Quantum Metrology with Photoelectrons Vol. 3 *Analysis methodologies*

Paul Hockett with Varun Makhija

Jul 22, 2023

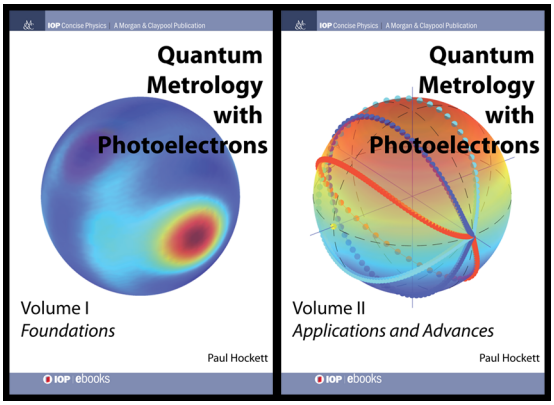
CONTENTS

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.

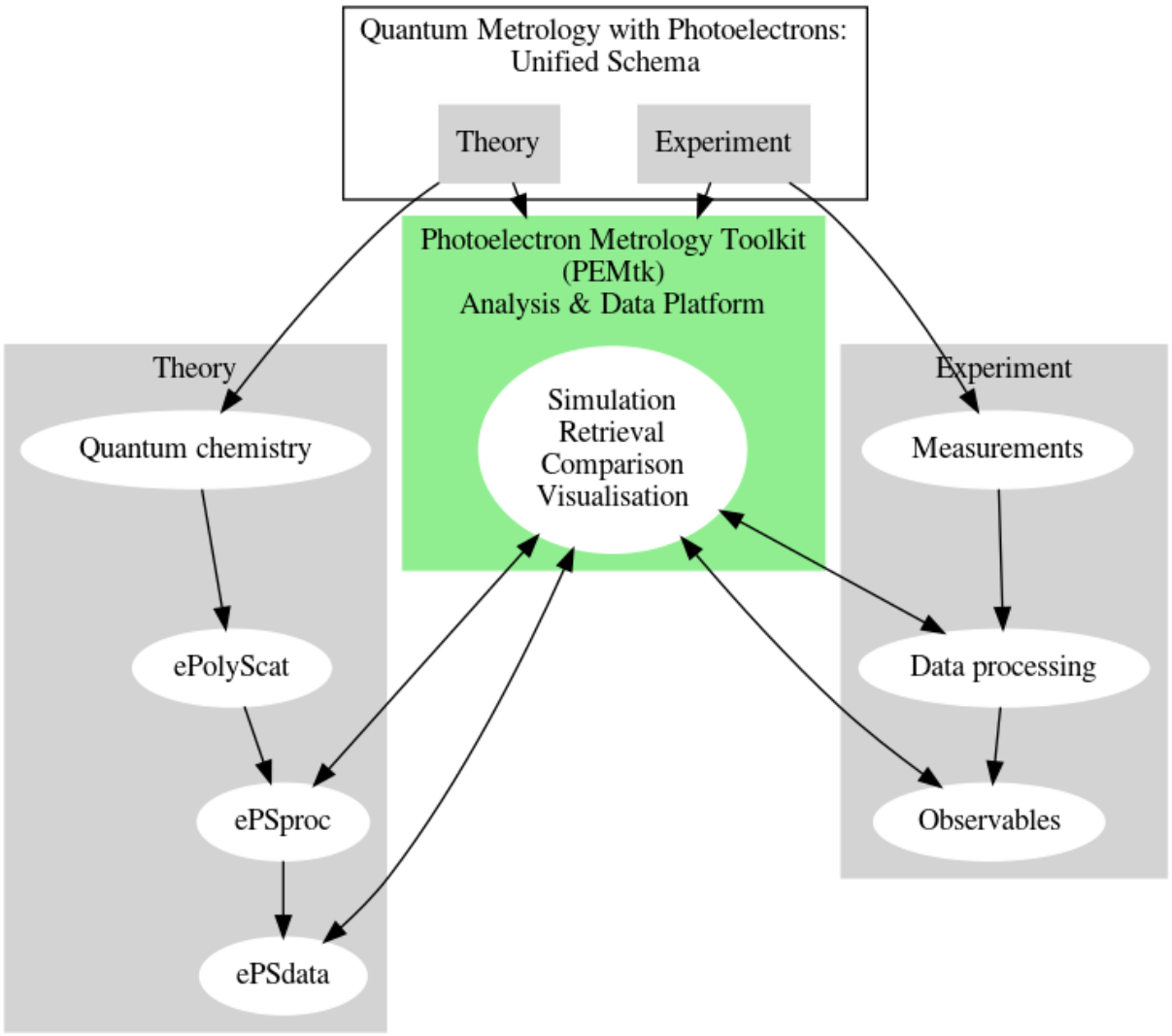
Technical details

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:

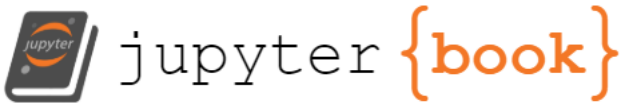
1. Clone this repository
2. Run `pip install -r requirements.txt` (it is recommended you do this within a virtual environment)
3. (Optional) Edit the books source files located in the `doc-source/` directory
4. Run `jupyter-book clean doc-source/` to remove any existing builds
5. For an HTML build:
    - 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. Varun is a faculty member at the University of Mary Washington, Virginia, USA.

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. 5.2.

CHAPTER

THREE

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.

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. 5.2.2.

3.3 Formatting

In some cases additional formatting is required for defining Jupter Notebook to HTML and PDF outputs (via the Jupyter Book build-chain, see Sect. 5.2.2), 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.

3.4 Numerics

At the time of writing the main code-bases used in this work (see Sect. 6 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]), 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.

CHAPTER

FOUR

OVERVIEW

4.1 General overview

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

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

4.2 Provisional contents

4.2.1 Part 1: theory & software

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

1. Introduction

   a. Topic overview.

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

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

2. Quantum metrology software platform/ecosystem overview

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

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

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

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

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

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

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

4. Numerical implementation & analysis platform tools

   a. Tensor methods implementation in ePSproc/PEMtk.

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

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

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

4.2.2 Part 2: numerical examples

Open-source worked examples using the new software platform.

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

   a. Experimental data overview & simulation.

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

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

   d. Comparison of methods.

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

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

   a. Experimental data overview & simulation.

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

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

   d. Comparison of methods.

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

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

   a. Experimental data overview & simulation.

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

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

   d. Comparison of methods.

   e. Information content/quantum information analysis.

4. Future directions & outlook

5. Summary & conclusions

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

Part II

Part I - Theory & software

CHAPTER

FIVE

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. 5.1 below), and the necessary core physics,as well as some recent extensions, is also presented herein (Chapter 7). The unfamiliar reader is referred to *Quantum Metrology* Vol. 1 [2] 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 6), recent theory developments (Chapter 7), and concludes with a general overview for approaching matrix element retrieval numerically (Chapter 8).

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 [3] fitting routines (see Chapter 9 for an outline), then proceeding with a (relatively) simple homonuclear diatomic example (Chapter 12), and escalating in complexity to a the most general polyatomic asymmetric top case.

5.1 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 [2] are reproduced below; the reader is referred to the introductory chapters of *Quantum Metrology* Vol. 1 [2] for a lengthier treatment, and an introductory video to *Phase-sensitive Photoelectron Metrology* can be found online (see also Refs. [4] for further introductory presentation videos, materials and resources around the topic).

5.1.1 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 [2], 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 [2], 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 introducted in more detail (including examples) in Sect. 7.6.

