0.084+0.087j	4,1
		-3	0	0.457	0.067	-1.739	1.719	-0.076-0.451j	-0.010+0.066j	4,-3
	3	0	0.837	0.121	-0.442	1.473	0.757-0.358j	0.012+0.121j	4,3	

```
# To set matrix elements from aggregate fit results, use `aggToXR` for Xarray
# data.aggToXR(refKey = 'orb5', returnType = 'ds', conformDims=True) # use full ref
# dataset
data.aggToXR(refKey = 'subset', returnType = 'ds', conformDims=True) # Subselected
# mate
```

```
Added dim Total
Added dim Targ
Added dim Total
Added dim Targ
```

(continues on next page)

(continued from previous page)

```
Set XR dataset for self.data['agg']['matE']
```

13.8.1 Density matrices

New (experimental) code for density matrix plots and comparison. See Sect. 6.4 for discussion. Code adapted from the PEMtk documentation [20] MF reconstruction page, original analysis for Ref. [3], illustrating the N_2 case. If the reconstruction is good, the differences (fidelity) should be on the order of the experimental noise level/reconstruction uncertainty, around 10% in the case studies herein; in general the values and patterns of the matrices can also indicate aspects of the retrieval that worked well, or areas where values are poorly defined/recovered from the given dataset.

13.8.2 Plot MF PADs

Routines as per https://pemtk.readthedocs.io/en/latest/topical_review_case_study/MFPAD_replotting_from_file_190722-dist.html - currently not working. Seems to be some difference in dim stacking/assignment now...? Might be Python/Xarray version change, or PEMtk/ePSproc implementation.

```
# Compute MFPADs for a range of cases

# Set Euler angs to include diagonal pol case
pRot = [0, 0, np.pi/2, 0]
tRot = [0, np.pi/2, np.pi/2, np.pi/4]
cRot = [0, 0, 0, 0]
labels = ['z','x','y', 'd']
eulerAngs = np.array([labels, pRot, tRot, cRot]).T      # List form to use later, rows_
→per set of angles

# Should also use MFBLM function below instead of numeric version?
# Numeric version is handy for direct surface and difference case.
R = ep.setPolGeoms(eulerAngs = eulerAngs)

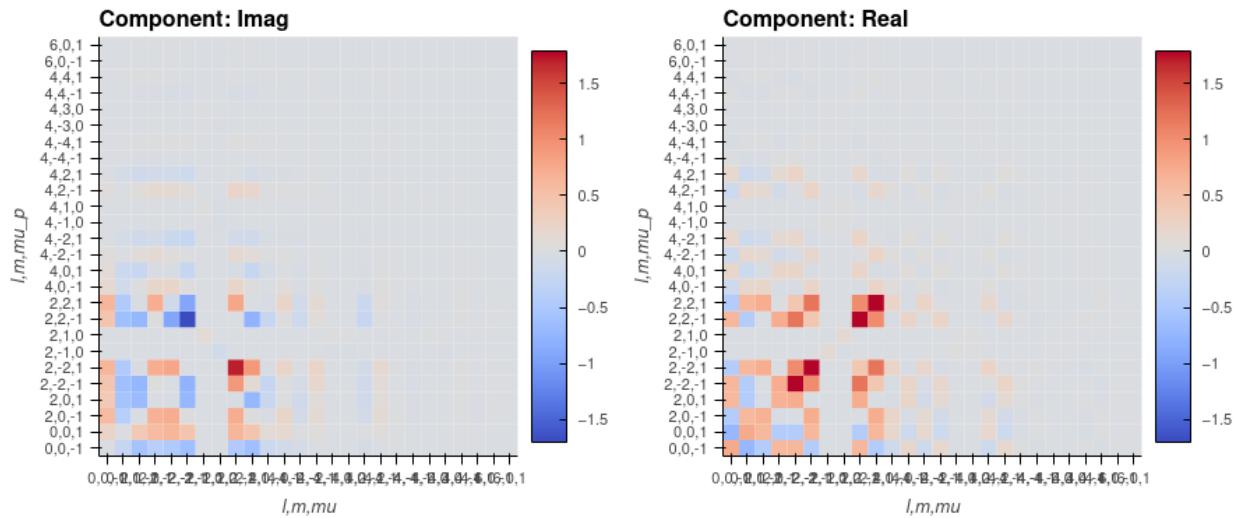
# Comparison and diff
pKey = [i for i in dataIn.data_vars if i != 'comp']    # List of arrays
dataTest.mfpadNumeric(keys=pKey, R = R)    # Compute MFPADs for each set of matrix_
→elements using numerical routine

dataTest.data['diff'] = {'TX': dataTest.data['subset']['TX'].sum('Sym')-dataTest.data[_
→'compC']['TX'].sum('Sym')}    # Add sum over sym to force matching dims
pKey.extend(['diff'])

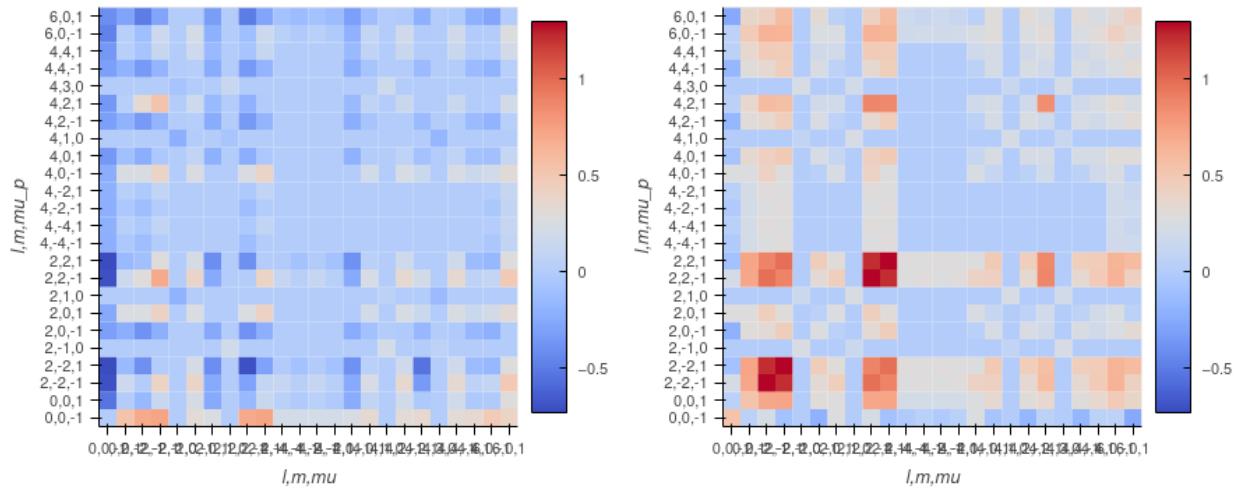
# Plot - all cases
# Now run in separate cells below for more stable output
# Erange=[1,2,1]    # Set for a range of Ekes
# Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']}    # Plot for selected Eke (as_
→used for fitting)
# print(f"\n*** Plotting for keys = {pKey}, one per row ***\n")    # Note plot labels_
→could do with some work!
# dataTest.padPlot(keys=pKey, Erange=Erange, backend='pl', returnFlag=True,_
→plotFlag=True) # Generate plotly polar surf plots for each dataset

# Change default plotting config, and then plot in separate cells below
ep.plot.hvPlotters.setPlotters(width=1000, height=500)
```

(a) Reference density matrix (unsigned phases)



(b) Reconstructed



(c) Difference

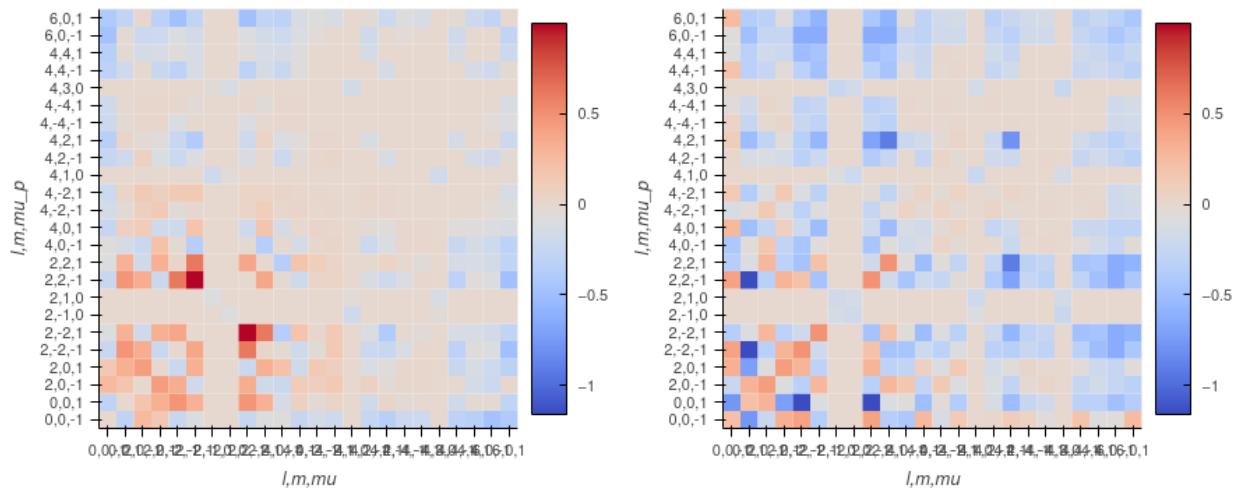


Fig. 13.3: Density matrix comparison - rows show (a) reference case (with signs of phases removed), (b) reconstructed case, (c) differences. Columns are (left) imaginary component, (right) real component. If the reconstruction is good, the differences (fidelity) should be on the order of the experimental noise level/reconstruction uncertainty, around 10% in the 13.8 Using the reconstructed matrix elements

```
# Plot results from reconstructed matE
pKey = 'compC'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpt['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("C2H4-compC", figObj)
```

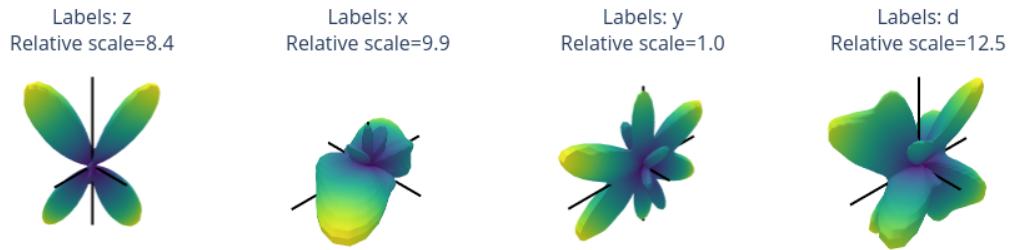


Fig. 13.4: *MF-PADs* computed from retrieved matrix elements for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis.

```
# Plot results from reference matE
pKey = 'subset'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
Eplot = {'Eke':data.selOpt['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("C2H4-ref", figObj)
```

```
# Plot normalised differences
pKey = 'diff'
print(f"\n*** Plotting for keys = {pKey} ***\n") # Note plot labels could do with
# some work!
# dataTest.padPlot(keys=pKey, Erange=Erage, backend='pl', returnFlag=True,
# plotFlag=True) # Generate plotly polar surf plots for each dataset
```

(continues on next page)

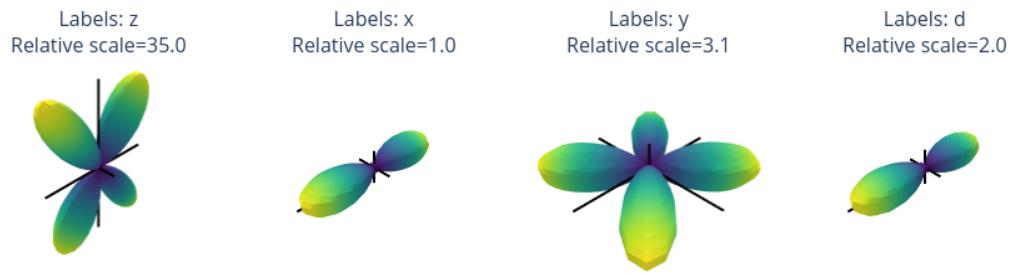


Fig. 13.5: *MF-PADs* computed from reference *ab initio* matrix elements for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis.

(continued from previous page)

```
Eplot = {'Eke':data.selOpts['matE']['inds']['Eke']} # Plot for selected Eke (as used
# for fitting)
dataTest.padPlot(keys=pKey, selDims=Eplot, backend='pl', returnFlag=True,
plotFlag=True) # Generate plotly polar surf plots for each dataset

# And GLUE for display later with caption
figObj = dataTest.data[pKey]['plots']['TX']['polar'][0]
glue("C2H4-diff", figObj)
```

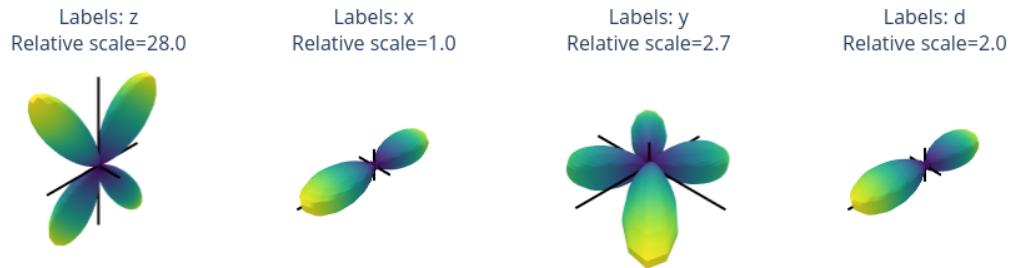


Fig. 13.6: *MF-PADs* differences between retrieved and reference cases for (x, y, z, d) polarization geometries, where d is the “diagonal” case with the polarization axis as 45 degrees to the z -axis. Note diffs are normalised to emphasize the shape, but not magnitudes, of the differences - see the density matrix comparisons for a more rigorous fidelity analysis.

CHAPTER
FOURTEEN

CASE STUDIES: SUMMARIES, CONCLUSIONS AND OUTLOOK

In the preceding chapters, three case studies were demonstrated. As already noted elsewhere (Chpt. 8), the N_2 case has already been demonstrated to be successful for a range of energies and ionization channels (orbitals), see Refs. [1, 3] for further details. However, it is included here as a benchmark case, useful for testing the methodology, computational routines and limitations and so forth. In particular, it represents a case of a reasonable level of complexity for method development, without an excessive number of *radial matrix elements*/fitting parameters to deal with.