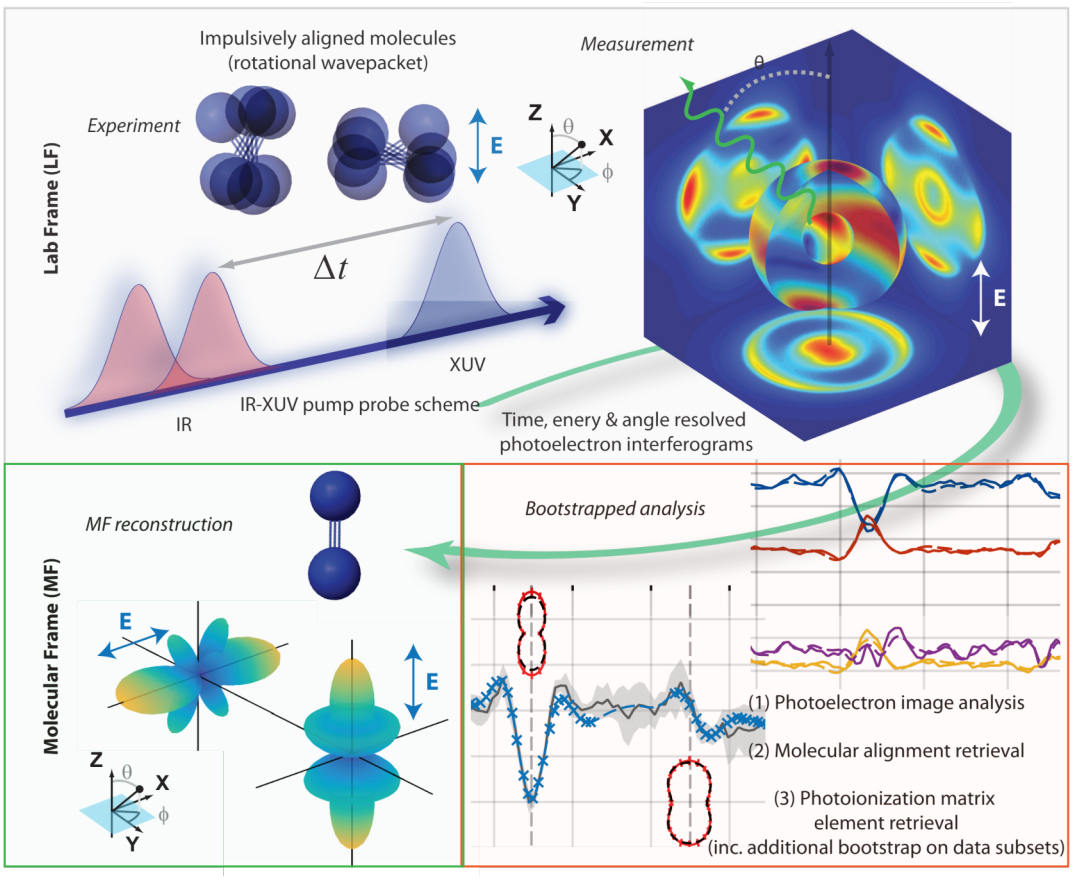


Fig. 5.1: Conceptual outline for the generalised bootstrap matrix element retrieval protocol, including retrieval of *MF* properties, via a set of time-resolved measurements and suitable post-processing scheme. In the *LF*/*AF*, a set of laser pulses creates and probes an aligned distribution of molecules, 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 show 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. 7.1 for a more detailed frame definition.

5.1.2 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 [2, 5]). 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 [5]). 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 bootstrapping* protocol, which makes use of experiments using rotational wavepackets 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 [5] (in particular Chpt. 11), and see also Ref. [1] for an experimental demonstration, and Refs. [] for a recent review in the context of molecular frame reconstruction.

As defined in *Quantum Metrology* Vol. 2 [5]:

> 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. 5.1, and outlined in more detail in Fig. 8.1 …] 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 [6]; 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 [5], 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 $> 10^{12}$~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 minumum - 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 8.
    - 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 [5] for more discussion; recent examples of such work may be found in Refs. [], 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. 5.2), 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.

5.2 Context & aims for Vol. 3

5.2.1 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 [5] Sect. 12.3; in particular the new Photoelectron Metrology Toolkit [3] 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 6), along with the theory (Chapter 7) and numerics (Chapter 8) 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 [2, 5], 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. 8.1.3.

5.2.2 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 [2, 5] 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 [3], introduced in more detail in Chapter 6. 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 [7].[1]
    - These are `.ipynb` files, usually running a Python kernel [8], 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 [9, 10] project platform,[2] which includes build tools and specifications for the specific flavour of Markdown (MyST) [11] used for the written text, and uses Sphinx [12] 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 [3].
    - 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 6: *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. 6.4.2: *Docker deployments* (see also the Open Photoionization Docker Stacks [13] 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 [3], 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. [7, 22, 23].)
[2] For more information see jupyterbook.org and Refs. [9, 10].

- 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 [14].

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 difficultly, 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 [15, 16].

5.2.3 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. [17, 18, 19, 20], or the Wikipeadia Open Science page for a brief summary [21].

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. 6. 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. 6.4.2.

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.

CHAPTER

SIX

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. 6.1 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 [3] 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.

6.1 Analysis components

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

- The Photoelectron Metrology Toolkit [3] (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 [14].
- The ePSproc codebase [24, 25, 26] aims to provide methods for post-processing with *ab initio* radial dipole matrix elements from ePolyScat (ePS) [27, 28, 29, 30], or equivalent matrix elements from other sources (dedicated support for R-matrix results from the RMT suite [31, 32] is in development, for an overview of *ab initio* methods/packages see Ref. [33]). 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 [26].

Warning: As *noted elsewhere*, many components of the toolkit are still in active development, and some numerical details may change.

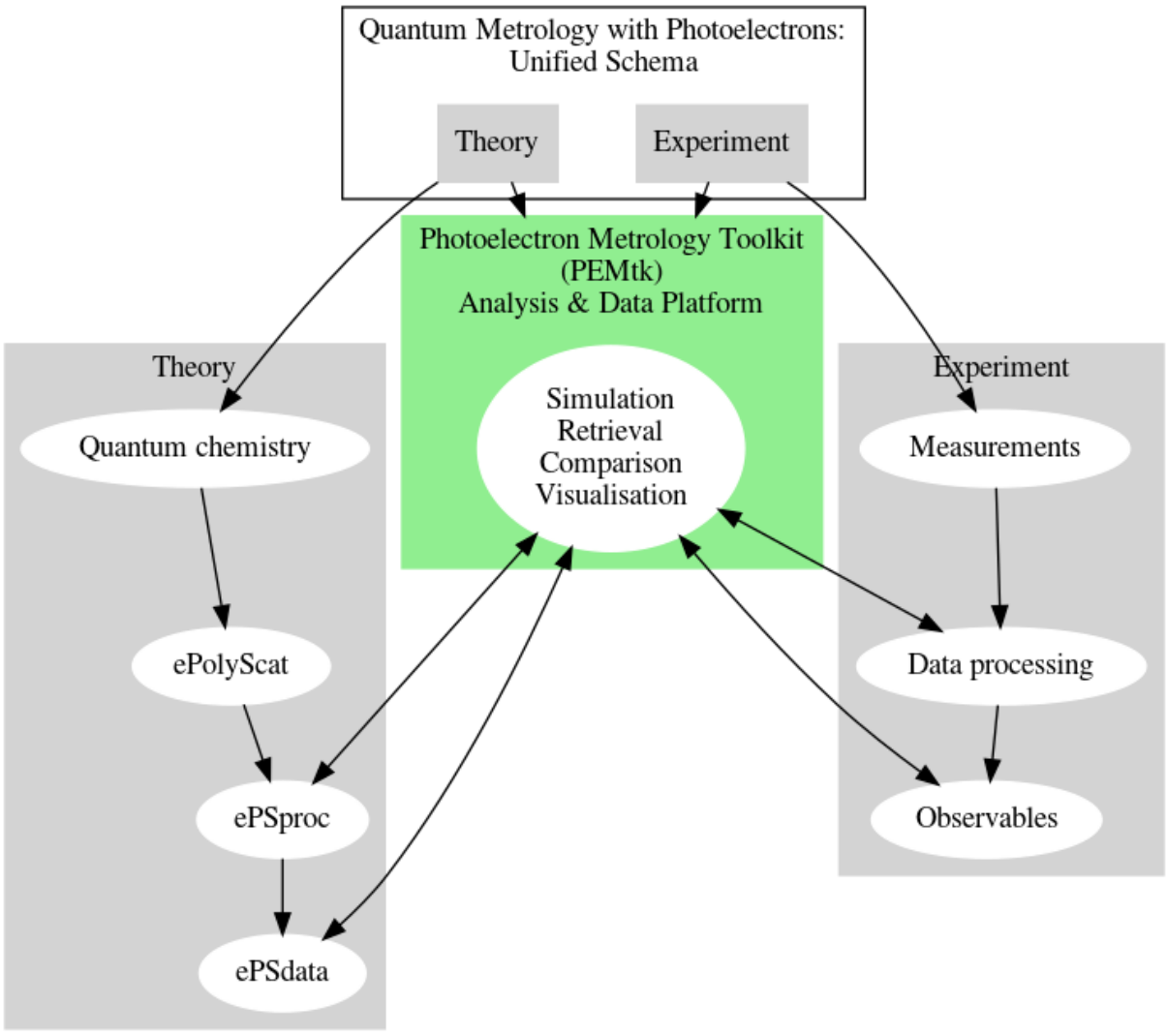


Fig. 6.1: Quantum metrology with photoelectrons ecosystem overview.