The other two case studies (Chpt. 12 & Chpt. 13) are presented in a more experimental/preliminary capacity. In both cases, a much larger number of *radial matrix elements* are required, and a higher-information content dataset with 3D alignment (as tested) and/or multiple polarization geometries, is expected to be required for a high-fidelity *bootstrap retrieval protocol* to be successful. In the current cases as presented, some progress towards this has been achieved, although further work remains to improve on the current results. Nonetheless, as stated previously, these new results are interesting and present a stepping-stone for studies on *radial matrix elements* retrieval in complex systems, and a launching point for similar studies.

14.1 General notes on fitting methodologies for the case studies

As discussed elsewhere, the general *bootstrap retrieval protocol* is quite flexible. In the case studies, much of this flexibility is yet to be explored! This is, primarily, a question of the time and effort available: for the N_2 case obtaining a set of 1000 fit results takes approximately 2 hours on a workstation.¹ However, scaling up to more complex cases is significantly more costly, with OCS and C_2H_4 results requiring approximately 1-2 orders of magnitude more computational time - on the order of days per few hundred fits. Work towards speeding up fitting, e.g. using GPU-based routines and/or lower-level code for speed, is ongoing. Deployment to high-performance computing (HPC) resources, e.g. clusters, is also ongoing. The results herein, therefore, represent only the preliminary stages of testing and tuning the *bootstrap retrieval protocol* for these more complex cases, but already appear quite promising.

In each case, as detailed in Chpt. 9 & Chpt. 10, the fitting basis sets were created based on *ab initio* results, with all *radial matrix elements* greater than a threshold value of 0.01 used for both simulated data generation and fitting, and automatic parameter constraints applied to reduce the effective number of “free” or “floated” terms in the fitting. Fitting was only tested for single energy-points. The overall size of the problems as run in this manner were:

Sys-tem	Sym-metry	Energy (eV)	Complex matrix ele-ments	Fitting params	Floated params (magnitude, phase)	t-points
N_2	$D_{\infty h}$	1	6	12	4,3	13
OCS	$C_{\infty v}$	10	22	44	15,14	51
C_2H_4	C_{2h}	6	38	76	14,25	73

¹ Running on 20 (logical) cores of an AMD Threadripper 2950X based workstation, this required 5 GB of RAM and took on the order of 2 hours, although note that the time per fit cycle had large variance, since convergence time depends on the start parameters. Further benchmarks for the current codebase can be found in the PEMtk documentation [20].

In both of the more complex cases, the computational testing involved starting with the protocol as used for the N_2 reference case, then adding data points until clustering was observed in the χ^2 histograms, indicating that multiple minima were successfully located. This is a necessary first step in testing, and signifies that the methodology is working as expected; however, this is not sufficient to guarantee that the information content of the dataset was adequate for complete *radial matrix elements* retrieval, nor to guarantee a high-fidelity retrieval (herein “complete” is used to indicate that a unique set of fit parameters may be found, whilst the “fidelity” is the more stringent test of the veracity of these parameters). In fitting, automatic parameter relations were set, which fixed equivalent parameters (see Chpt. 9 & Chpt 10 for details). For N_2 and *OCS* no additional constraints were imposed; for C_2H_4 some additional magnitude-only constraints were set for parameters with identical magnitudes which were not set by the automated routine, although corresponding phase relations were *not* set (see tabulations in Chpt. 12 for full details) - this case is therefore an interesting test of whether the correlated phase relations will be correctly recovered by the *bootstrap retrieval protocol*. It is, however, anticipated that for high-fidelity retrieval, additional constraints may be required on a case-by-case basis, and particularly for large problems, e.g. by fixing additional symmetry relations, making use of additional data and so forth - this is discussed further below.

In each case, the lowest χ^2 cluster was assumed to be the “best” result, i.e. the true set of *radial matrix elements*, and further tested for spread and veracity, as compared with the known inputs. This is an easy methodology to implement computationally although, in general, one may wish to inspect several sets of candidate fit parameters in cases with noise and/or large solution spaces, since the lowest χ^2 may not, in fact, be the best, and a true global minima may not exist. Similarly, the presence of multiple equivalent minima is possible in cases where some quantities are undefined, which may lead to issues with phase retrieval in particular (see *Quantum Metrology* Vols. 1 & 2 [4, 9] for further discussion) - with the current codebase this can be probed by changing the binning of candidate fit results, and examining the spread of parameters for a given choice of binning, as outlined in the case studies. (In the case studies, the retrieved matrix elements are taken as an average over the best cluster of fits, hence large spread in a given parameter for the chosen cluster indicates a case where the given parameter is not well defined - this is likely to appear as a clear deficiency in the corresponding density matrix and/or *MF-PADs* when assessing the fidelity of the retrieval.) The likelihood of a good fit result will also depend on the number of data-points used in fitting, as well as the underlying alignment and molecular symmetry properties - generally the form of the *channel functions* as discussed in Sect. 6.3 - which have not been carefully investigated in these cases as yet.

For the N_2 case, the *RWP* is as detailed in Refs. [1, 3], which made use of a two-pulse alignment scheme. Data-points over the *RWP* half-revival feature were chosen initially to approximately match the previously published cases (13 t -points in the current case, vs. 11 in the original study, although that study also made use of larger datasets up to 89 t -points for higher-fidelity *bootstrap retrieval protocol*, see Ref. [1] supplementary materials for further details).

For the *OCS* case, the *RWP* is as per obtained in recent ultrafast experiments [144], which made use of a single-pulse alignment scheme to prepare a 1D molecular alignment (no orientation). For testing purposes, an (arbitrary) number of low-order terms $Q \neq 0$ and $S \neq 0$ were added for the work herein (with a linear ramp in t), including $K = 1$ terms for orientation (see figures in Chpt. 12 for full details). Although these are not expected to be physically realistic, they will be indicative of the terms present in a true 3D alignment case. Note that the temporal data used in this case is discontinuous, with t -points over both the 1/4 and 1/2 *RWP* revival features selected.

For the C_2H_4 case, the *RWP* is a realistic case, as used in Ref. [98], which includes $S \neq 0$ terms and even- K even- S terms (see figures in Chpt. 13 for full details). Interestingly, this case was sufficient for retrieval of *MF-PADs* using the matrix-inversion method, so is expected to be successful with the *bootstrap retrieval protocol* provided enough data-points are incorporated and the fitting is well-constrained. Note that the temporal axis in this case is in arbitrary units.

14.2 Retrieval fidelity

The continuum density matrices provide a way to quickly assess the *bootstrap retrieval protocol* fidelity in these test cases, without the necessity of consulting rather complex tabulations of *radial matrix elements*/fitting parameters and the associated statistics. In general, it is expected that the *differences* between the true and reconstructed density matrices will be on the order of the experimental noise in a high-fidelity case. For each case study, such a comparison is given - see Fig. 11.3 (N_2 case), Fig. 12.3 (OCS case), and Fig. 13.3 (C_2H_4 case) (see *Quantum Metrology Vol. 3 (HTML version)* for interactive plots). Note that in the comparisons herein the sign of the phases is assumed to be undefined, so phases are remapped to positive values only (i.e. assumed to be in the $0 : \pi$ range). Depending on the symmetry of the problem (hence the form of the *channel functions*), this may or may not be the case in practice - in particular cases with multiple polarisation geometries and/or 3D alignment may break this assumption.

For N_2 , the recovered and density matrices are in good agreement, and differences in both the real and imaginary components are on the order of the noise-floor in the simulated dataset ($\approx \pm 10\%$). This indicates a high-fidelity retrieval from only a small number of data-points (13 in the example case), and also indicates that improvements may be found here by incorporating additional data into the fitting procedure.

For OCS (Fig. 12.3) and C_2H_4 (Fig. 13.3) cases, a number of general comments may be made. Firstly, the overall patterns of values observed in the reconstructed density matrices are in good agreement with the reference case, however the agreement in the absolute values (hence fidelity) are quite variable. For OCS differences as large as 0.4 are observed, whilst for C_2H_4 differences as large as 1 are observed. Secondly, some patterns are inverted in the retrieved cases, indicating issues with the signs of the retrieved quantities. This is likely indicative of missing phase relations in the data (i.e. *channel functions*), or indicates that multiple minima exist in the χ^2 hypersurface with differences in the phases. Thirdly, the retrieved *radial matrix elements* tend to have, in general, larger values in both real and imaginary components for the higher-order terms (larger l) than the reference values: this can be seen in the fading (vignetting) of the colourmaps to higher-order in the reference cases, which is not present in the retrieved maps.

In general, these issues all suggest that the *bootstrap retrieval protocol* is partially working for these cases, but may not yet be complete. A number of avenues remain to be explored here, in particular:

- The analysis of other candidate parameter sets to check for uniqueness/equivalence.
 - This may require deeper analysis of the existing results with finer χ^2 binning than the current cases, which indicate significant spreading in some parameters (see the *Raw results* and *Phases* subsections in each case study chapter, particularly the parameter plots which indicate the spread in results).
 - This analysis is not yet automated, but this may be possible by setting preferences on spreading of results to determine optimum grouping of results.
- Further bootstrapping with larger datasets and/or using the existing best fit parameters as seed values to further probe the local χ^2 surface.
- Fitting with reduced parameter sets. These were set based on the reference matrix elements with a threshold for inclusion of 0.01 (see Chpt. 10), but lowering this to 0.1 would remove many of the higher-order terms from the fitting, which may result in higher-fidelity retrieval of the lower-order terms which then remain.
- Fitting with more constraints, e.g.
 - Setting lower limits on higher-order terms. Although, in practice, this may not be known *a priori* for a given case, as a general rule-of-thumb it should be applicable (as indicated by the *ab initio* results for the cases herein).
 - Enforcing orthogonality on different continua, as per the symmetrized harmonics defining each case. For the C_2H_4 case in particular, this would add additional phase restrictions which were missing in the current study.
- Further testing with simulated data with both more and less noise present to evaluate the effect of the noise-floor in different cases.
- Finally, it is of note (and also noted elsewhere) that there may still be bugs or numerical issues with the preliminary results presented herein, since they represent the first trials of the *Photoelectron Metrology Toolkit* [5] and *bootstrap*

retrieval protocol for complex cases. However, given that the same routines are used to generate the simulated data, any issues should, at least, be self-consistent for these test cases. Eagle-eyed readers may notice that in the C_2H_4 test case, the $\beta_{0,0}(t)$ parameters are negative, which is unphysical; this indicates a phase error in the calculation and has since been fixed, but does *not* affect the fitting protocol thanks to the self-consistent nature of the numerics.

14.3 MF-PAD retrieval

The *MF-PADs* also provide an interesting observable to test from the retrieved *radial matrix elements* - they provide less detail than the density matrices, but do give a better indication of the sensitivity of the observables to the retrieved parameters (via the *channel functions*) in a given case. For example, some phase relations may not be important for *MF* retrieval, or only appear under certain polarization geometries. For both the N_2 and C_2H_4 cases the work in Ref. [98] provides an interesting perspective here, since it outlines a direct *MF* reconstruction method which bypasses the difficulty of phase-retrieval and fitting via a matrix-inversion protocol. This is found to work well for both these cases, although in a similar manner to a fitting approach will be fundamentally limited in any given case by the symmetry of the problem.

For the case studies, the reconstructed *MF-PADs* are somewhat variable, as may be expected from the preceding discussion on fidelity. For N_2 (Fig. 11.4 - Fig. 11.6), the results are excellent - as expected from the previous studies in this case. In particular, the good agreement for the diagonally-polarized case indicates that the relative phase *between* the two continua is defined in this type of data, and successfully retrieved. The normalised difference plots in Fig. 11.6 indicate that the main differences between the retrieved and reference cases can be observed in some of the smaller lobes, indicating that the fidelity is high, but not perfect. As discussed in the previous section, adding data and analysing the effect of noise on the *bootstrap retrieval protocol* may improve on the retrieval fidelity, and this was previously explored for N_2 in Ref. [1].

For the more complex cases, the fidelity is lower, and the results generally suggest that, although the dominant continuum is fairly well recovered, the loss of phase information and/or generally lower information content of the data is an issue. For *OCS* (Fig. 12.4 - Fig. 12.6), the (x, y) *MF-PADs* are close to the reference case, although the lobes are significantly sharper. The z case, however, is very different, with the main lobe in the $-z$ direction, as opposed to $+z$ in the reference case. This indicates that the terms related to breaking up-down symmetry are not well reproduced in this case, and is most likely an issue with the associated phases; as noted above this may be due to issues with the analysis and binning of the fit results, or a more fundamental limitation due to the dataset and/or symmetry of the problem - potentially with the choice of additional terms added to the *ADM*s - and more work is required here to ascertain this.

For C_2H_4 (Fig. 13.4 - Fig. 13.6), the opposite is true: the z case is fairly well reproduced, although rotated, whilst the (x, y) cases are in poor agreement with the reference results. Again it is the dominant continuum which is best reproduced; however, in this case the discrepancies in all polarization geometries appear as additional symmetry breaking, which would not be present in the correctly symmetrized (orthogonal continua) case - as seen in the reference results (Fig. 13.5). This indicates that the respective continua are not well-separated in the recovered *radial matrix elements*, i.e. the phase relations are such that additional symmetry breaking is seen. As for the *OCS* case, this may be due to averaging over multiple minima in the fitting phase space, and/or may relate to a lack of information in the test dataset. In this case, given the results of Ref. [98], it is anticipated that the dataset is sufficient, and that most issues arise from the complexity of the χ^2 hypersurface and the configuration of the *bootstrap retrieval protocol*, and/or post-processing of the results. As discussed above, in the current test case some phase relationships between parameters were not constrained, and this is likely the main source of the discrepancy; ideally the fitting should recover these relations in general, but this will, again, depend on the form of the *channel functions* and the dataset used, which will ultimately determine which phase relations are defined and therefore recoverable. Probing this behaviour in general, and in this case, in more depth is currently underway. One way to probe this type of issue further is to define specific orthogonality relations in the fitting procedure via symmetrized matrix elements. This would provide a means to enforce the distinction between the continua without the necessity of *ab initio radial matrix elements* (this is possible with the current codebase, but has not been explored for this case as yet). This would also be consistent with the success of the approach in Ref. [98], since symmetry is rigorously enforced in the mathematics in the matrix-inversion method.

In all cases, the “diagonal” polarization case is another good test of the phase relations, since this is also sensitive to the phase relations between the z and (x, y) continua. For N_2 the good agreement of the results with the reference case is, again, indicative of the generally good fidelity of the reconstruction in this case. Similarly, the differences in the more

complex OCS and C_2H_4 cases are consistent with the previous discussion: the main difference between these cases is the additional symmetry breaking in the lower-symmetry C_2H_4 case, leading to the appearance of an overall rotation and additional lobes, particularly directed out of the (x, z) plane, relative to the reference case. Again, this is indicative of continua mixing, arising from incorrect/undefined/low-fidelity phase terms.