6.2 Additional tools

Other tools listed in Fig. 6.1 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") [34, 35] 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 [25] 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) [27, 28, 29, 30] 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 [30] and Refs. [27, 28, 29].
  - A Docker build is available (via the Open Photoionization Docker Stacks [13] project).
  - ePS (along with a range of other computational AMO tools) is also available online via the AMOS gateway[1] [36, 37, 38].
- ePSdata [39] 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 [39] Github project repository, and notebooks + datasets via ePSdata Zenodo [40]. 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.

6.3 Python ecosystem (backends, libraries and packages)

The core analysis tools, which constitute the Photoelectron Metrology Toolkit [3] 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` [41] for general numerical methods and data types.
  - `pandas` [42] for statistical methods, and various tabulation and sorting tasks.
  - `Scipy` [43] 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 [44, 45].
- Angular momentum functions

[1] Formerly known as the AMP gateway.

  - Wigner D and 3js are currently implemented directly, or via the Spherical Functions library [46, 47], and have been tested for consistency with the definitions in Zare (for details see the ePSproc docs [26]). The Spherical Functions library also uses `quaternion` [48, 49] 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 [50]. Further manipulation and conversion between different normalisations can be readily implemented with the `pySHtools` library [51, 52, 53, 54]. See Sect. 7.6.1 for examples.
  - Symmetry functionality, specifically computing symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta,\phi)$ (see Eq. (7.37)), makes use of `libmsym` [55, 56] (symmetry coefficients) and `pySHtools` [51, 52, 53, 54] (general spherical harmonic handling and conversion). See Sect. 7.6.2 for examples.

- Non-linear optimization (fitting):
  - Fitting is handled via the `lmfit` library [57, 58], 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 [50], hence this is the core numerical minimization routine for the demonstration cases herein. (See Chapter 8 for further discussion of fitting methods.)
  - Basic parallelization for fitting routines is implemented using the `xyzpy` library [59], see Chapter 11 for further details.

- For plotting a range of tools can be used, some of which are implemented/wrapped in the Photoelectron Metrology Toolkit [3], or can be used directly with `Xarray` data structures, including:
  - `Matplotlib` [60]: basic plotting, including `Xarray` direct plotters.
  - `Holoviews` [61]: used for data handling and interactive plotting, Holoviews is a general plotting tool which wraps various backends; `hvplot` [62] can also be used to provide additional `Pandas` and `Xarray` integration for Holoviews. Most of the plots herein use Holoviews.
  - `Bokeh` [63]: used for interactive plots, implemented in the Photoelectron Metrology Toolkit [3] via Holoviews wrappers/methods.
  - `Plotly` [64]: used in the the Photoelectron Metrology Toolkit [3] and the ePSproc codebase [24, 25, 26] for spherical polar plotting routines.
  - `Seaborn` [65, 66]: used for statistical methods and some specialist plots and styles, particularly the `lmPlot` routine in ePSproc codebase [24, 25, 26].

- 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 [25]: pyvista 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` [67] 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 7 and Chapter 8.

6.4 Installation and environment set-up

6.4.1 Quick-start installation

For a basic installation, up-to-date version of Photoelectron Metrology Toolkit [3] and ePSproc codebase [24, 25, 26] 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).

6.4.2 Docker deployments

Docker [68] 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. 5.2.3).

A Docker-based distribution of various codes for tackling photoionization problems is available from the Open Photoionization Docker Stacks [13] 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 [69], 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 [9, 10] and related tools) as discussed in Sect. 5.2.2. 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.

6.4.3 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 [7]). 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`).

6.5 General platform discussion

Note that, at the time of writing:

- Rotational wavepacket simulation is not yet implemented in the Photoelectron Metrology Toolkit [3], and these must be obtained via other codes. An intial build of the `limapack` suite [70] for rotational wavepacket simulations is currently part of the Open Photoionization Docker Stacks [13], but has yet to be used in this work.
- Fitting has 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 [57, 58] or with other Python libraries or custom codes; optimization making use of `Numba` should also be possible.
- The Photoelectron Metrology Toolkit [3] codebase is currently still under heavy development, so readers may wish to consult the ongoing PEMtk documentation [14] 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 [26], specifically the ePSproc class intro page.
  - For more details of the data structures used, see the ePSproc documentation [26], 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

SEVEN

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 [3] and ePSproc codebase [24, 25, 26] 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. 7.6: and Sect. 7.7: should provide sufficient background for pure reconstruction problems.

7.1 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 [2] (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 photoionzation, see Refs. [71, 72].) 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 [2], 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 $\hat{\Gamma}(\mathbf{E},\mathbf{t})$) 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 photoelecton 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,

$$\left|\Psi_f\right\rangle = \sum \int \left|\Psi_+;\mathbf{k}\right\rangle \left\langle\Psi_+;\mathbf{k}|\Psi_f\right\rangle d\mathbf{k}, \tag{7.1}$$

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

$$\hat{\Gamma}(\mathbf{E}) = \hat{\mu}.\mathbf{E} \tag{7.2}$$

Hence,

$$\left\langle\Psi_+;\mathbf{k}|\Psi_f\right\rangle = \langle\Psi_+;\,\mathbf{k}|\hat{\mu}.\mathbf{E}|\Psi_i\rangle \tag{7.3}$$

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 $\hat{\mu}.\mathbf{E}$ is the usual dipole interaction term [73], which includes a sum over all electrons s defined in position space as $\mathbf{r_s}$:

$$\hat{\mu} = -e\sum_s \mathbf{r_s} \tag{7.4}$$

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

$$\Psi_{\mathbf{k}}(r) \equiv \langle r|\mathbf{k}\rangle = \sum_{lm} Y_{lm}(\hat{\mathbf{k}})\psi_{lm}(r,k) \tag{7.5}$$

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. 7.6).

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. [74] for general discussion), and written generally as (using notation similar to [75]):

$$\langle\Psi_+;\,\mathbf{k}|\hat{\mu}.\mathbf{E}|\Psi_i\rangle = \sum_{lm} \gamma_{l,m}\mathbf{r}_{k,l,m} \tag{7.6}$$

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. (7.6) and other angular-momentum coupling terms - are know, 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. 8).

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:

$$\mathbf{r}_{k,l,m} = \int \psi^*_{lm}(r,k) r \Psi_i(r) dr \tag{7.7}$$

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. 7.3.2 for details).

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

$$I(\epsilon,\theta,\phi) = \left\langle\Psi_f|\Psi_+;\mathbf{k}\right\rangle \left\langle\Psi_+;\mathbf{k}|\Psi_f\right\rangle \tag{7.8}$$

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 [2], 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 [2]):

$$I(\theta,\phi;\,k)=\sum_{ll'}\sum_{\lambda\lambda'}\sum_{mm'}\gamma_{\alpha\alpha_{+}l\lambda ml'\lambda'm'}r_{kl\lambda}r_{kl'\lambda'}e^{i(\eta_{l\lambda}(k)-\eta_{l'\lambda'}(k))}Y_{lm}(\hat{k})Y^{*}_{l'm'}(\hat{k}) \tag{7.9}$$

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. 7.3.2, and includes a full breakdown of these terms and details of numerical implementations.