14.4 Conclusions and outlook

Overall, the case studies provide interesting material for a number of reasons. First and foremost, they indicate that the *bootstrap retrieval protocol* method is applicable to larger problems (38 *radial matrix elements* in the largest case), at least in principle. Secondly, they indicate the issues in these cases, which still remain exceedingly complicated retrieval problems, despite the generality and automation of the *bootstrap retrieval protocol* implemented in the *Photoelectron Metrology Toolkit* [5]. In these specific cases the route to complete and/or higher-fidelity *bootstrap retrieval protocol* is clear, but significant efforts are still required to conclusive demonstrate this, and develop more efficient and effective fitting strategies for large solution hyperspaces. Nonetheless, this is an interesting and notable step towards a general method for these problems, and (equivalently) density matrix retrieval and general quantum tomography on larger systems (e.g. 38x38 density matrix retrieval in the C_2H_4 case).

Future work is planned to look at these specific problems in more detail, including different energy regions and ionization channels, and other small molecules. As part of this effort, HPC resources will be used to allow for scaling up of the computational effort available, and development of the *Photoelectron Metrology Toolkit* [5] and *ePSproc codebase* [33, 34, 35] will continue in tandem with these efforts. Fitting strategies, algorithms and methodologies also remain as a large area to be explored, in particular cross-fertilization from other fields dealing with large computational hyperspaces and complex phase-retrieval problems is expected to (continue) to prove fruitful (see *Quantum Metrology* Vols. 1 & 2 [4, 9] and Ref. [3] for futher discussion along these lines).

As discussed in Chpt. 5.1, it is also hoped that the efforts on the code-development side of the problem, including the case studies presented herein, the open-source nature of the work, and the easy deployment of the full software stack via Docker, will all encourage other researchers to make use of these tools, and make further developments to the methodology and platform. Given the complex nature of both the core physics, data processing, simulation and phase retrieval problems, there are still many avenues to explore, but hopefully the barrier to entry is now significantly lowered.

Part IV

Backmatter

CHAPTER
FIFTEEN

BIBLIOGRAPHY

CHAPTER SIXTEEN

GLOSSARY

MF

Molecular frame (MF) - coordinate system referenced to the molecule, usually with the z-axis corresponding to the highest symmetry axis. See [Sect. 6.3.3](#) for further details.

LF

Laboratory or lab frame (LF) - coordinate system referenced to the laboratory frame, usually with the z-axis corresponding to the laser field polarization. For circularly or elliptically polarized light the propagation direction is conventionally used for the z-axis. In some cases a different z-axis may be chosen, e.g. as defined by a detector. See [Sect. 6.3.3](#) for further details.

AF

Aligned frame (AF) - coordinate system referenced to molecular alignment axis or axes. For 1D alignment, the z-axis usually corresponds to the alignment field polarization, and hence may be identical to the standard [LF](#) definition, although is usually reserved for use in cases where there is some molecular alignment. For the limiting case of an isotropic distribution, the [AF](#) and (traditional) [LF](#) are identical. For high degrees of (3D) alignment the AF may approach the [MF](#) in the ideal case, although will usually be limited by the symmetry of the system. See [Sect. 6.3.3](#) for further details.

PADs

Photoelectron angular distributions (PADs), often with a prefix denoting the reference frame, e.g. LFPADs, MF-PADs (sometimes also hyphenated, e.g. LF-PADs). Usage is often synonymous with the associated [anisotropy paramters](#) (or “betas”).

anisotropy paramters

Expansion parameters $\beta_{L,M}$ for an expansion in spherical harmonics (or similar basis sets of angular momentum functions in polar coordinates), e.g. Eq. (6.37). Often referred to simply as “beta parameters”, and may be dependent on various properties, e.g. $\beta_{L,M}(\epsilon, t\dots)$. Herein upper-case L, M usually refer to observables or the general case, whilst lower-case (l, m) usually refer specifically to the photoelectron wavefunction partial waves (see [partial-wave expansion](#)), and (l, λ) usually denote these terms referenced specifically to the molecular frame.

ADMs

Expansion parameters $A_{Q,S}^K(t)$ for describing a molecular ensemble alignment described as a set of axis distribution moments, usually expanded as Wigner rotation matrix element, spherical harmonics or Legendre polynomial functions. See [Sect. 6.5](#) for details.

axis distribution moments

See [ADMs](#).

MS

Molecular symmetry group. Symmetry group classification of a molecule, isomorphic to the point group in rigid molecules. See Bunker and Jensen [89] for discussion.

PG

Point group. Symmetry group classification of a molecule, strictly only applicable to rigid systems. See [MS](#) for more general case.

HOMO

Highest occupied molecular orbital. Short-hand for the outermost (highest energy) valence orbital, also often used in the form HOMO-n to number lower-lying orbitals in reverse energetic order, e.g. HOMO-1 for the penultimate valence orbital.

VMI

Velocity-map imaging. Experimental technique for measuring energy and angle-resolved photoelectron “images”.

RWP

Rotational wavepacket. A purely rotational wavepacket (superposition of rotational eigenstates) in a molecular system, typically created via cascaded Raman interaction with a (relatively) strong IR pulse ($> 10^{12}\text{~Wcm}^{-2}$). The resulting time-dependent molecular axis distribution can be described by a set of [ADMs](#).

partial-wave expansion

General term for an expansion of a wavefunction in a spherical-wave basis in scattering theory, typically spherical harmonics $Y_{l,m}$, where the spherical harmonics are the partial wave basis set, and specific $\psi_{l,m}$ terms can be referred to as partial waves - see for example Refs. [145, 146, 147]. Note conventional use of lower-case l, m for these components, whilst upper-case L, M are usually used for labelling harmonics pertaining to observable quantities (see [anisotropy parameters](#)).

partial-waves

See [partial-wave expansion](#).

channel functions

Geometric (angular-momentum) coupling parameters in the tensor formulation of photoionization, denoted by $\mathcal{T}_{L,M}^{u,\zeta\zeta'}$ herein. See Eq. (6.13). These can be regarded as an alternative form of the more traditional geometric coupling parameters (Eq. (6.9)). See also [geometric coupling parameters](#).

geometric coupling parameters

Geometric (angular-momentum) coupling parameters in photoionization, comprising all angular-momentum coupling terms. Denoted $\gamma_{l,m}$ herein (Eq. (6.6)), and $\gamma_{\alpha\alpha_+ l\lambda m' \lambda' m'}$ for the coherent square of these terms (Eq. (6.9)). See also [channel functions](#).

radial matrix elements

General term for the radial part of the ionization matrix elements, after separation into radial and angular parts. Although this type of separation may be applied in many cases, herein this term always refers specifically to the radial (or reduced) [photoionization dipole matrix elements](#). These are denoted herein as $\mathbf{r}_{k,l,m}$ (see Eqs. (6.6), (6.7)), and also appear as $\mathbb{I}^{\zeta\zeta'}$ for the coherent square of these terms in the [channel functions](#) (tensor) form, see Eq. (6.13). These complex matrix elements are the unknowns to be determined in quantum metrology with photoelectrons fitting or reconstruction problems.

symmetrized harmonics

A basis set of spherical harmonics expanded/defined for a given point-group symmetry. See Sect. 6.6.2, particularly Eq. (6.37), for details. Other symmetrized functions may assume such a basis set, or explicitly incorporate symmetry parameters/weightings directly, as is the case for symmetrized [radial matrix elements](#) which incorporate symmetry parameters $b_{hl\lambda}^{\Gamma\mu}$ into the value of the matrix elements.

frame rotation

General term for the rotation of one frame of reference (corresponding to a set of x, y, z axes), e.g. as defined by an electric field vector in the laboratory, to another, e.g. as defined by a molecular axis. Usually specified herein in terms of a set of Euler angles $R_{\hat{n}} = \{\chi, \Theta, \Phi\}$, and can also be schematically denoted as, e.g., $(x, y, z) \leftarrow (x', y', z')$.

Euler angles

A set of angles $R_{\hat{n}} = \{\chi, \Theta, \Phi\}$ defining a frame rotation. For discussion see Wikipedia[148] and Zare, Chpt. 3 [83].

Wigner rotation matrix elements

A function defining a basis set for rotations in three-dimensions ($SO(3)$). Also known as “Wigner D-matrix elements”. For discussion see [Wikipedia\[149\]](#) and Zare, Chpt. 3 [83].

molecular alignment

General term for describing the case where molecules have some alignment in the LF , as compared to an isotropic distribution. Defined herein terms of molecular *axis distribution moments* and associated parameters $A_{Q,S}^K(t)$. See [Sect. 6.5](#) for details.

bootstrap retrieval protocol

General term of a retrieval method which determines successively more complex properties of a system in multiple steps, each of which builds on the previous step and adds complexity. Herein, used for the “generalised bootstrapping” method for *radial matrix elements* retrieval from photoionization data. For general usage, see [wikipedia’s Bootstrapping page](#).

Part V

Test pages

BUILD VERSIONS AND CONFIG TESTS

17.1 Versions

```
import scooby
scooby.Report(additional=['pemtk','epsproc','xarray', 'pandas', 'scipy', 'matplotlib',
                           'jupyterlab','plotly','holoviews'])
```

```
* sparse not found, sparse matrix forms not available.
* natsort not found, some sorting functions not available.
* Setting plotter defaults with epsproc.basicPlotters.setPlotters(). Run directly
  ↵to modify, or change options in local env.
* Set Holoviews with bokeh.
* pyevtk not found, VTK export not available.
```

```
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
  ↵levels instead.
```

```
-----  
Date: Mon Jul 31 16:56:07 2023 EDT
```

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

```
Python 3.10.11 | packaged by conda-forge | (main, May 10 2023, 18:58:44)
[GCC 11.3.0]
```

```
      pemtk : 0.0.1
      epsproc : 1.3.2-dev
        xarray : 2022.3.0
        pandas : 1.5.3
          scipy : 1.10.1
      matplotlib : 3.5.3
      jupyterlab : 3.6.3
        plotly : 5.15.0
      holoviews : 1.16.2
      numpy : 1.23.5
```

(continues on next page)

(continued from previous page)

```
IPython : 8.13.2
scooby  : 0.7.2
```

```
!jupyter-book --version
```

```
Jupyter Book      : 0.15.1
External ToC      : 0.3.1
MyST-Parse        : 0.18.1
MyST-NB          : 0.17.2
Sphinx Book Theme: 1.0.1
Jupyter-Cache    : 0.6.1
NbClient          : 0.7.4
```

```
!jupyter --version
```

```
Selected Jupyter core packages...
IPython           : 8.13.2
ipykernel         : 6.23.0
ipywidgets        : 8.0.6
jupyter_client    : 8.2.0
jupyter_core      : 5.3.0
jupyter_server    : 2.5.0
jupyterlab        : 3.6.3
nbclient          : 0.7.4
nbconvert         : 7.4.0
nbformat          : 5.8.0
notebook          : 6.5.4
qtconsole         : not installed
traitlets         : 5.9.0
```

17.2 Docker build env

To do

17.3 Book versions

```
QMpath = '/home/jovyan/QM3'
!git -C {QMpath} branch
!git -C {QMpath} log --format="%H" -n 1
```

```
gh-pages
main
* reviewJuly2023
79974db00e28189453428261c0a9171cb0dba419
```

```
# Check current remote commits
!git ls-remote --heads https://github.com/phockett/Quantum-Metrology-with-
˓→Photoelectrons-Vol3
```

93ab730f6f302f269809d726b50dd39b5ea52130	refs/heads/gh-pages
c0fcecc831b5683882b15ac1473fc6d732310da48	refs/heads/main
a9ebba80f671ea3784db4f5a0b28d895293ef035	refs/heads/reviewJuly2023

17.4 Github pkg versions

Note - can't get versions for local pip installs from repo(?).

```
from pathlib import Path
import epsproc as ep
ep.__file__
```

```
'/home/jovyan/github/ePSproc/epsproc/__init__.py'
```

```
# Check current Git commit for local ePSproc version - NOTE THIS ONLY WORKS FOR  
→INSTALLED FROM GIT CLONES  
# from pathlib import Path  
# import epsproc as ep  
!git -C {Path(ep.__file__).parent} branch  
!git -C {Path(ep.__file__).parent} log --format="%H" -n 1
```

* 3d-AFPAD-dev
c1eedf10631cb17fe33421427c195a2391369c37

```
# Check current remote commits  
!git ls-remote --heads https://github.com/phockett/ePSproc
```

(continues on next page)

(continued from previous page)

7e4270370d66df44c334675ac487c87d702408da	refs/heads/dev
1c0b8fd409648f07c85f4f20628b5ea7627e0c4e	refs/heads/master
69cd89ce5bc0ad6d465a4bd8df6fba15d3fd1aee	refs/heads/numba-tests
ea30878c842f09d525fbf39fa269fa2302a13b57	refs/heads/revert-9-master
baf0be0c962e8ab3c3df57c8f70f0e939f99cbd7	refs/heads/testDev

```
# Check current remote commits
!git ls-remote --heads https://github.com/phockett/PEMtk
```

465c161ca9d2c2f453a58f16c8f334f5e2c07eaa	refs/heads/master
3f4686dffdbb310f15692f978ba36d6a3d15e8d3	refs/heads/mfFittingDev

17.5 Full conda env

```
!conda list
```

# packages in environment at /opt/conda:				
#	# Name	Version	Build	Channel
	_libgcc_mutex	0.1	conda_forge	conda-forge
	_openmp_mutex	4.5	2_kmp_llvm	conda-forge
	accessible-pygments	0.0.4	pypi_0	pypi
	aiofiles	22.1.0	pyhd8ed1ab_0	conda-forge
	aiosqlite	0.19.0	pyhd8ed1ab_0	conda-forge
	alabaster	0.7.13	pypi_0	pypi
	alembic	1.10.4	pyhd8ed1ab_0	conda-forge
	altair	5.0.0	pyhd8ed1ab_0	conda-forge
	ansi2html	1.8.0	py310hff52083_1	conda-forge
	anyio	3.6.2	pyhd8ed1ab_0	conda-forge
	aom	3.5.0	h27087fc_0	conda-forge
	argon2-cffi	21.3.0	pyhd8ed1ab_0	conda-forge
	argon2-cffi-bindings	21.2.0	py310h5764c6d_3	conda-forge
	arrow	1.2.3	pypi_0	pypi
	arrow-cpp	12.0.0	ha770c72_1_cpu	conda-forge
	asteval	0.9.31	pyhd8ed1ab_0	conda-forge
	astropy	5.3.1	py310h278f3c1_0	conda-forge
	asttokens	2.2.1	pyhd8ed1ab_0	conda-forge
	async_generator	1.10	py_0	conda-forge
	attrs	23.1.0	pyh71513ae_1	conda-forge
	aws-c-auth	0.6.26	h2c7c9e7_6	conda-forge
	aws-c-cal	0.5.26	h71eb795_0	conda-forge
	aws-c-common	0.8.17	hd590300_0	conda-forge
	aws-c-compression	0.2.16	h4f47f36_6	conda-forge
	aws-c-event-stream	0.2.20	h69ce273_6	conda-forge
	aws-c-http	0.7.7	h7b8353a_3	conda-forge
	aws-c-io	0.13.21	h2c99d58_4	conda-forge
	aws-c-mqtt	0.8.6	h3a1964a_15	conda-forge
	aws-c-s3	0.2.8	h0933b68_4	conda-forge
	aws-c-sdkutils	0.1.9	h4f47f36_1	conda-forge
	aws-checksums	0.1.14	h4f47f36_6	conda-forge
	aws-crt-cpp	0.19.9	h85076f6_5	conda-forge

(continues on next page)

(continued from previous page)

aws-sdk-cpp	1.10.57	hf40e4db_10	conda-forge
babel	2.12.1	pyhd8ed1ab_1	conda-forge
backcall	0.2.0	pyh9f0ad1d_0	conda-forge
backports	1.0	pyhd8ed1ab_3	conda-forge
backports.functools_lru_cache	1.6.4	pyhd8ed1ab_0	conda-forge
beautifulsoup4	4.12.2	pyha770c72_0	conda-forge
blas	2.116	openblas	conda-forge
blas-devel	3.9.0	16_linux64_openblas	conda-forge
bleach	6.0.0	pyhd8ed1ab_0	conda-forge
blinker	1.6.2	pyhd8ed1ab_0	conda-forge
blosc	1.21.3	hafa529b_0	conda-forge
bokeh	3.1.1	pyhd8ed1ab_0	conda-forge
boltons	23.0.0	pyhd8ed1ab_0	conda-forge
boost-cpp	1.78.0	h6582d0a_3	conda-forge
bottleneck	1.3.7	py310h0a54255_0	conda-forge
brotli	1.0.9	h166bdaf_8	conda-forge
brotli-bin	1.0.9	h166bdaf_8	conda-forge
brotlipy	0.7.0	py310h5764c6d_1005	conda-forge
brunsli	0.1	h9c3ff4c_0	conda-forge
bzip2	1.0.8	h7f98852_4	conda-forge
c-ares	1.18.1	h7f98852_0	conda-forge
c-blosc2	2.8.0	hf91038e_1	conda-forge
ca-certificates	2023.7.22	hbcca054_0	conda-forge
cached-property	1.5.2	hd8ed1ab_1	conda-forge
cached_property	1.5.2	pyha770c72_1	conda-forge
cairo	1.16.0	h35add3b_1015	conda-forge
cartopy	0.21.1	py310h7eb24ba_1	conda-forge
certifi	2023.7.22	pyhd8ed1ab_0	conda-forge
certipy	0.1.3	py_0	conda-forge
cffi	1.15.1	py310h255011f_3	conda-forge
cfitsio	4.2.0	hd9d235c_0	conda-forge
cftime	1.6.2	py310hde88566_1	conda-forge
charls	2.4.1	hcb278e6_0	conda-forge
charset-normalizer	3.1.0	pyhd8ed1ab_0	conda-forge
click	8.1.3	unix_pyhd8ed1ab_2	conda-forge
cloudpickle	2.2.1	pyhd8ed1ab_0	conda-forge
colorama	0.4.6	pyhd8ed1ab_0	conda-forge
colorcet	3.0.1	pyhd8ed1ab_0	conda-forge
comm	0.1.3	pyhd8ed1ab_0	conda-forge
conda	23.3.1	py310hff52083_0	conda-forge
conda-package-handling	2.0.2	pyh38be061_0	conda-forge
conda-package-streaming	0.7.0	pyhd8ed1ab_1	conda-forge
configurable-http-proxy	4.5.4	h3b247e2_2	conda-forge
contourpy	1.0.7	py310hdf3cbec_0	conda-forge
cryptography	40.0.2	py310h34c0648_0	conda-forge
curl	8.0.1	h588be90_0	conda-forge
cycler	0.11.0	pyhd8ed1ab_0	conda-forge
cython	0.29.34	py310heca2aa9_0	conda-forge
cytoolz	0.12.0	py310h5764c6d_1	conda-forge
dash	2.11.1	pyhd8ed1ab_0	conda-forge
dask	2023.5.0	pyhd8ed1ab_0	conda-forge
dask-core	2023.5.0	pyhd8ed1ab_0	conda-forge
dav1d	1.0.0	h166bdaf_1	conda-forge
dcw-gmt	2.1.1	ha770c72_0	conda-forge
debugpy	1.6.7	py310heca2aa9_0	conda-forge
decorator	5.1.1	pyhd8ed1ab_0	conda-forge

(continues on next page)

(continued from previous page)

defusedxml	0.7.1	pyhd8ed1ab_0	conda-forge
dill	0.3.6	pyhd8ed1ab_1	conda-forge
distributed	2023.5.0	pyhd8ed1ab_0	conda-forge
docutils	0.18.1	pypi_0	pypi
ducc0	0.31.0	py310hc6cd4ac_0	conda-forge
entrypoints	0.4	pyhd8ed1ab_0	conda-forge
et_xmlfile	1.1.0	pyhd8ed1ab_0	conda-forge
exceptiongroup	1.1.2	pyhd8ed1ab_0	conda-forge
executing	1.2.0	pyhd8ed1ab_0	conda-forge
expat	2.5.0	hcb278e6_1	conda-forge
fftw	3.3.10	nompi_hc118613_108	conda-forge
firefox	115.0	hd3aeb46_0	conda-forge
flask	2.3.2	pyhd8ed1ab_0	conda-forge
flit-core	3.9.0	pyhd8ed1ab_0	conda-forge
fmt	9.1.0	h924138e_0	conda-forge
font-ttf-dejavu-sans-mono	2.37	hab24e00_0	conda-forge
font-ttf-inconsolata	3.000	h77eed37_0	conda-forge
font-ttf-source-code-pro	2.038	h77eed37_0	conda-forge
font-ttf-ubuntu	0.83	hab24e00_0	conda-forge
fontconfig	2.14.2	h14ed4e7_0	conda-forge
fonts-conda-ecosystem	1	0	conda-forge
fonts-conda-forge	1	0	conda-forge
fonttools	4.39.4	py310h2372a71_0	conda-forge
fqdn	1.5.1	pypi_0	pypi
freetype	2.12.1	hca18f0e_1	conda-forge
freexl	1.0.6	h166bdaf_1	conda-forge
fsspec	2023.5.0	pyh1a96a4e_0	conda-forge
future	0.18.3	pyhd8ed1ab_0	conda-forge
gdal	3.6.4	py310hf0ca374_2	conda-forge
geckodriver	0.33.0	hd2f7af9_0	conda-forge
geos	3.11.2	hcb278e6_0	conda-forge
geotiff	1.7.1	h480ec47_8	conda-forge
gettext	0.21.1	h27087fc_0	conda-forge
gflags	2.2.2	he1b5a44_1004	conda-forge
ghostscript	9.54.0	h27087fc_2	conda-forge
ghp-import	2.1.0	pypi_0	pypi
giflib	5.2.1	h0b41bf4_3	conda-forge
gitdb	4.0.10	pyhd8ed1ab_0	conda-forge
gitpython	3.1.31	pyhd8ed1ab_0	conda-forge
glog	0.6.0	h6f12383_0	conda-forge
gmp	6.2.1	h58526e2_0	conda-forge
gmpy2	2.1.2	py310h3ec546c_1	conda-forge
gmt	6.4.0	h4733502_10	conda-forge
greenlet	2.0.2	py310hc6cd4ac_1	conda-forge
gshhg-gmt	2.3.7	ha770c72_1003	conda-forge
h11	0.14.0	pyhd8ed1ab_0	conda-forge
h5netcdf	1.2.0	pyhd8ed1ab_0	conda-forge
h5py	3.8.0	nompi_py310ha66b2ad_101	conda-forge
hdf4	4.2.15	h501b40f_6	conda-forge
hdf5	1.14.0	nompi_hb72d44e_103	conda-forge
holoviews	1.16.2	pyhd8ed1ab_0	conda-forge
hvplot	0.8.4	py_0	pyviz
icu	72.1	hcb278e6_0	conda-forge
idna	3.4	pyhd8ed1ab_0	conda-forge
imagecodecs	2023.1.23	py310h241fb82_2	conda-forge
imageio	2.28.1	pyh24c5eb1_0	conda-forge

(continues on next page)

(continued from previous page)

imagesize	1.4.1	pypi_0	pypi
importlib-metadata	6.6.0	pyha770c72_0	conda-forge
importlib_metadata	6.6.0	hd8ed1ab_0	conda-forge
importlib_resources	5.12.0	pyhd8ed1ab_0	conda-forge
ipykernel	6.23.0	pyh210e3f2_0	conda-forge
ipymp	0.9.3	pyhd8ed1ab_0	conda-forge
ipython	8.13.2	pyh41d4057_0	conda-forge
ipython_genutils	0.2.0	py_1	conda-forge
ipywidgets	8.0.6	pyhd8ed1ab_0	conda-forge
isoduration	20.11.0	pypi_0	pypi
itsdangerous	2.1.2	pyhd8ed1ab_0	conda-forge
jedi	0.18.2	pyhd8ed1ab_0	conda-forge
jinja2	3.1.2	pyhd8ed1ab_1	conda-forge
joblib	1.2.0	pyhd8ed1ab_0	conda-forge
json-c	0.16	hc379101_0	conda-forge
json5	0.9.5	pyh9f0ad1d_0	conda-forge
jsonpatch	1.32	pyhd8ed1ab_0	conda-forge
jsonpointer	2.0	py_0	conda-forge
jsonschema	4.17.3	pyhd8ed1ab_0	conda-forge
jupyter-book	0.15.1	pypi_0	pypi
jupyter-cache	0.6.1	pypi_0	pypi
jupyter-dash	0.4.2	pyhd8ed1ab_1	conda-forge
jupyter-server-mathjax	0.2.6	pyh5bfe37b_1	conda-forge
jupyter_client	8.2.0	pyhd8ed1ab_0	conda-forge
jupyter_core	5.3.0	py310hff52083_0	conda-forge
jupyter_events	0.6.3	pyhd8ed1ab_0	conda-forge
jupyter_server	2.5.0	pyhd8ed1ab_0	conda-forge
jupyter_server_fileid	0.9.0	pyhd8ed1ab_0	conda-forge
jupyter_server_terminals	0.4.4	pyhd8ed1ab_1	conda-forge
jupyter_server_ydoc	0.8.0	pyhd8ed1ab_0	conda-forge
jupyter_telemetry	0.1.0	pyhd8ed1ab_1	conda-forge
jupyter_ydoc	0.2.3	pyhd8ed1ab_0	conda-forge
jupyterhub	4.0.0	pyh2a2186d_0	conda-forge
jupyterhub-base	4.0.0	pyh2a2186d_0	conda-forge
jupyterlab	3.6.3	pyhd8ed1ab_0	conda-forge
jupyterlab-git	0.41.0	pyhd8ed1ab_1	conda-forge
jupyterlab-spellchecker	0.8.3	pypi_0	pypi
jupyterlab_pygments	0.2.2	pyhd8ed1ab_0	conda-forge
jupyterlab_server	2.22.1	pyhd8ed1ab_0	conda-forge
jupyterlab_widgets	3.0.7	pyhd8ed1ab_0	conda-forge
jupytext	1.14.7	pyh5da7574_0	conda-forge
jxrlib	1.1	h7f98852_2	conda-forge
kaleido-core	0.2.1	h3644ca4_0	conda-forge
kealib	1.5.1	h3845be2_3	conda-forge
keyutils	1.6.1	h166bdaf_0	conda-forge
kiwisolver	1.4.4	py310hbf28c38_1	conda-forge
krb5	1.20.1	h81ceb04_0	conda-forge
latexcodec	2.0.1	pypi_0	pypi
lazy_loader	0.2	pyhd8ed1ab_0	conda-forge
lcms2	2.15	haa2dc70_1	conda-forge
ld_impl_linux-64	2.40	h41732ed_0	conda-forge
lerc	4.0.0	h27087fc_0	conda-forge
libabseil	20230125.0	cxx17_hcb278e6_1	conda-forge
libaec	1.0.6	hcb278e6_1	conda-forge
libarchive	3.6.2	h3d51595_0	conda-forge
libarrow	12.0.0	h1cdf7b0_1_cpu	conda-forge

(continues on next page)

(continued from previous page)

libavif	0.11.1	h5cddd6b5_0	conda-forge
libblas	3.9.0	16_linux64_openblas	conda-forge
libbrotlicommon	1.0.9	h166bdaf_8	conda-forge
libbrotlidec	1.0.9	h166bdaf_8	conda-forge
libbrotlienc	1.0.9	h166bdaf_8	conda-forge
libcblas	3.9.0	16_linux64_openblas	conda-forge
libcrc32c	1.1.2	h9c3ff4c_0	conda-forge
libcurl	8.0.1	h588be90_0	conda-forge
libdeflate	1.18	h0b41bf4_0	conda-forge
libedit	3.1.20191231	he28a2e2_2	conda-forge
libev	4.33	h516909a_1	conda-forge
libevent	2.1.12	h3358134_0	conda-forge
libexpat	2.5.0	hcb278e6_1	conda-forge
libffi	3.4.2	h7f98852_5	conda-forge
libgcc-ng	12.2.0	h65d4601_19	conda-forge
libgdal	3.6.4	hada8d5e_2	conda-forge
libgfortran-ng	12.2.0	h69a702a_19	conda-forge
libgfortran5	12.2.0	h337968e_19	conda-forge
libglib	2.76.4	hebfc3b9_0	conda-forge
libgomp	12.2.0	h65d4601_19	conda-forge
libgoogle-cloud	2.10.0	hac9eb74_0	conda-forge
libgrpc	1.54.2	hcf146ea_0	conda-forge
libiconv	1.17	h166bdaf_0	conda-forge
libjpeg-turbo	2.1.5.1	h0b41bf4_0	conda-forge
libkml	1.3.0	h37653c0_1015	conda-forge
liblapack	3.9.0	16_linux64_openblas	conda-forge
liblapacke	3.9.0	16_linux64_openblas	conda-forge
libllvml11	11.1.0	he0ac6c6_5	conda-forge
libmamba	1.4.2	hcea66bb_0	conda-forge
libmambapy	1.4.2	py310h1428755_0	conda-forge
libnetcdf	4.9.2	nompi_hdf9a29f_104	conda-forge
libnghttp2	1.52.0	h61bc06f_0	conda-forge
libnsl	2.0.0	h7f98852_0	conda-forge
libnuma	2.0.16	h0b41bf4_1	conda-forge
libopenblas	0.3.21	pthreads_h78a6416_3	conda-forge
libpng	1.6.39	h753d276_0	conda-forge
libpq	15.3	hbcd7760_0	conda-forge
libprotobuf	3.21.12	h3eb15da_0	conda-forge
librttopo	1.1.0	h0d5128d_13	conda-forge
libsodium	1.0.18	h36c2ea0_1	conda-forge
libsolv	0.7.23	h3eb15da_0	conda-forge
libspatialite	5.0.1	h7d1ca68_25	conda-forge
libsqlite	3.41.2	h2797004_1	conda-forge
libssh2	1.10.0	hf14f497_3	conda-forge
libstdc++-ng	12.2.0	h46fd767_19	conda-forge
libthrift	0.18.1	h8fd135c_1	conda-forge
libtiff	4.5.0	ha587672_6	conda-forge
libutf8proc	2.8.0	h166bdaf_0	conda-forge
libuuid	2.38.1	h0b41bf4_0	conda-forge
libuv	1.44.2	h166bdaf_0	conda-forge
libwebp-base	1.3.0	h0b41bf4_0	conda-forge
libxcb	1.13	h7f98852_1004	conda-forge
libxml2	2.10.4	hfdac1af_0	conda-forge
libzip	1.9.2	hc929e4a_1	conda-forge
libzlib	1.2.13	h166bdaf_4	conda-forge
libzopfli	1.0.3	h9c3ff4c_0	conda-forge