Comparison of Eq. (7.9) with Eq. (7.37) indicates that the amplitudes in Eq. (7.6) also determine the observable *anisotropy paramters* $\beta_{L,M}(\epsilon,t)$ (Eq. (7.37)), which basically collect all the terms in Eq. (7.9) 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. 7.6:

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. [76] and *Quantum Metrology* Vol. 1 [2]; discussions of more complex cases with electronic and nuclear dynamics can be found in Refs. [72, 77, 78, 79].

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. 7.3.2).

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. (7.9) - 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.

7.2 Symmetry in photoionization

Symmetry in photoionization has briefly been suggested by the introduction of symmetrized harmonics (Sect. 7.6.2), and is discussed in detail in *Quantum Metrology* Vol. 1 [2] (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 [2]; for a more general discussion of symmetry in molecular spectroscopy see the textbook by Bunker and Jensen [80], and the specific case of photoionization is expanded on in the work of Signorell and Merkt [81].

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:

$$\Gamma^{f}_{rve}\otimes\Gamma_{dipole}\otimes\Gamma^{i}_{rve}\supset\Gamma^{s} \tag{7.10}$$

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 [81]:

$$\Gamma^e \otimes \Gamma^+_{rve} \otimes \Gamma_{dipole} \otimes \Gamma^i_{rve} \supset \Gamma^s \tag{7.11}$$

This is, essentially, a statement of the limiting case of Eq. (7.3) (see also alternative forms of Eqs. (7.6), (7.7)), 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:

$$\Gamma^{e(X)} \otimes \Gamma^+_e \otimes \Gamma_{dipole} \otimes \Gamma^i_e \supset \Gamma^s \tag{7.12}$$

And $\Gamma^{e(X)}$ indicates that the continuum symmetries are expressed in a basis of symmetrized harmonics (Sect. 7.6.2). From Eq. (7.12), the set of allowed matrix elements for a given ionization event can be expressed, in terms of the allowed set of symmetrized harmonics $X^{\Gamma\mu *}_{hl}(\theta,\phi)$, or (equivalently) the usual partial wave basis expressed in spherical harmonics $Y_{l,\lambda}(\theta,\phi)$, and a set of associated symmetrization coefficients $b^{\Gamma\mu}_{hl\lambda}$.

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 %s:` 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']}}
```

```
/home/jovyan/github/PEMtk/pemtk/sym/_dipoleTerms.py:102: FutureWarning: iteritems␣
↪is deprecated and will be removed in a future version. Use .items instead.
  for (col, vals) in dipolePD.iteritems():
```

```
{'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 0x7efec6a230d0>
```

```
/opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:344:␣
↪FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise␣
↪the base implementation of `Styler.to_latex` for formatting and rendering. The␣
↪arguments signature may therefore change. It is recommended instead to use␣
↪`DataFrame.style.to_latex` which also contains additional functionality.
  return method()
```

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.displayXlm(symFilter = True)
```

```
/opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:344:␣
↪FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise␣
↪the base implementation of `Styler.to_latex` for formatting and rendering. The␣
↪arguments signature may therefore change. It is recommended instead to use␣
↪`DataFrame.style.to_latex` which also contains additional functionality.
  return method()
```

Character (Γ)	SALC (h)	PFIX (μ)	l m	b 0	1	2	3	4
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

7.3 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 [5] Chpt. 8 for examples and discussion, a range of review articles can also be found in the literature, e.g. Refs. [82, 83, 84, 85]); herein, a geometric tensor based formalism is developed, which is close in spirit to the treatments given by Underwood and co-workers [79, 86, 87], 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 [74, 88], 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 [88] and Zare [74].

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. 7.3.1, in terms of *channel functions*, and the remainder of this section (Sect. 7.3.2 - Section 7.3.9) gives a detailed breakdown of the various components of the *channel functions*, and numerical examples.

7.3.1 Channel functions

A simple form of the equations[1], 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),

$$\beta^{u}_{L,M} = \sum_{\zeta,\zeta'} \Upsilon^{u,\zeta\zeta'}_{L,M} \mathbb{I}^{\zeta\zeta'} \tag{7.13}$$

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

$$\mathbb{I}^{\zeta,\zeta} = I^{\zeta}(\epsilon) I^{\zeta'*}(\epsilon) \tag{7.14}$$

[1] Cf. the general form of Eq. (7.9). See also *Quantum Metrology* Vol. 2 [5] Chpt. 12 for early discussion and motivation for this formalism.

Eq. (7.13) is effectively a convolution equation (cf. Refs. [86, 89]) 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. [27, 28, 29]), these matrix elements are essentially identical to the simplified *radial matrix elements* $\mathbf{r}_{k,l,m}$ defined in Eq. (7.6), 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. (7.6), (7.9), 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. (7.37), 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:

$$I^{\zeta}(\epsilon) \equiv \mathbf{r}_{\zeta} \equiv r_{\zeta} e^{i\phi_{\zeta}} \tag{7.15}$$

This tensorial form is numerically implemented in the ePSproc [25] 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 [3] 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. (7.13), with terms $I^{\zeta}(\epsilon)$ as the unknowns (in magnitude, phase form per (7.15)). The main computational cost of a tensor-based approach is that more RAM is required to store the full set of tensor 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. 7.3.2 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.

7.3.2 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. (7.13) in terms of the relevant tensors. Further details and numerical examples are given in the following sub-sections.

For the *MF*:

$$\begin{aligned}\beta_{L,-M}^{\mu_i,\mu_f}(\epsilon) = & (-1)^M \sum_{P,R',R} [P]^{\frac{1}{2}} E_{P-R}(\hat{e};\mu_0) \\ \times & \sum_{l,m,\mu} \sum_{l',m',\mu'} (-1)^{(\mu'-\mu_0)} \Lambda_{R',R}(R_{\hat{n}};\mu,P,R,R') B_{L,-M}(l,l',m,m') \\ \times & I_{l,m,\mu}^{p_i\mu_i,p_f\mu_f}(\epsilon) I_{l',m',\mu'}^{p_i\mu_i,p_f\mu_f *}(\epsilon)\end{aligned} \tag{7.16}$$

And the *LF*/*AF* as:

$$\begin{aligned}\bar{\beta}_{L,-M}^{\mu_i,\mu_f}(\epsilon,t) = & (-1)^M \sum_{P,R',R} [P]^{\frac{1}{2}} E_{P-R}(\hat{e};\mu_0) \\ \times & \sum_{l,m,\mu} \sum_{l',m',\mu'} (-1)^{(\mu'-\mu_0)} \bar{\Lambda}_{R'}(\mu,P,R') B_{L,S-R'}(l,l',m,m') \\ \times & I_{l,m,\mu}^{p_i\mu_i,p_f\mu_f}(\epsilon) I_{l',m',\mu'}^{p_i\mu_i,p_f\mu_f *}(\epsilon) \sum_{K,Q,S} \Delta_{L,M}(K,Q,S) A_{Q,S}^{K}(t)\end{aligned} \tag{7.17}$$

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).
- $A^K_{Q,S}(t)$: 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 $I^{p_i\mu_i,p_f\mu_f}_{l,m,\mu}(\epsilon)$ are the *radial matrix elements*, as a function of energy ϵ. As noteed above, these *radial matrix elements* are essentially identical to the simplified forms $r_{k,l,m}$ defined in Eqn. (7.6), 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) [27, 28, 29, 30], 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. (7.13) for the *AF* and *MF* can be written as:

$$\begin{aligned} \Upsilon^{u,\zeta\zeta'}_{L,M} = \; & (-1)^M [P]^{\frac{1}{2}} E_{P-R}(\hat{e};\mu_0)(-1)^{(\mu'-\mu_0)} \Lambda_{R',R}(R_{\hat{n}};\mu,P,R,R') \\ \times \; & B_{L,-M}(l,l',m,m') \end{aligned} \tag{7.18}$$

$$\begin{aligned} \bar{\Upsilon}^{u,\zeta\zeta'}_{L,M} = \; & (-1)^M [P]^{\frac{1}{2}} E_{P-R}(\hat{e};\mu_0)(-1)^{(\mu'-\mu_0)} \bar{\Lambda}_{R'}(\mu,P,R') \\ \times \; & B_{L,S-R'}(l,l',m,m') \Delta_{L,M}(K,Q,S) A^K_{Q,S}(t) \end{aligned} \tag{7.19}$$

Note that, in this case as given, time-dependence arises purely from the $A^K_{Q,S}(t)$ 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 [24, 25, 26], and further wrapped for fitting cases in the Photoelectron Metrology Toolkit [3]. 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 [26], including extended discussion of each tensor and complete function references in the geomCalc submodule documentation.

7.3.3 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. 5.1. A more detailed definition is given in Fig. 7.1, 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. 7.1 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.

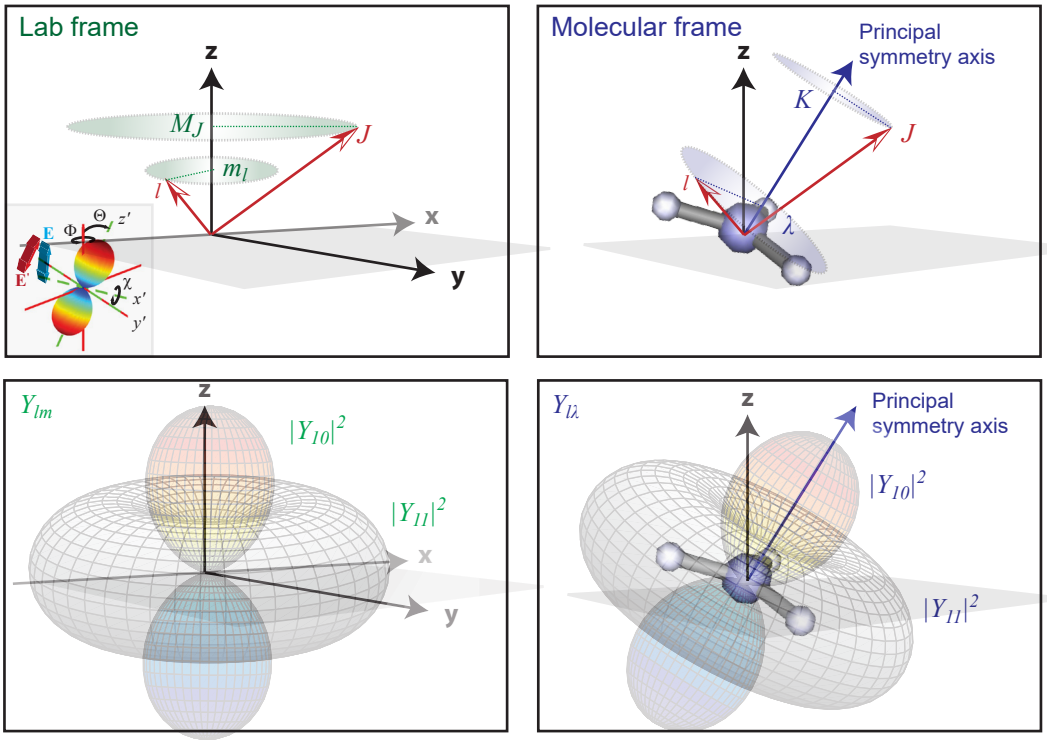


Fig. 7.1: Reference frame and angular momentum definitions for the Laboratory frame (*LF*) and Molecular frame (*MF*), using a general notation from molecular spectroscopy. In this case the *LF* shows an angular momentum vectors J and l; J is usually used to define rotational (or sometimes total) angular 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. 5.1. Figure reproduced from *Quantum Metrology* Vol. 1 [2], Fig. 2.3, and shows - note that some alternative notations are used in this volume.

7.3.4 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. 7.2, and discussion of *symmetrized harmonics* in Sect. 7.6.2.

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 (7.37)) have the property that $\mu = 0$ only, and the h index is redundant, since it maps uniquely to l - see Fig. 7.2 - 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 [3] (see Sect. 10.3.4 for further discussion of the numerical implementation), and used with the standard routines. These are demonstrated below, for two flavours - the base ePSproc [25] routine for computation of *AF-PADs*, and the Photoelectron Metrology Toolkit [3] fitting routine which wraps this functionality. Note that all tensors are stored as `Xarray` [44, 45] 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 (Γ)	PFIX (μ)	l	h m	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	
			4						1.0
A1u	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. 7.2: Symmetrized harmonics coefficients ($b^{\Gamma\mu}_{hl\lambda}$, see (7.37)) for D2h symmetry (l_{max} =4) generated with the Photoelectron Metrology Toolkit [3] wrapper for `libmsym` [55, 56]. Note that, in this case, the coeffcients 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(['QNs', 'EPRX', 'lambdaTerm', 'BLMtable',
 ↪'BLMtableResort', 'AFterm', 'AKQS', 'polProd', 'phaseConvention', 'BLMRenorm',
 ↪'matEmult'])
```

7.3.5 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:

$$B_{L,M} = (-1)^m \left(\frac{(2l+1)(2l'+1)(2L+1)}{4\pi}\right)^{1/2} \begin{pmatrix} l & l' & L \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} l & l' & L \\ -m & m' & M \end{pmatrix} \tag{7.20}$$

Note that this term is equivalent, effectively, to a triple integral over spherical harmonics (e.g. Eq. 3.119 in Zare [74]):

$$\int_0^{2\pi}\int_0^{\pi} Y_{J_3M_3}(\theta,\phi) Y_{J_2M_2}(\theta,\phi) Y_{J_1M_1}(\theta,\phi) \sin\theta d\theta d\phi = \left(\frac{(2J_1+1)(2J_2+1)(2J_3+1)}{4\pi}\right)^{1/2} \times \begin{pmatrix} J_1 & J_2 & J_3 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} J_1 & J_2 & J_3 \\ M_1 & M_2 & M_3 \end{pmatrix}$$

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 [88]) - this is how the term arises in the derivation of the observables.

$$Y_{J_1M_1}(\theta,\phi) Y_{J_2M_2}(\theta,\phi) = \sum_{J_3M_3} \left(\frac{(2J_1+1)(2J_2+1)(2J_3+1)}{4\pi}\right)^{1/2} \times \begin{pmatrix} J_1 & J_2 & J_3 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} J_1 & J_2 & J_3 \\ M_1 & M_2 & M_3 \end{pmatrix} Y^*_{J_3M_3}(\theta,\phi)$$

Note also some definitions use conjugate spherical harmonics, which can be converted as, e.g., Eq. C.21 in Blum [88]:

$$\beta_{L,M}^{\mu_i,\mu_f} Y_{LM}^*(\theta_{\hat{k}},\phi_{\hat{k}}) = \beta_{L,-M}^{\mu_i,\mu_f}(-1)^M Y_{L,-M}(\theta_{\hat{k}},\phi_{\hat{k}}) \tag{7.21}$$

In the current Photoelectron Metrology Toolkit [3] 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. 7.3 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. (7.20)), 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)
```