(continues on next page)

(continued from previous page)

linkify-it-py	2.0.0	pyhd8ed1ab_0	conda-forge
llvm-openmp	16.0.3	h4dfa4b3_0	conda-forge
llvmlite	0.39.1	py310h58363a5_1	conda-forge
lmfit	1.2.2	pyhd8ed1ab_0	conda-forge
locket	1.0.0	pyhd8ed1ab_0	conda-forge
lz4	4.3.2	py310h0cfdcf0_0	conda-forge
lz4-c	1.9.4	hcb278e6_0	conda-forge
lzo	2.10	h516909a_1000	conda-forge
mako	1.2.4	pyhd8ed1ab_0	conda-forge
mamba	1.4.2	py310h51d5547_0	conda-forge
markdown	3.4.3	pyhd8ed1ab_0	conda-forge
markdown-it-py	2.2.0	pypi_0	pypi
markupsafe	2.1.2	py310h1fa729e_0	conda-forge
mathjax	2.7.7	ha770c72_3	conda-forge
matplotlib	3.5.3	pypi_0	pypi
matplotlib-inline	0.1.6	pyhd8ed1ab_0	conda-forge
mdit-py-plugins	0.3.5	pypi_0	pypi
mdurl	0.1.0	pyhd8ed1ab_0	conda-forge
mistune	2.0.5	pyhd8ed1ab_0	conda-forge
mpc	1.3.1	hfe3b2da_0	conda-forge
mpfr	4.2.0	hb012696_0	conda-forge
mpmath	1.3.0	pyhd8ed1ab_0	conda-forge
msgpack-python	1.0.5	py310hdf3cbec_0	conda-forge
munkres	1.1.4	pyh9f0ad1d_0	conda-forge
myst-nb	0.17.2	pypi_0	pypi
myst-parser	0.18.1	pypi_0	pypi
nbclassic	1.0.0	pyhb4ecaf3_1	conda-forge
nbclient	0.7.4	pyhd8ed1ab_0	conda-forge
nbconvert	7.4.0	pyhd8ed1ab_0	conda-forge
nbconvert-core	7.4.0	pyhd8ed1ab_0	conda-forge
nbconvert-pandoc	7.4.0	pyhd8ed1ab_0	conda-forge
nbdime	3.2.1	pyhd8ed1ab_0	conda-forge
nbformat	5.8.0	pyhd8ed1ab_0	conda-forge
ncurses	6.3	h27087fc_1	conda-forge
nest-asyncio	1.5.6	pyhd8ed1ab_0	conda-forge
netcdf4	1.6.4	nompi_py310hde23a83_100	conda-forge
networkx	3.1	pyhd8ed1ab_0	conda-forge
nodejs	16.19.0	h4abf6b9_1	conda-forge
nomkl	1.0	h5ca1d4c_0	conda-forge
notebook	6.5.4	pyha770c72_0	conda-forge
notebook-shim	0.2.3	pyhd8ed1ab_0	conda-forge
nspr	4.35	h27087fc_0	conda-forge
nss	3.89	he45b914_0	conda-forge
numba	0.56.4	py310h0e39c9b_1	conda-forge
numexpr	2.8.4	py310h690d005_100	conda-forge
numpy	1.23.5	py310h53a5b5f_0	conda-forge
oauthlib	3.2.2	pyhd8ed1ab_0	conda-forge
openblas	0.3.21	pthreads_h320a7e8_3	conda-forge
openjpeg	2.5.0	hfec8fc6_2	conda-forge
openpyxl	3.1.2	py310h2372a71_0	conda-forge
openssl	3.1.1	hd590300_1	conda-forge
orc	1.8.3	hfdbbad2_0	conda-forge
outcome	1.2.0	pyhd8ed1ab_0	conda-forge
packaging	23.1	pyhd8ed1ab_0	conda-forge
pamela	1.0.0	py_0	conda-forge
pandas	1.5.3	py310h9b08913_1	conda-forge

(continues on next page)

(continued from previous page)

pandoc	2.19.2	h32600fe_2	conda-forge
pandocfilters	1.5.0	pyhd8ed1ab_0	conda-forge
panel	1.2.0	pyhd8ed1ab_0	conda-forge
param	1.13.0	pyh1a96a4e_0	conda-forge
parquet-cpp	1.5.1	2	conda-forge
parso	0.8.3	pyhd8ed1ab_0	conda-forge
partd	1.4.0	pyhd8ed1ab_0	conda-forge
patsy	0.5.3	pyhd8ed1ab_0	conda-forge
pcre	8.45	h9c3fff4c_0	conda-forge
pcre2	10.40	hc3806b6_0	conda-forge
pexpect	4.8.0	pyh1a96a4e_2	conda-forge
pickleshare	0.7.5	py_1003	conda-forge
pillow	9.5.0	py310h065c6d2_0	conda-forge
pip	23.1.2	pyhd8ed1ab_0	conda-forge
pixman	0.40.0	h36c2ea0_0	conda-forge
pkgutil-resolve-name	1.3.10	pyhd8ed1ab_0	conda-forge
platformdirs	3.5.1	pyhd8ed1ab_0	conda-forge
plotly	5.15.0	pyhd8ed1ab_0	conda-forge
pluggy	1.0.0	pyhd8ed1ab_5	conda-forge
pooch	1.7.0	pyha770c72_3	conda-forge
poppler	23.05.0	hd18248d_1	conda-forge
poppler-data	0.4.12	hd8ed1ab_0	conda-forge
postgresql	15.3	h814edd5_0	conda-forge
proj	9.2.0	h8ffa02c_0	conda-forge
prometheus_client	0.16.0	pyhd8ed1ab_0	conda-forge
prompt-toolkit	3.0.38	pyha770c72_0	conda-forge
prompt_toolkit	3.0.38	hd8ed1ab_0	conda-forge
protobuf	4.21.12	py310heca2aa9_0	conda-forge
psutil	5.9.5	py310h1fa729e_0	conda-forge
pthread-stubs	0.4	h36c2ea0_1001	conda-forge
ptyprocess	0.7.0	pyhd3deb0d_0	conda-forge
pure_eval	0.2.2	pyhd8ed1ab_0	conda-forge
py-cpuinfo	9.0.0	pyhd8ed1ab_0	conda-forge
pyarrow	12.0.0	py310he6bfd7f_1_cpu	conda-forge
pybind11-abi	4	hd8ed1ab_3	conda-forge
pybtex	0.24.0	pypi_0	pypi
pybtex-docutils	1.0.2	pypi_0	pypi
pycosat	0.6.4	py310h5764c6d_1	conda-forge
pycparser	2.21	pyhd8ed1ab_0	conda-forge
pyct	0.4.6	py_0	conda-forge
pyct-core	0.4.6	py_0	conda-forge
pycurl	7.45.1	py310h60f9ec7_3	conda-forge
pydata-sphinx-theme	0.13.3	pypi_0	pypi
pyerfa	2.0.0.3	py310h0a54255_0	conda-forge
pygments	2.15.1	pyhd8ed1ab_0	conda-forge
pygmt	0.9.0	pyhd8ed1ab_0	conda-forge
pyjwt	2.7.0	pyhd8ed1ab_0	conda-forge
pyopenssl	23.1.1	pyhd8ed1ab_0	conda-forge
pyparsing	3.0.9	pyhd8ed1ab_0	conda-forge
pyproj	3.6.0	py310ha254fea_0	conda-forge
pyrsistent	0.19.3	py310h1fa729e_0	conda-forge
pyshp	2.3.1	pyhd8ed1ab_0	conda-forge
pyshtools	4.10.3	py310h3e61171_0	conda-forge
pysocks	1.7.1	pyha2e5f31_6	conda-forge
pytables	3.8.0	py310hde6a235_1	conda-forge
python	3.10.11	he550d4f_0_cpython	conda-forge

(continues on next page)

(continued from previous page)

python-dateutil	2.8.2	pyhd8ed1ab_0	conda-forge
python-fastjsonschema	2.16.3	pyhd8ed1ab_0	conda-forge
python-json-logger	2.0.7	pyhd8ed1ab_0	conda-forge
python-kaleido	0.2.1	pyhd8ed1ab_0	conda-forge
python-tzdata	2023.3	pyhd8ed1ab_0	conda-forge
python_abi	3.10	3_cp310	conda-forge
pytz	2023.3	pyhd8ed1ab_0	conda-forge
pyviz_comms	2.3.2	pyhd8ed1ab_0	conda-forge
pywavelets	1.4.1	py310h0a54255_0	conda-forge
pyyaml	6.0	py310h5764c6d_5	conda-forge
pyzmq	25.0.2	py310h059b190_0	conda-forge
quaternion	2022.4.3	py310h0a54255_0	conda-forge
qutip	4.7.2	py310hfb6f7a9_1	conda-forge
re2	2023.02.02	hcb278e6_0	conda-forge
readline	8.2	h8228510_1	conda-forge
reproc	14.2.4	h0b41bf4_0	conda-forge
reproc-cpp	14.2.4	hcb278e6_0	conda-forge
requests	2.29.0	pyhd8ed1ab_0	conda-forge
retrying	1.3.3	py_2	conda-forge
rfc3339-validator	0.1.4	pyhd8ed1ab_0	conda-forge
rfc3986-validator	0.1.1	pyh9f0ad1d_0	conda-forge
ruamel.yaml	0.17.26	py310h2372a71_0	conda-forge
ruamel.yaml.clib	0.2.7	py310h1fa729e_1	conda-forge
s2n	1.3.44	h06160fa_0	conda-forge
scikit-image	0.20.0	py310h9b08913_1	conda-forge
scikit-learn	1.2.2	py310h41b6a48_1	conda-forge
scipy	1.10.1	py310ha4c1d20_3	conda-forge
scooby	0.7.2	pyhd8ed1ab_0	conda-forge
seaborn	0.9.0	py_2	conda-forge
seaborn-base	0.12.2	pyhd8ed1ab_0	conda-forge
selenium	4.10.0	pyhd8ed1ab_0	conda-forge
send2trash	1.8.2	pyh41d4057_0	conda-forge
setuptools	67.7.2	pyhd8ed1ab_0	conda-forge
setuptools-scm	7.1.0	pyhd8ed1ab_0	conda-forge
shapely	2.0.1	py310h056c13c_1	conda-forge
six	1.16.0	pyh6c4a22f_0	conda-forge
smmmap	3.0.5	pyh44b312d_0	conda-forge
snappy	1.1.10	h9fff704_0	conda-forge
sniffio	1.3.0	pyhd8ed1ab_0	conda-forge
snowballstemmer	2.2.0	pypi_0	pypi
sortedcontainers	2.4.0	pyhd8ed1ab_0	conda-forge
soupsieve	2.3.2.post1	pyhd8ed1ab_0	conda-forge
spherical_functions	2022.4.2	pyhd8ed1ab_0	conda-forge
sphinx	5.0.2	pypi_0	pypi
sphinx-book-theme	1.0.1	pypi_0	pypi
sphinx-comments	0.0.3	pypi_0	pypi
sphinx-copybutton	0.5.2	pypi_0	pypi
sphinx-design	0.3.0	pypi_0	pypi
sphinx-external-toc	0.3.1	pypi_0	pypi
sphinx-jupyterbook-latex	0.5.2	pypi_0	pypi
sphinx-multitoc-numbering	0.1.3	pypi_0	pypi
sphinx-thebe	0.2.1	pypi_0	pypi
sphinx-togglebutton	0.3.2	pypi_0	pypi
sphinxcontrib-applehelp	1.0.4	pypi_0	pypi
sphinxcontrib-bibtex	2.5.0	pypi_0	pypi
sphinxcontrib-devhelp	1.0.2	pypi_0	pypi

(continues on next page)