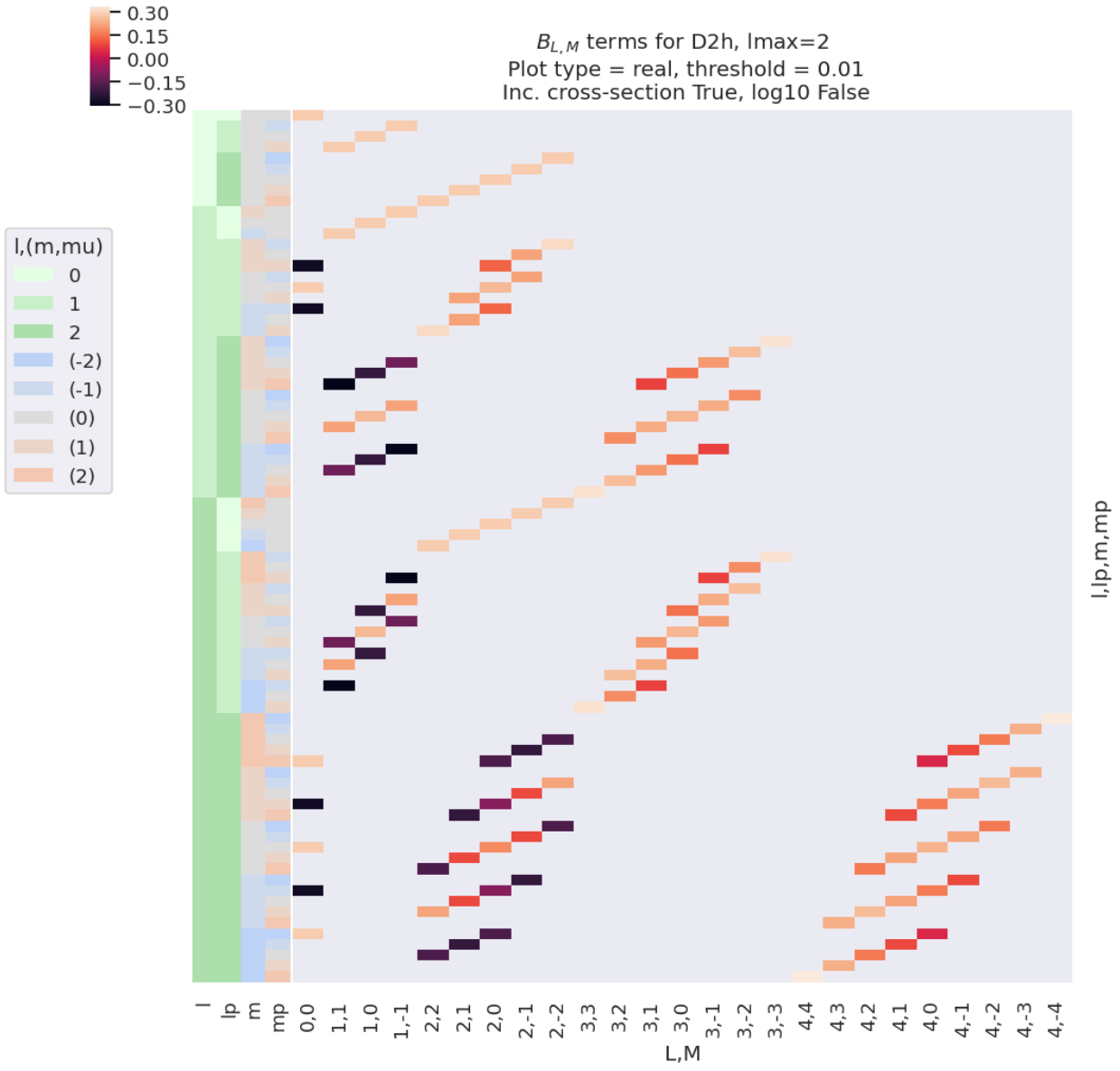


Fig. 7.3: Example $B_{L,M}$ basis functions for D2h symmetry. Note figure is truncated to $l_{max} = l'_{max}$ =2 for clarity.

7.3.6 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. (7.9)) can be written as a tensor contraction:

$$E_{P,R}(\hat{e}) = [e \otimes e^*]^P_R = [P]^{\frac{1}{2}} \sum_p (-1)^R \begin{pmatrix} 1 & 1 & P \\ p & R-p & -R \end{pmatrix} e_p e^*_{R-p} \tag{7.22}$$

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. 5.1 for the linear example), or defined relative to the molecular symmetry axis in the *MF*. (See Sect. 7.3.3 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. (7.22), 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. 7.3.7.
- 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 [25], 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 [74]; for two first-rank tensors (e.g. electric field vectors) this contraction is given explicitly by Eq. 5.40 in Zare [74]:

$$[A^{(1)} \otimes B^{(1)}]^k_q = \sum_m \langle 1m, 1q-m | kq \rangle A(1,m) B(1, q-m) \tag{7.23}$$

Convert to $3j$ form:

$$[A^{(1)} \otimes B^{(1)}]^k_q = \sum_m (-1)^q [k]^{1/2} \begin{pmatrix} 1 & 1 & k \\ m & q-m & -q \end{pmatrix} A(1,m) B(1, q-m) \tag{7.24}$$

And substitute in appropriate terms.

As before, we can visualise these values with the Photoelectron Metrology Toolkit [3].

```
#*** 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 = 'r')

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

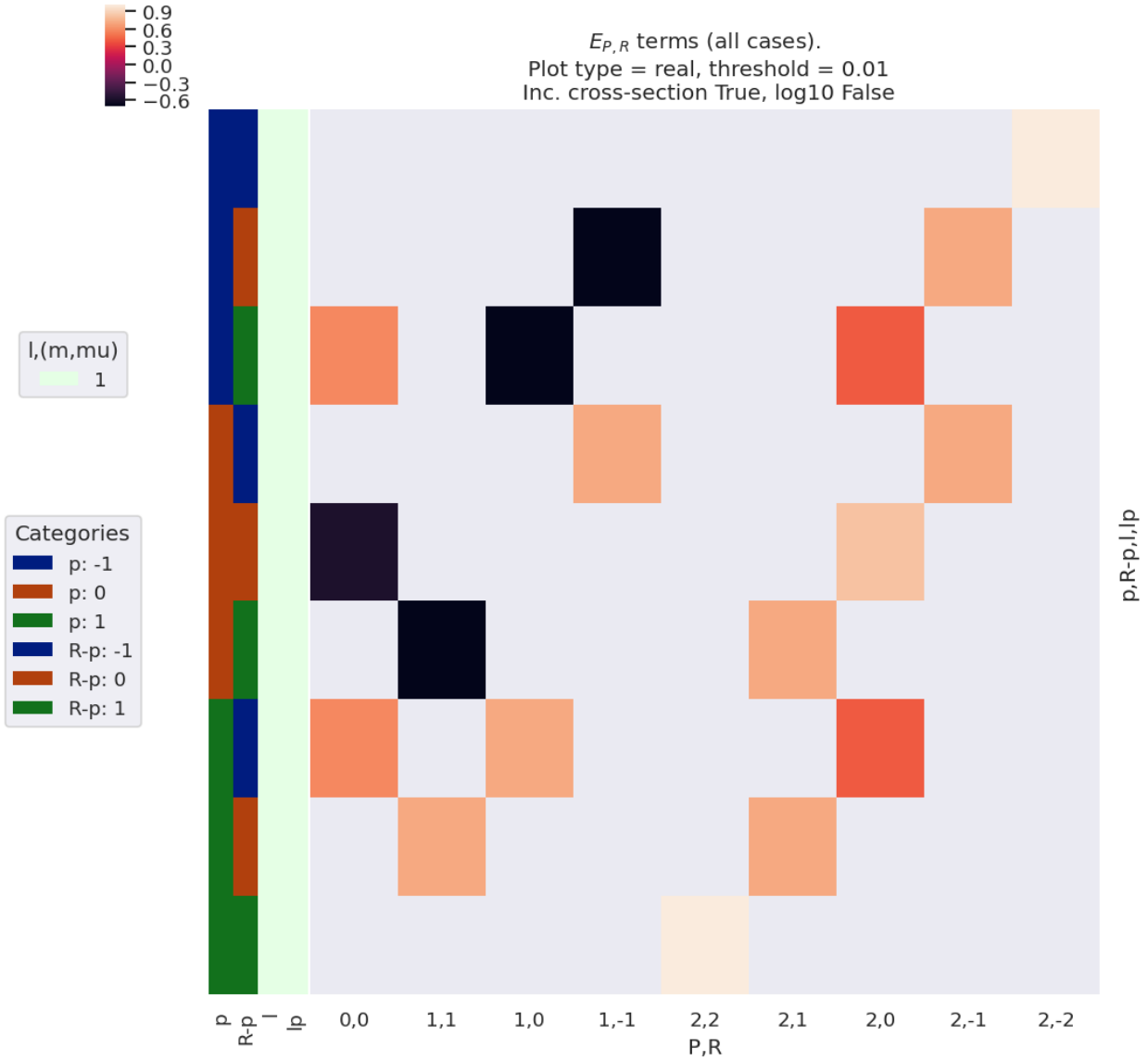


Fig. 7.4: Example $E_{P,R}$ basis functions. Note that for linearly polarised light $p = R - p = 0$ only, hence only the terms $E_{0,0}$ and $E_{2,0}$ are non-zero in this case. For non-linearly polarised cases many other terms are allowed.

7.3.7 Molecular frame projection term Λ

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

$$\Lambda_{R',R}(R_{\hat{n}}) = (-1)^{(R')} \begin{pmatrix} 1 & 1 & P \\ \mu & -\mu' & R' \end{pmatrix} D^{P}_{-R',-R}(R_{\hat{n}}) \tag{7.25}$$

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* $D^{P}_{-R',-R}(R_{\hat{n}})$.

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

$$\bar{\Lambda}_{R'} = (-1)^{(R')} \begin{pmatrix} 1 & 1 & P \\ \mu & -\mu' & R' \end{pmatrix} \equiv \Lambda_{R',R'}(R_{\hat{n}} = 0) \tag{7.26}$$

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 [3] 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. 7.3.3 for further discussion.)

7.3.8 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. 7.5 for details) and the photoionization multipole terms into the final observable.

$$\Delta_{L,M}(K, Q, S) = (2K+1)^{1/2}(-1)^{K+Q} \begin{pmatrix} P & K & L \\ R & -Q & -M \end{pmatrix} \begin{pmatrix} P & K & L \\ R' & -S & S-R' \end{pmatrix} \tag{7.27}$$

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

$$\tilde{\Delta}_{L,M}(t) = \sum_{K,Q,S} \Delta_{L,M}(K, Q, S) A^{K}_{Q,S}(t) \tag{7.28}$$

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. [2, 87]); 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 examplar cases.

Basic cases

Fig. 7.5 illustrates the alignment tensor $\Delta_{L,M}(K, Q, S)$ for some basic cases, and values are also tabulated in Table 7.6. 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. 7.3.3). For elliptically polarized light, mixing of terms with different p allows for non-zero R terms (see Eq. (7.22)), hence non-zero M is allowed (see Eq. (7.27)), 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 = 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={'lDims':['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)

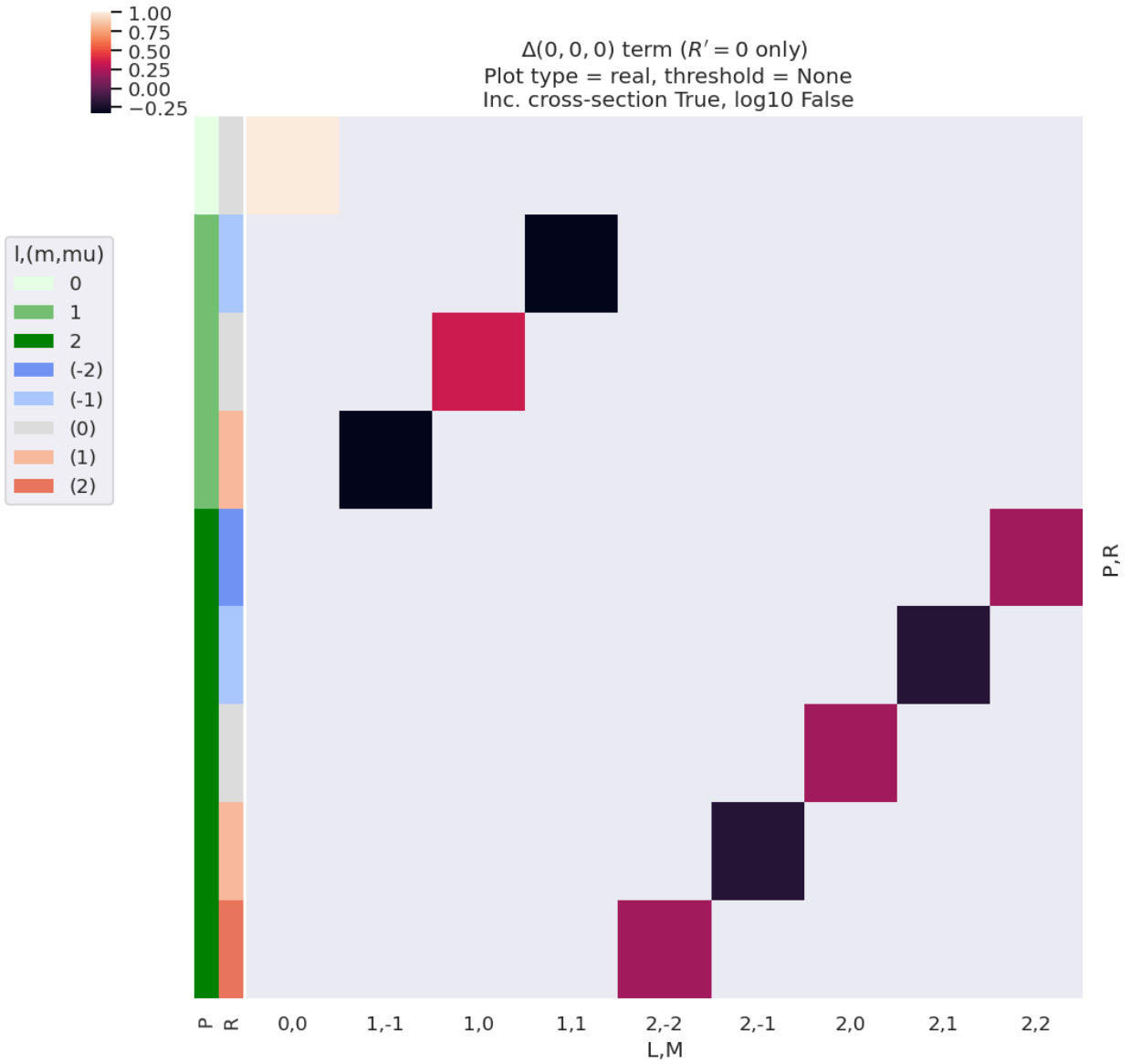


Fig. 7.5: Example $\Delta_{L,M}(0,0,0)$ basis functions (see also Fig. 7.6). For illustration purposes, the plot only shows terms for $R' = 0$. See main text for discussion.