(continued from previous page)

sphinxcontrib-htmlhelp	2.0.1	pypi_0	pypi
sphinxcontrib-jsmath	1.0.1	pypi_0	pypi
sphinxcontrib-qthelp	1.0.3	pypi_0	pypi
sphinxcontrib-serializinghtml	1.1.5	pypi_0	pypi
spinsfast	2022.4.2	py310hc9031d1_0	conda-forge
sqlalchemy	2.0.13	py310h2372a71_0	conda-forge
sqlite	3.41.2	h2c6b66d_1	conda-forge
stack_data	0.6.2	pyhd8ed1ab_0	conda-forge
statsmodels	0.14.0	py310h278f3c1_1	conda-forge
sympy	1.11.1	pypyh9d50eac_103	conda-forge
tabulate	0.9.0	pypi_0	pypi
tblib	1.7.0	pyhd8ed1ab_0	conda-forge
tenacity	8.2.2	pyhd8ed1ab_0	conda-forge
terminado	0.17.1	pyh41d4057_0	conda-forge
threadpoolctl	3.1.0	pyh8a188c0_0	conda-forge
tifffile	2023.4.12	pyhd8ed1ab_0	conda-forge
tiledb	2.13.2	hd532e3d_0	conda-forge
tinyCSS2	1.2.1	pyhd8ed1ab_0	conda-forge
tk	8.6.12	h27826a3_0	conda-forge
toml	0.10.2	pyhd8ed1ab_0	conda-forge
tomli	2.0.1	pyhd8ed1ab_0	conda-forge
toolz	0.12.0	pyhd8ed1ab_0	conda-forge
tornado	6.3	py310h1fa729e_0	conda-forge
tqdm	4.65.0	pyhd8ed1ab_1	conda-forge
traitlets	5.9.0	pyhd8ed1ab_0	conda-forge
trio	0.21.0	py310hff52083_0	conda-forge
trio-websocket	0.10.3	pyhd8ed1ab_0	conda-forge
typing-extensions	4.5.0	hd8ed1ab_0	conda-forge
typing_extensions	4.5.0	pyha770c72_0	conda-forge
tzcode	2023c	h0b41bf4_0	conda-forge
tzdata	2023c	h71feb2d_0	conda-forge
uc-micro-py	1.0.1	pyhd8ed1ab_0	conda-forge
ucx	1.14.0	h3484d09_2	conda-forge
uncertainties	3.1.7	pyhd8ed1ab_0	conda-forge
unicodedata2	15.0.0	py310h5764c6d_0	conda-forge
uri-template	1.3.0	pypi_0	pypi
urllib3	1.26.15	pyhd8ed1ab_0	conda-forge
wcwidth	0.2.6	pyhd8ed1ab_0	conda-forge
webcolors	1.13	pypi_0	pypi
webencodings	0.5.1	pyhd8ed1ab_0	conda-forge
websocket-client	1.5.1	pyhd8ed1ab_0	conda-forge
werkzeug	2.3.6	pyhd8ed1ab_0	conda-forge
wget	3.2	pypi_0	pypi
wheel	0.40.0	pyhd8ed1ab_0	conda-forge
widgetsnbextension	4.0.7	pyhd8ed1ab_0	conda-forge
wsproto	1.2.0	pyhd8ed1ab_0	conda-forge
xarray	2022.3.0	pyhd8ed1ab_0	conda-forge
xerces-c	3.2.4	h8d71039_2	conda-forge
xlrd	2.0.1	pyhd8ed1ab_3	conda-forge
xorg-kbproto	1.0.7	h7f98852_1002	conda-forge
xorg-libice	1.1.1	hd590300_0	conda-forge
xorg-libsrm	1.2.4	h7391055_0	conda-forge
xorg-libx11	1.8.4	h0b41bf4_0	conda-forge
xorg-libxau	1.0.9	h7f98852_0	conda-forge
xorg-libxdmcp	1.1.3	h7f98852_0	conda-forge
xorg-libxext	1.3.4	h0b41bf4_2	conda-forge

(continues on next page)

(continued from previous page)

xorg-libxrender	0.9.10	h7f98852_1003	conda-forge
xorg-renderproto	0.11.1	h7f98852_1002	conda-forge
xorg-xextproto	7.3.0	h0b41bf4_1003	conda-forge
xorg-xproto	7.0.31	h7f98852_1007	conda-forge
xyzpy	1.2.1	pyhd8ed1ab_0	conda-forge
xyzservices	2023.2.0	pyhd8ed1ab_0	conda-forge
xz	5.2.6	h166bdaf_0	conda-forge
y-py	0.5.9	py310h4426083_0	conda-forge
yaml	0.2.5	h7f98852_2	conda-forge
yaml-cpp	0.7.0	h27087fc_2	conda-forge
ypy-websocket	0.8.2	pyhd8ed1ab_0	conda-forge
zeromq	4.3.4	h9c3ff4c_1	conda-forge
zfp	1.0.0	h27087fc_3	conda-forge
zict	3.0.0	pyhd8ed1ab_0	conda-forge
zipp	3.15.0	pyhd8ed1ab_0	conda-forge
zlib	1.2.13	h166bdaf_4	conda-forge
zlib-ng	2.0.7	h0b41bf4_0	conda-forge
zstandard	0.19.0	py310hdeb6495_1	conda-forge
zstd	1.5.2	h3eb15da_6	conda-forge

BIBLIOGRAPHY

- [1] Claude Marceau, Varun Makhija, Dominique Platzer, A. \relax Yu. Naumov, P. B. Corkum, Albert Stolow, D. M. Villeneuve, and Paul Hockett. Molecular Frame Reconstruction Using Time-Domain Photoionization Interferometry. *Physical Review Letters*, 119(8):083401, August 2017. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.119.083401>.
- [2] Paul Hockett and Varun Makhija. Topical Review: Extracting Molecular Frame Photoionization Dynamics from Experimental Data (Preprint). September 2022. URL: <https://www.authorea.com/users/71114/articles/447808-extracting-molecular-frame-photoionization-dynamics-from-experimental-data> (visited on 2022-09-15), arXiv:2209.04301.
- [3] Paul Hockett and Varun Makhija. Topical Review: Extracting molecular frame photoionization dynamics from experimental data. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 56(11):112001, May 2023. URL: <https://dx.doi.org/10.1088/1361-6455/acd03e> (visited on 2023-05-12), arXiv:2209.04301.
- [4] Paul Hockett. *Quantum Metrology with Photoelectrons, Volume 1: Foundations*. IOP Publishing, 2018. ISBN 978-1-68174-684-5. URL: <http://iopscience.iop.org/book/978-1-6817-4684-5>.
- [5] Paul Hockett. Photoelectron Metrology Toolkit (PEMtk) Github repository. 2021. URL: <https://github.com/phockett/PEMtk> (visited on 2022-02-18).
- [6] Paul Hockett. Phase-sensitive Photoelectron Metrology (DAMOP 2017). June 2017. URL: <https://vimeo.com/223603377> (visited on 2023-07-06).
- [7] Paul Hockett. Bootstrapping (Ultrafast) Photoionization Dynamics. January 2018. URL: <https://vimeo.com/252040672> (visited on 2023-07-06).
- [8] Paul Hockett. Presentations archive: ultrafast light-matter interactions. *Figshare*, 2016. URL: <https://figshare.com/collections/Presentations/3312291> (visited on 2022-02-16).
- [9] Paul Hockett. *Quantum Metrology with Photoelectrons, Volume 2: Applications and Advances*. IOP Publishing, 2018. ISBN 978-1-68174-688-3. URL: <http://iopscience.iop.org/book/978-1-6817-4688-3>.
- [10] Varun Makhija, Xiaoming Ren, Drue Gockel, Ahn-Thu Le, and Vinod Kumarappan. Orientation Resolution through Rotational Coherence Spectroscopy. *arXiv*, pages 1–6, 2016. URL: <http://arxiv.org/abs/1611.06476>, arXiv:1611.06476.
- [11] Varun Makhija, Kevin Veyrinas, Andrey E. Boguslavskiy, Ruaridh Forbes, Iain Wilkinson, Rune Lausten, Simon P. Neville, Stephen T. Pratt, Michael S. Schuurman, and Albert Stolow. Ultrafast molecular frame electronic coherences from lab frame scattering anisotropies. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 53(11):114001, May 2020. URL: <https://dx.doi.org/10.1088/1361-6455/ab7a84> (visited on 2023-06-14).
- [12] Luna Morrigan, Simon P. Neville, Margaret Gregory, Andrey E. Boguslavskiy, Ruaridh Forbes, Iain Wilkinson, Rune Lausten, Albert Stolow, Michael S. Schuurman, Paul Hockett, and Varun Makhija. Ultrafast Molecular Frame Quantum Tomography. March 2023. URL: <http://arxiv.org/abs/2303.03558> (visited on 2023-03-08), arXiv:2303.03558.

- [13] Project Jupyter. URL: <https://jupyter.org> (visited on 2023-01-16).
- [14] Python.org. July 2023. URL: <https://www.python.org/> (visited on 2023-07-07).
- [15] Jupyter Book Project. URL: <https://jupyterbook.org>.
- [16] Executable Books Community. Jupyter Book. Zenodo, February 2020. URL: <https://zenodo.org/record/4539666> (visited on 2023-01-16).
- [17] MyST Markdown - Tools for the future of technical communication. URL: <https://mystmd.org/> (visited on 2023-07-07).
- [18] Sphinx documentation. URL: <https://www.sphinx-doc.org> (visited on 2023-07-07).
- [19] Paul Hockett. Open Photoionization Docker Stacks. URL: <https://github.com/phockett/open-photoionization-docker-stacks> (visited on 2022-08-04).
- [20] Paul Hockett. PEMtk - the Photoelectron Metrology Toolkit - documentation. 2021. URL: <https://pemtk.readthedocs.io> (visited on 2022-02-18).
- [21] Jake VanderPlas. *Python Data Science Handbook*. O'Reilly Media, Inc., 2016. ISBN 978-1-4919-1205-8. URL: <https://www.oreilly.com/library/view/python-data-science/9781491912126/> (visited on 2023-07-07).
- [22] Jake VanderPlas. Python Data Science Handbook. July 2023. URL: <https://github.com/jakevdp/PythonDataScienceHandbook> (visited on 2023-07-07).
- [23] Nick Barnes. Publish your computer code: it is good enough. *Nature*, 467(7317):753, October 2010. URL: <http://www.nature.com/news/2010/101013/full/467753a.html> (visited on 2016-04-18).
- [24] Marcia McNutt. Taking up TOP. *Science*, 2016. URL: <http://science.scienmag.org/content/352/6290/1147.full> (visited on 2017-04-04).
- [25] B. A. Nosek, G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, C. D. Chambers, G. Chin, G. Christensen, M. Contestabile, A. Dafoe, E. Eich, J. Freese, R. Glennerster, D. Goroff, D. P. Green, B. Hesse, M. Humphreys, J. Ishiyama, D. Karlan, A. Kraut, A. Lupia, P. Mabry, T. Madon, N. Malhotra, E. Mayo-Wilson, M. McNutt, E. Miguel, E. Levy Paluck, U. Simonsohn, C. Soderberg, B. A. Spellman, J. Turitto, G. VandenBos, S. Vazire, E. J. Wagenmakers, R. Wilson, and T. Yarkoni. Promoting an open research culture. *Science*, 348(6242):1422–1425, June 2015. URL: <http://science.scienmag.org.proxy.bib.uottawa.ca/content/348/6242/1422.full> (visited on 2017-04-04).
- [26] Jon Treadway, Mark Hahnel, Sabina Leonelli, Dan Penny, David Groenewegen, Nobuko Miyairi, Kazuhiro Hayashi, Daniel O'Donnell, Digital Science, and Daniel Hook. The State of Open Data Report. Report, Digital Science, October 2016. URL: https://figshare.com/articles/report/The_State_of_Open_Data_Report/4036398/1 (visited on 2023-07-31).
- [27] Victoria Stodden and Sheila Miguez. Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research. *Journal of Open Research Software*, 2(1):e21, July 2014. URL: <http://openresearchsoftware.metajnl.com/articles/10.5334/jors.ay/> (visited on 2016-03-17).
- [28] Robert R. Downs, W. Christopher Lenhardt, Erin Robinson, Ethan Davis, and Nicholas Weber. Community Recommendations for Sustainable Scientific Software. *Journal of Open Research Software*, November 2015. URL: <http://openresearchsoftware.metajnl.com/articles/10.5334/jors.bt/> (visited on 2016-06-01).
- [29] James Howison, Ewa Deelman, Michael J. McLennan, Rafael Ferreira da Silva, and James D. Herbsleb. Understanding the scientific software ecosystem and its impact: Current and future measures. *Research Evaluation*, 24(4):454–470, October 2015. URL: <https://doi.org/10.1093/reseval/rvv014> (visited on 2022-03-12).
- [30] "Open science". Open science. *Wikipedia*, May 2023. URL: https://en.wikipedia.org/w/index.php?title=Open_science&oldid=1155039330 (visited on 2023-07-07).
- [31] Thomas Kluyver, Benjamin Ragan-Kelley, P. Fernando Rez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Dami Avila, n, Safia Abdalla, Carol Willing, and Jupyter Development Team. Jupyter Notebooks – a publishing format for reproducible

- computational workflows. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90, 2016. URL: <https://ebooks.iospress.nl/doi/10.3233/978-1-61499-649-1-87> (visited on 2023-01-16).
- [32] Brian E. Granger and Fernando Pérez. Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering*, 23(2):7–14, March 2021.
 - [33] Paul Hockett. ePSproc: Post-processing suite for ePolyScat electron-molecule scattering calculations. *Authorea*, 2016. URL: https://www.authorea.com/users/71114/articles/122402/_show_article.
 - [34] Paul Hockett. ePSproc: Post-processing for ePolyScat (Github repository). Github, 2016. URL: <https://github.com/phockett/ePSproc>.
 - [35] Paul Hockett. ePSproc: Post-processing for ePolyScat documentation. 2020. URL: <https://epsproc.readthedocs.io> (visited on 2022-02-18).
 - [36] Robert R. Lucchese, Kazuo Takatsuka, and Vincent McKoy. Applications of the Schwinger variational principle to electron-molecule collisions and molecular photoionization. *Physics Reports*, 131(3):147–221, January 1986. URL: <http://www.sciencedirect.com/science/article/pii/037015738690147X> (visited on 2012-07-18).
 - [37] F. A. Gianturco, R. R. Lucchese, and N. Sanna. Calculation of low-energy elastic cross sections for electron-CF₄ scattering. *The Journal of Chemical Physics*, 100(9):6464, May 1994. URL: <http://scitation.aip.org/content/aip/journal/jcp/100/9/10.1063/1.467237> (visited on 2015-08-13).
 - [38] Alexandra P P Natalense and Robert R Lucchese. Cross section and asymmetry parameter calculation for sulfur 1s photoionization of SF₆. *The Journal of Chemical Physics*, 111(12):5344, 1999. URL: <http://link.aip.org/link/JCPA6/v111/i12/p5344/s1&Agg=doi> (visited on 2012-07-18).
 - [39] R R Lucchese. ePolyScat User's Manual. URL: <https://epolyscat.droppages.com/> (visited on 2022-04-26).
 - [40] Andrew C. Brown, Gregory S. J. Armstrong, Jakub Benda, Daniel D. A. Clarke, Jack Wragg, Kathryn R. Hamilton, Zdeněk Mašín, Jimena D. Gorfinkel, and Hugo W. van der Hart. RMT: R-matrix with time-dependence. Solving the semi-relativistic, time-dependent Schrödinger equation for general, multielectron atoms and molecules in intense, ultrashort, arbitrarily polarized laser pulses. *Computer Physics Communications*, 250:107062, May 2020. URL: <https://www.sciencedirect.com/science/article/pii/S0010465519303856> (visited on 2022-11-09), arXiv:1905.06156.
 - [41] Andrew C. Brown, Gregory S. J. Armstrong, Jakub Benda, Daniel D. A. Clarke, Jack Wragg, Kathryn R. Hamilton, Zdeněk Mašín, Jimena D. Gorfinkel, and Hugo W. van der Hart. RMT: R-matrix with time-dependence (repository). May 2020. URL: <https://gitlab.com/Uk-amor/RMT/rmt> (visited on 2022-11-09), arXiv:1905.06156.
 - [42] Danielle Dowek and Piero Decleva. Trends in angle-resolved molecular photoelectron spectroscopy. *Physical Chemistry Chemical Physics*, 2022. URL: <https://pubs.rsc.org/en/content/articlelanding/2022/cp/d2cp02725a> (visited on 2022-10-13).
 - [43] Michael W Schmidt, Kim K Baldridge, Jerry A Boatz, Steven T Elbert, Mark S Gordon, Jan H Jensen, Shiro Koseki, Nikita Matsunaga, Kiet A Nguyen, Shujun Su, Theresa L Windus, Michel Dupuis, and John A Montgomery. General atomic and molecular electronic structure system. *Journal of Computational Chemistry*, 14(11):1347–1363, 1993. URL: <http://dx.doi.org/10.1002/jcc.540141112>.
 - [44] Mark S. Gordon. Gamess website. URL: <http://www.msg.ameslab.gov/gamess/>.
 - [45] AMOSGateway. URL: <https://amosgateway.org/> (visited on 2023-07-10).
 - [46] Barry I. Schneider, Klaus Bartschat, Oleg Zatsarinny, Kathryn R. Hamilton, Igor Bray, Armin Scrinzi, Fernando Martin, Jesus Gonzalez Vasquez, Jonathan Tennyson, Jimena D. Gorfinkel, Robert Lucchesse, and Sudhakar Pamidighantam. Atomic and Molecular Scattering Applications in an Apache Airavata Science Gateway. In *Practice and Experience in Advanced Research Computing*, PEARC '20, 270–277. New York, NY, USA, July 2020. Association for Computing Machinery. URL: <https://dl.acm.org/doi/10.1145/3311790.3397342> (visited on 2023-07-10).
 - [47] Barry I. Schneider, Klaus-Bartschat, Oleg Zatsarinny, Igor Bray, Armin Scrinzi, Fernando Martin, Markus Klinker, Jonathan Tennyson, Jimena D. Gorfinkel, and Sudhakar Pamidighantam. A Science Gateway for

- Atomic and Molecular Physics. January 2020. URL: <http://arxiv.org/abs/2001.02286> (visited on 2023-07-10), arXiv:2001.02286.
- [48] Paul Hockett. ePS data: Photoionization calculations archive. 2019. URL: <https://phockett.github.io/ePSdata/> (visited on 2022-02-16).
- [49] Paul Hockett. ePSdata repositories on Zenodo. 2019. URL: <https://zenodo.org/search?page=1&size=20&q=hockett&keywords=Data>.
- [50] NumPy. URL: <https://numpy.org/> (visited on 2023-07-10).
- [51] Pandas - Python Data Analysis Library. URL: <https://pandas.pydata.org/> (visited on 2023-07-10).
- [52] SciPy. URL: <https://scipy.org/> (visited on 2023-07-10).
- [53] Stephan Hoyer and Joe Hamman. Xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1):10, April 2017. URL: <http://openresearchsoftware.metajnl.com/article/10.5334/jors.148/> (visited on 2022-08-03).
- [54] Xarray documentation. URL: <https://docs.xarray.dev/en/latest/index.html> (visited on 2022-08-03).
- [55] Mike Boyle. Spherical Functions Github. April 2022. URL: https://github.com/moble/spherical_functions (visited on 2022-08-03).
- [56] Mike Boyle and Leo C. Stein. Moble/spherical_functions: Release v2022.4.2. Zenodo, May 2023. URL: <https://zenodo.org/record/7960723> (visited on 2023-07-10).
- [57] Mike Boyle, Blair Bonnett, Jon Long, Martin Ling, stiin, Leo C. Stein, Eric Wieser, Dante A. B. Iozzo, Hunter Haglid, John Belmonte, John Long, Mark Wiebe, Yin Li, Zé Vinícius, James Macfarlane, and odidev. Moble/quaternion: Release v2022.4.3. Zenodo, February 2023. URL: <https://zenodo.org/record/7636919> (visited on 2023-07-10).
- [58] Moble/quaternion Github. URL: <https://github.com/moble/quaternion> (visited on 2023-07-10).
- [59] SciPy documentation. URL: <https://docs.scipy.org/doc/scipy/index.html> (visited on 2022-08-03).
- [60] Mark A. Wieczorek and Matthias Meschede. SHtools Github. SHTOOLS, August 2022. URL: <https://github.com/SHTOOLS/SHTOOLS> (visited on 2022-08-03).
- [61] Mark A. Wieczorek and Matthias Meschede. SHTools: Tools for Working with Spherical Harmonics. *Geochemistry, Geophysics, Geosystems*, 19(8):2574–2592, August 2018. URL: <http://doi.wiley.com/10.1029/2018GC007529> (visited on 2022-08-03).
- [62] Mark Wieczorek, MMesch, Elliott Sales de Andrade, Ilya Oshchepkov, xoviat, Benda Xu, Katrin Leinweber, and Andrew Walker. SHTOOLS/SHTOOLS: Version 4.5. Zenodo, September 2019. URL: <https://zenodo.org/record/3457861> (visited on 2023-07-10).
- [63] Mark A. Wieczorek and Matthias Meschede. SHtools Docs. SHTOOLS, August 2022. URL: <https://shtools.github.io/SHTOOLS/> (visited on 2022-08-03).
- [64] Marcus Johansson and Valera Veryazov. Automatic procedure for generating symmetry adapted wavefunctions. *Journal of Cheminformatics*, 9(1):8, February 2017. URL: <https://doi.org/10.1186/s13321-017-0193-3> (visited on 2022-08-03).
- [65] Marcus Johansson. Libmsym Github. July 2022. URL: <https://github.com/mcodev31/libmsym> (visited on 2022-08-03).
- [66] LMFIT documentation. URL: <https://lmfit.github.io/lmfit-py/intro.html> (visited on 2022-08-03).
- [67] Matthew Newville, Till Stensitzki, Daniel B. Allen, and Antonino Ingargiola. LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python. Zenodo, September 2014. URL: <https://zenodo.org/record/11813> (visited on 2022-08-03).
- [68] Xyzpy documentation. URL: <https://xyzpy.readthedocs.io> (visited on 2023-07-10).

- [69] Matplotlib — Visualization with Python. URL: <https://matplotlib.org/> (visited on 2023-07-10).
- [70] HoloViews documentation. URL: <https://holoviews.org/> (visited on 2023-07-10).
- [71] hvPlot documentation. URL: <https://hvplot.holoviz.org/index.html> (visited on 2023-07-10).
- [72] Bokeh. URL: <https://bokeh.org/> (visited on 2023-07-10).
- [73] Plotly. URL: <https://plotly.com/python/> (visited on 2023-07-10).
- [74] Seaborn documentation. URL: <https://seaborn.pydata.org> (visited on 2023-07-10).
- [75] Michael Waskom. Seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, April 2021. URL: <https://joss.theoj.org/papers/10.21105/joss.03021> (visited on 2023-07-10).
- [76] Numba: A High Performance Python Compiler. URL: <https://numba.pydata.org/> (visited on 2023-07-10).
- [77] Docker: Accelerated, Containerized Application Development. May 2022. URL: <https://www.docker.com/> (visited on 2023-07-10).
- [78] Jupyter Docker Stacks documentation. URL: <https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html> (visited on 2023-07-10).
- [79] Jonathan Underwood. Limapack. March 2021. URL: <https://github.com/jonathanunderwood/limapack> (visited on 2023-07-10).
- [80] Tamar Seideman. Time-resolved photoelectron angular distributions: concepts, applications, and directions. *Annual review of physical chemistry*, 53:41–65, January 2002. URL: <http://www.ncbi.nlm.nih.gov/pubmed/11972002> (visited on 2012-07-17), arXiv:11972002.
- [81] Tamar Seideman. Time-resolved photoelectron angular distributions as a probe of coupled polyatomic dynamics. *Physical Review A*, 64(4):042504, September 2001. URL: <http://link.aps.org/doi/10.1103/PhysRevA.64.042504> (visited on 2013-03-14).
- [82] Christopher Gerry and Peter Knight. *Introductory Quantum Optics*. Cambridge University Press, 2005.
- [83] Richard N Zare. *Angular Momentum: Understanding Spatial Aspects in Chemistry and Physics*. John Wiley & Sons, 1988.
- [84] Katharine L. Reid, David J. Leahy, and Richard N. Zare. Effect of breaking cylindrical symmetry on photoelectron angular distributions resulting from resonance-enhanced two-photon ionization. *The Journal of Chemical Physics*, 95(3):1746, 1991. URL: <http://scitation.aip.org/content/aip/journal/jcp/95/3/10.1063/1.461023>.
- [85] Guorong Wu, Paul Hockett, and Albert Stolow. Time-resolved photoelectron spectroscopy: from wavepackets to observables. *Physical chemistry chemical physics : PCCP*, 13(41):18447–67, November 2011. URL: <http://pubs.rsc.org/en/content/articlelanding/2011/cp/c1cp22031d>.
- [86] Yasuki Arasaki, Kazuo Takatsuka, Kwanhsing Wang, and Vincent McKoy. Probing wavepacket dynamics with femtosecond energy- and angle-resolved photoelectron spectroscopy. *Journal of Electron Spectroscopy and Related Phenomena*, 108(1-3):89–98, 2000. URL: <http://www.sciencedirect.com/science/article/B6TGC-40T9H2X-B/2/ea056307101b30df942fc8387a61867d>.
- [87] Toshinori Suzuki and Benjamin J. Whitaker. Non-adiabatic effects in chemistry revealed by time-resolved charged-particle imaging. *International Reviews in Physical Chemistry*, 20(3):313–356, July 2001. URL: <http://www.tandfonline.com/doi/abs/10.1080/01442350110045046> (visited on 2012-07-17).
- [88] Albert Stolow and Jonathan G. Underwood. Time-Resolved Photoelectron Spectroscopy of Non-Adiabatic Dynamics in Polyatomic Molecules. In Stuart A. Rice, editor, *Advances in Chemical Physics*, volume 139, pages 497–584. John Wiley & Sons, Inc., Hoboken, NJ, USA, March 2008. URL: <http://doi.wiley.com/10.1002/9780470259498> (visited on 2012-07-18).
- [89] P R Bunker and P Jensen. *Molecular Symmetry and Spectroscopy*. NRC Research Press, Ottawa, 2nd edition, 1998.

- [90] R Signorell and F Merkt. General symmetry selection rules for the photoionization of polyatomic molecules. *Molecular Physics*, 92(5):793–804, 1997.
- [91] Uwe Becker. Complete photoionisation experiments. *Journal of Electron Spectroscopy and Related Phenomena*, 96(1-3):105–115, November 1998. URL: <http://linkinghub.elsevier.com/retrieve/pii/S0368204898002266> (visited on 2012-10-02).
- [92] Katharine L Reid. Photoelectron angular distributions. *Annual review of physical chemistry*, 54(19):397–424, January 2003. URL: <http://dx.doi.org/10.1146/annurev.physchem.54.011002.103814> (visited on 2012-07-17).
- [93] H. Kleinpoppen, B. Lohmann, A. Grum-Grzhimailo, and U. Becker. Approaches to Perfect/Complete Scattering Experiments in Atomic and Molecular Physics. In H. H. Stroke, editor, *Advances In Atomic, Molecular, and Optical Physics*, volume 51, pages 471–534. Academic Press, January 2005. URL: <https://www.sciencedirect.com/science/article/pii/S1049250X05510243> (visited on 2023-07-31).
- [94] Hans Kleinpoppen, Bernd Lohmann, and Alexei N Grum-Grzhimailo. *Perfect/Complete Scattering Experiments*. Volume 75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-40513-6. URL: <https://books.google.ca/books?id=Cia4BAAAQBAJ>.
- [95] Katharine L. Reid and Jonathan G. Underwood. Extracting molecular axis alignment from photoelectron angular distributions. *The Journal of Chemical Physics*, 112(8):3643, 2000. URL: <http://link.aip.org/link/JCPA6/v112/i8/p3643/s1&Agg=doi>.
- [96] Jonathan G. Underwood and Katharine L. Reid. Time-resolved photoelectron angular distributions as a probe of intramolecular dynamics: Connecting the molecular frame and the laboratory frame. *The Journal of Chemical Physics*, 113(3):1067, 2000. URL: <http://link.aip.org/link/JCPA6/v113/i3/p1067/s1&Agg=doi> (visited on 2012-07-17).
- [97] Karl Blum. *Density Matrix Theory and Applications*. Volume 64. Springer Berlin Heidelberg, Berlin, Heidelberg, 3rd editio edition, 2012. ISBN 978-3-642-20560-6. URL: <http://link.springer.com/10.1007/978-3-642-20561-3>.
- [98] Margaret Gregory, 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(14):145601, July 2021. URL: <https://doi.org/10.1088/1361-6455/ac135f> (visited on 2021-08-17), arXiv:2012.04561.
- [99] Margaret Gregory, Simon Neville, Michael Schuurman, and Varun Makhija. A laboratory frame density matrix for ultrafast quantum molecular dynamics. *The Journal of Chemical Physics*, 157(16):164301, October 2022. URL: <https://aip.scitation.org/doi/10.1063/5.0109607> (visited on 2022-10-31).
- [100] G. Mauro D'Ariano, Matteo G.A. Paris, and Massimiliano F. Sacchi. Quantum Tomography. In *Advances in Imaging and Electron Physics*, Vol. 128, pages 205–308. 2003. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1076567003800654> (visited on 2017-09-11).
- [101] Malte C Tichy, Florian Mintert, and Andreas Buchleitner. Essential entanglement for atomic and molecular physics. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 44(19):192001, October 2011. URL: <http://stacks.iop.org/0953-4075/44/i=19/a=192001?key=crossref.18d3f3352e48809821ebdd35c6d00cb6> (visited on 2014-08-19).
- [102] Joel Yuen-Zhou, Jacob J Krich, Ivan Kassal, Allan S Johnson, and Alán Aspuru-Guzik. *Ultrafast Spectroscopy: Quantum Information and Wavepackets*. IOP Publishing, 2014. ISBN 978-0-7503-1062-8. URL: <http://iopscience.iop.org/book/978-0-750-31062-8> (visited on 2017-09-25).
- [103] J. R. Johansson, P. D. Nation, and Franco Nori. QuTiP: An open-source Python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 183(8):1760–1772, August 2012. URL: <https://www.sciencedirect.com/science/article/pii/S0010465512000835> (visited on 2023-02-25).
- [104] J. R. Johansson, P. D. Nation, and Franco Nori. QuTiP 2: A Python framework for the dynamics of open quantum systems. *Computer Physics Communications*, 184(4):1234–1240, April 2013. URL: <https://www.sciencedirect.com/science/article/pii/S0010465512003955> (visited on 2023-02-25).

- [105] QuTiP - Quantum Toolbox in Python. URL: <https://qutip.org/> (visited on 2023-02-25).
- [106] Fabio Benatti, Mark Fannes, Roberto Floreanini, and Dimitri Petritis, editors. *Quantum Information, Computation and Cryptography: An Introductory Survey of Theory, Technology and Experiments*. Volume 808 of Lecture Notes in Physics. Springer, Berlin, Heidelberg, 2010. ISBN 978-3-642-11913-2 978-3-642-11914-9. URL: <https://link.springer.com/10.1007/978-3-642-11914-9> (visited on 2023-02-25).
- [107] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. ISBN 978-0-511-97666-7. URL: <https://www.cambridge.org/highereducation/books/quantum-computation-and-quantum-information/01E10196D0A682A6AEFFEA52D53BE9AE>.
- [108] Henrik Stapelfeldt and Tamar Seideman. Colloquium: Aligning molecules with strong laser pulses. *Reviews of Modern Physics*, 75(2):543–557, April 2003. URL: http://rmp.aps.org/abstract/RMP/v75/i2/p543_1 (visited on 2013-01-28).
- [109] Hirokazu Hasegawa and Yasuhiro Ohshima. Nonadiabatic Molecular Alignment and Orientation. In Kaoru Yamamoto, Luis Roso, Ruxin Li, Deepak Mathur, and Didier Normand, editors, *Progress in Ultrafast Intense Laser Science XII*, Springer Series in Chemical Physics, pages 45–64. Springer International Publishing, Cham, 2015. URL: https://doi.org/10.1007/978-3-319-23657-5_3 (visited on 2022-08-26).
- [110] Christiane P Koch, Mikhail Lemeshko, and Dominique Sugny. Quantum control of molecular rotation. *Reviews of Modern Physics*, 91(3):035005, 2019.
- [111] Jens H Nielsen, Dominik Pentlehner, Lars Christiansen, Benjamin Shepperson, Anders A Søndergaard, Adam S Chatterley, James D Pickering, Constant A Schouder, Alberto Viñas Muñoz, Lorenz Kranabetter, and others. Laser-induced alignment of molecules in helium nanodroplets. In *Molecules in Superfluid Helium Nanodroplets: Spectroscopy, Structure, and Dynamics*, pages 381–445. Springer International Publishing Cham, 2022.
- [112] S Ramakrishna and Tamar Seideman. On the information content of time- and angle-resolved photoelectron spectroscopy. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 45(19):194012, October 2012. URL: <http://stacks.iop.org/0953-4075/45/i=19/a=194012?key=crossref.68faa78abc832ed11020afde085ac486> (visited on 2012-11-06).
- [113] S. Ramakrishna and Tamar Seideman. Rotational wave-packet imaging of molecules. *Physical Review A*, 87(2):023411, February 2013. URL: <http://link.aps.org/doi/10.1103/PhysRevA.87.023411> (visited on 2014-02-10).
- [114] Paul Hockett. General phenomenology of ionization from aligned molecular ensembles. *New Journal of Physics*, 17(2):023069, February 2015. URL: <http://dx.doi.org/10.1088/1367-2630/17/2/023069> <http://stacks.iop.org/1367-2630/17/i=2/a=023069?key=crossref.bdbbf6f53e1f801f11c6bfeca01330fde>.
- [115] Varun Makhija. *Laser-Induced Rotational Dynamics As a Route To Molecular Frame Measurements*. PhD thesis, Kansas State University, 2014. URL: <http://hdl.handle.net/2097/18522>.
- [116] C. Yang. On the Angular Distribution in Nuclear Reactions and Coincidence Measurements. *Physical Review*, 74(7):764–772, October 1948. URL: <http://link.aps.org/doi/10.1103/PhysRev.74.764> (visited on 2015-01-15).
- [117] D Dill. Fixed-molecule photoelectron angular distributions. *The Journal of Chemical Physics*, 65(3):1130–1133, 1976. URL: <http://scitation.aip.org/content/aip/journal/jcp/65/3/10.1063/1.433187> (visited on 2014-04-03).
- [118] S. L. Altmann and C. J. Bradley. On the Symmetries of Spherical Harmonics. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 255(1054):199–215, January 1963. URL: <http://rsta.royalsocietypublishing.org/cgi/doi/10.1098/rsta.1963.0002> (visited on 2012-07-17).
- [119] \relax SL Altmann and \relax AP Cracknell. Lattice harmonics I. Cubic groups. *Reviews of Modern Physics*, 37(1):19–32, 1965. URL: http://rmp.aps.org/abstract/RMP/v37/i1/p19_1 (visited on 2013-06-12).
- [120] N Chandra. Photoelectron spectroscopic studies of polyatomic molecules. I. Theory. *Journal of Physics B: Atomic and Molecular Physics*, 20(14):3405–3415, July 1987. URL: <http://stacks.iop.org/0022-3700/20/i=14/a=013?key=crossref.84d7b9236af8a867d51605ee407558b9>.

- [121] Katharine L. Reid and Ivan Powis. Symmetry considerations in molecular photoionization: Fixed molecule photo-electron angular distributions in C₃v molecules as observed in photoelectron–photoion coincidence experiments. *The Journal of Chemical Physics*, 100(2):1066, 1994. URL: <http://scitation.aip.org/content/aip/journal/jcp/100/2/1063/1.466638>.
- [122] \Spherical harmonics\|. Spherical harmonics. *Wikipedia*, June 2023. URL: https://en.wikipedia.org/w/index.php?title=Spherical_harmonics&oldid=1159615082 (visited on 2023-07-13).
- [123] B Schmidtke, M Drescher, N a Cherepkov, and U Heinzmann. On the impossibility to perform a complete valence-shell photoionization experiment with closed-shell atoms. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 33(13):2451–2465, July 2000. URL: <http://stacks.iop.org/0953-4075/33/i=13/a=306?key=crossref.ec62c50a4abb6a8ccb8209c7b3c89478>.
- [124] P. Hockett, M. Wollenhaupt, C. Lux, and T. Baumert. Complete Photoionization Experiments via Ultrafast Coherent Control with Polarization Multiplexing. *Physical Review Letters*, 112(22):223001, June 2014. URL: <http://link.aps.org/doi/10.1103/PhysRevLett.112.223001>.
- [125] Paul Hockett, Matthias Wollenhaupt, Christian Lux, and Thomas Baumert. Complete photoionization experiments via ultrafast coherent control with polarization multiplexing. II. Numerics and analysis methodologies. *Physical Review A*, 92(1):013411, July 2015. URL: <http://link.aps.org/doi/10.1103/PhysRevA.92.013411>.
- [126] P Hockett, M Wollenhaupt, and T Baumert. Coherent control of photoelectron wavepacket angular interferograms. *Journal of Physics B: Atomic, Molecular and Optical Physics*, 48(21):214004, November 2015. URL: <http://stacks.iop.org/0953-4075/48/i=21/a=214004?key=crossref.8f534585af9180a499934cb25c9994c1>.
- [127] Rick Trebino. *Frequency-Resolved Optical Gating: The Measurement of Ultrashort Laser Pulses*. Springer New York, NY, 2000. ISBN 978-1-4613-5432-1. URL: <https://link.springer.com/book/10.1007/978-1-4615-1181-6> (visited on 2022-05-02).
- [128] Varun Makhija, Xiaoming Ren, Drue Gockel, Anh-Thu Le, and Vinod Kumarappan. Orientation resolution through rotational coherence spectroscopy. *arXiv preprint arXiv:1611.06476*, 2016.
- [129] Xu Wang, Anh-Thu Le, Zhaoyan Zhou, Hui Wei, and CD Lin. Theory of retrieving orientation-resolved molecular information using time-domain rotational coherence spectroscopy. *Physical Review A*, 96(2):023424, 2017.
- [130] Péter Sándor, Adonay Sissay, François Mauger, Paul M Abanador, Timothy T Gorman, Timothy D Scarborough, Mette B Gaarde, Kenneth Lopata, Kenneth J Schafer, and Robert R Jones. Angle dependence of strong-field single and double ionization of carbonyl sulfide. *Physical Review A*, 98(4):043425, 2018.
- [131] Péter Sándor, Adonay Sissay, François Mauger, Mark W Gordon, TT Gorman, TD Scarborough, Mette B Gaarde, Kenneth Lopata, KJ Schafer, and RR Jones. Angle-dependent strong-field ionization of halomethanes. *The Journal of chemical physics*, 151(19):194308, 2019.
- [132] Tomthin Nganba Wangjam, Huynh Van Sa Lam, and Vinod Kumarappan. Strong-field ionization of the triplet ground state of o 2. *Physical Review A*, 104(4):043112, 2021.
- [133] Huynh Van Sa Lam, Suresh Yarlagadda, Anbu Venkatachalam, Tomthin Nganba Wangjam, Rajesh K Kushawaha, Chuan Cheng, Peter Svhra, Andrei Nomerotski, Thomas Weinacht, Daniel Rolles, and others. Angle-dependent strong-field ionization and fragmentation of carbon dioxide measured using rotational wave packets. *Physical Review A*, 102(4):043119, 2020.
- [134] Huynh Van Sa Lam, Tomthin Nganba Wangjam, and Vinod Kumarappan. Alignment dependence of photoelectron angular distributions in the few-photon ionization of molecules by ultraviolet pulses. *Physical Review A*, 105(5):053109, 2022.
- [135] Jonathan G. Underwood, I. Procino, L. Christiansen, J. Maurer, and H. Stapelfeldt. Velocity map imaging with non-uniform detection: Quantitative molecular axis alignment measurements via Coulomb explosion imaging. *Review of Scientific Instruments*, 86(7):073101, July 2015. URL: <https://aip.scitation.org/doi/10.1063/1.4922137> (visited on 2022-09-08), [arXiv:1502.04007](https://arxiv.org/abs/1502.04007).

- [136] Lixin He, Pengfei Lan, Anh-Thu Le, Baoning Wang, Bincheng Wang, Xiaosong Zhu, Peixiang Lu, and C. D. Lin. Real-Time Observation of Molecular Spinning with Angular High-Harmonic Spectroscopy. *Physical Review Letters*, 121(16):163201, October 2018. URL: <https://link.aps.org/doi/10.1103/PhysRevLett.121.163201> (visited on 2023-03-09).
- [137] Yanqing He, Lixin He, Pu Wang, Bincheng Wang, Siqi Sun, Ruxuan Liu, Baoning Wang, Pengfei Lan, and Peixiang Lu. Measuring the rotational temperature and pump intensity in molecular alignment experiments via high harmonic generation. *Optics Express*, 28(14):21182–21191, July 2020. URL: <https://opg.optica.org/oe/abstract.cfm?uri=oe-28-14-21182> (visited on 2023-03-09).
- [138] V. Loriot, R. Tehini, E. Hertz, B. Lavorel, and O. Faucher. Snapshot imaging of postpulse transient molecular alignment revivals. *Physical Review A*, 78(1):013412, July 2008. URL: <http://link.aps.org/doi/10.1103/PhysRevA.78.013412> (visited on 2013-02-27).
- [139] Pu Wang, Lixin He, Yanqing He, Jianchang Hu, Siqi Sun, Pengfei Lan, and Peixiang Lu. Rotational echo spectroscopy for accurate measurement of molecular alignment. *Optics Letters*, 47(5):1033–1036, March 2022. URL: <https://opg.optica.org/ol/abstract.cfm?uri=ol-47-5-1033> (visited on 2023-03-09).
- [140] By P W Atkins, M S Child, and C S G Phillips. *Tables for Group Theory*. Oxford University Press, 2006. URL: <https://global.oup.com/uk/orc/chemistry/qchem2e/student/tables/>.
- [141] Achim Gelessus. Character tables for chemically important point groups. URL: <http://symmetry.jacobs-university.de/> (visited on 2023-04-23).
- [142] Gernot Katzer. Character Tables for Point Groups Cn, Cnv, Cnh, Dn, Dnh, Dnd, S2n etc. URL: http://www.gernot-katzers-spice-pages.com/character_tables/index.html (visited on 2023-04-23).
- [143] Paul Hockett. Molecular Frame Reconstruction using Time-domain Photoionization Interferometry (Figshare). *Figshare*, May 2017. URL: https://figshare.com/articles/journal_contribution/Bootstrapping_to_the_Molecular_Frame_with_Time-domain_Photoionization_Interferometry/4480349/10 (visited on 2023-07-27).
- [144] Jo Woodhose. OCS experimental work (personal communication, manuscript in preparation). 2022.
- [145] L D Landau and E M Lifshitz. *Quantum Mechanics (Non-relativistic Theory)*. Pergamon Press, 3 edition, 1977.
- [146] Albert Messiah. *Quantum Mechanics Volume I*. North-Holland Publishing Company, 1970.
- [147] Jun John Sakurai. *Modern Quantum Mechanics*. Addison-Wesley, Reading, MA, revised edition edition, 1994. URL: <https://cds.cern.ch/record/1167961>.
- [148] \Euler angles\ . Euler angles. *Wikipedia*, June 2023. URL: https://en.wikipedia.org/w/index.php?title=Euler_angles&oldid=1160186404 (visited on 2023-07-17).
- [149] \Wigner D-matrix\ . Wigner D-matrix. *Wikipedia*, June 2023. URL: https://en.wikipedia.org/w/index.php?title=Wigner_D-matrix&oldid=1160193091 (visited on 2023-07-17).

INDEX

A

ADMs, **209**
AF, **209**
anisotropy paramters, **209**
axis distribution moments, **209**

B

bootstrap retrieval protocol, **211**

C

channel functions, **210**

E

Euler angles, **210**

F

frame rotation, **210**

G

geometric coupling parameters, **210**

H

HOMO, **210**

L

LF, **209**

M

MF, **209**
molecular alignment, **211**
MS, **209**

P

PADs, **209**
partial-wave expansion, **210**
partial-waves, **210**
PG, **209**

R

radial matrix elements, **210**
RWP, **210**

S

symmetrized harmonics, **210**

V

VMI, **210**

W

Wigner rotation matrix elements, **210**