	L	0	1			2				
	M	0	-1	0	1	-2	-1	0	1	2
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. 7.6: Example $\Delta_{L,M}(0,0,0)$ basis functions (see also Fig. 7.5). 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", ADMFig.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 anticipted 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 [5], and in the current case the focus is purely on rotational wavepackets.

Fig. 7.8 shows $\tilde{\Delta}_{L,M}(t)$ for various choices of alignment (as per the *ADMs* shown in Fig. 7.7), 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 ($Q = S = 0$).

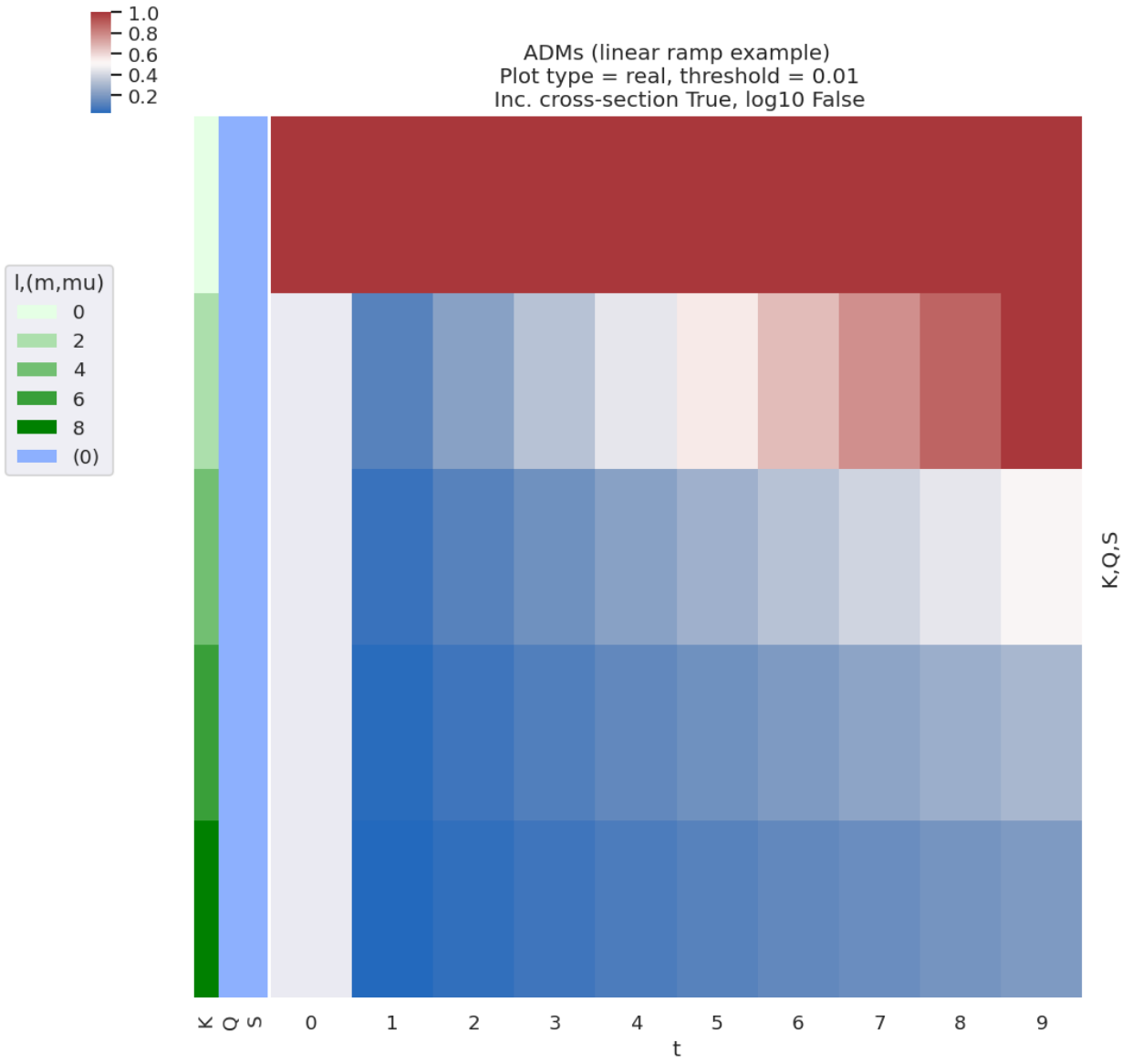


Fig. 7.7: Example ADMs used for *AF* basis function example (see Fig. 7.8). These ADMs essentially show an increasing degree of alignment with the t parameter, with high-order terms increasing at later t.

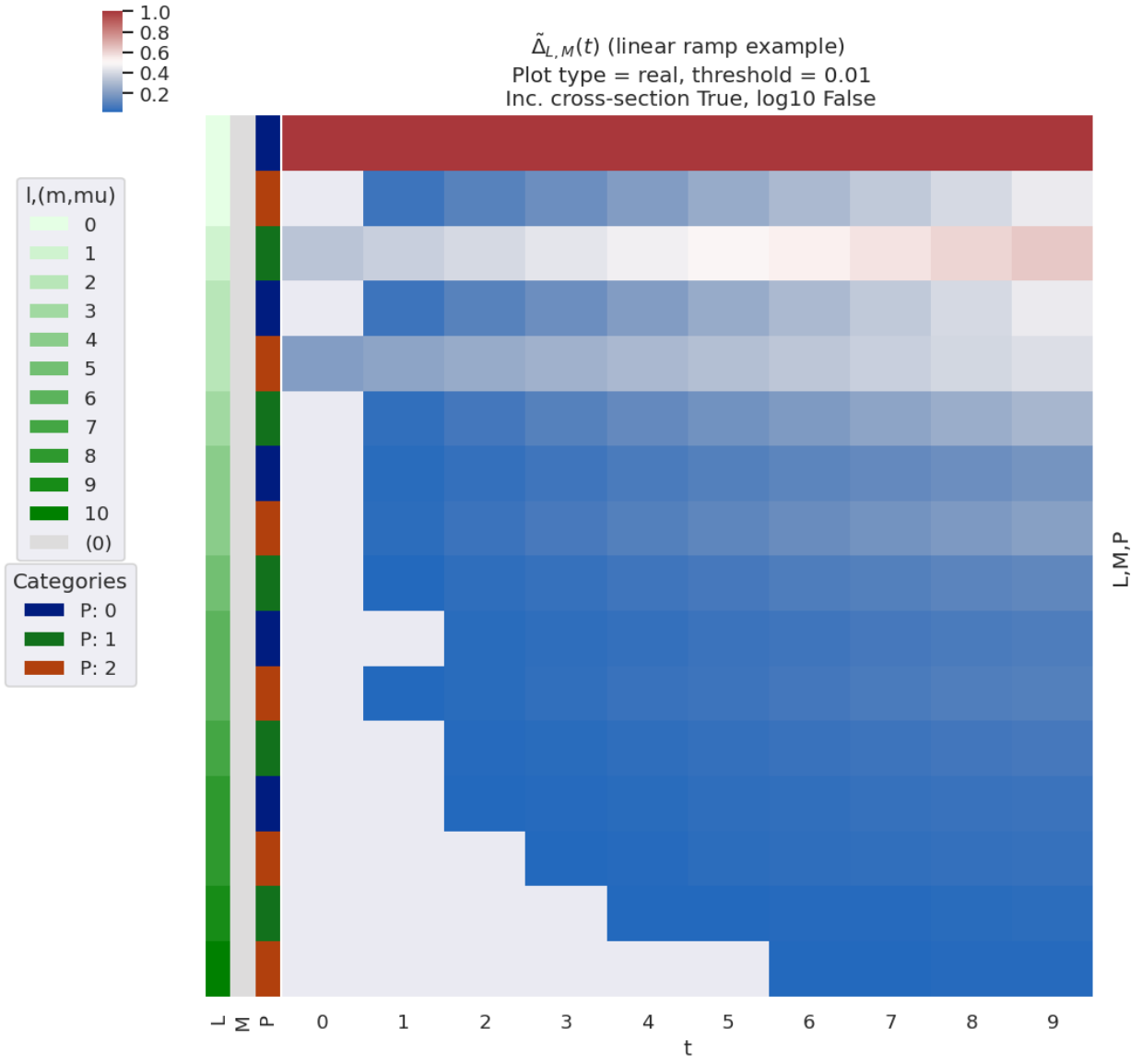


Fig. 7.8: Example of $\tilde{\Delta}_{L,M}(t)$ basis values for various choices of alignment (as per Fig. 7.7). The ADMs essentially show an increasing degree of alignment with the t parameter, with high-order terms increasing at later t, and this is reflected in the $\tilde{\Delta}_{L,M}(t)$ terms with 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. (7.28)). (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. 7.9 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).

7.3.9 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 (7.13)) for a given case. Numerically these are all implemented in the main ePSproc codebase [24, 25, 26], 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` [44, 45] data objects (for more details of the data structures used, see the ePSproc documentation [26], specifically the data structures page).

Polarisation & ADM product term: the main product basis returned, labelled `polProd` in the output dictionary, contains the tensor product $\Lambda_R \otimes E_{PR}(\hat{e}) \otimes \Delta_{L,M}(K,Q,S) \otimes A^K_{Q,S}(t)$, expanded over all quantum numbers (see full definition here). This term, therefore, incorporates all of the dependence (or response) of the AF-β_{LM}s on the polarisation state, and the axis distribution. Example results, making use of the linear-ramp *ADMs* of Sect. 7.3.8 are illustrated in Fig. 7.10.

The full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ as defined in (7.18) and (7.19) can be determined by the product of this

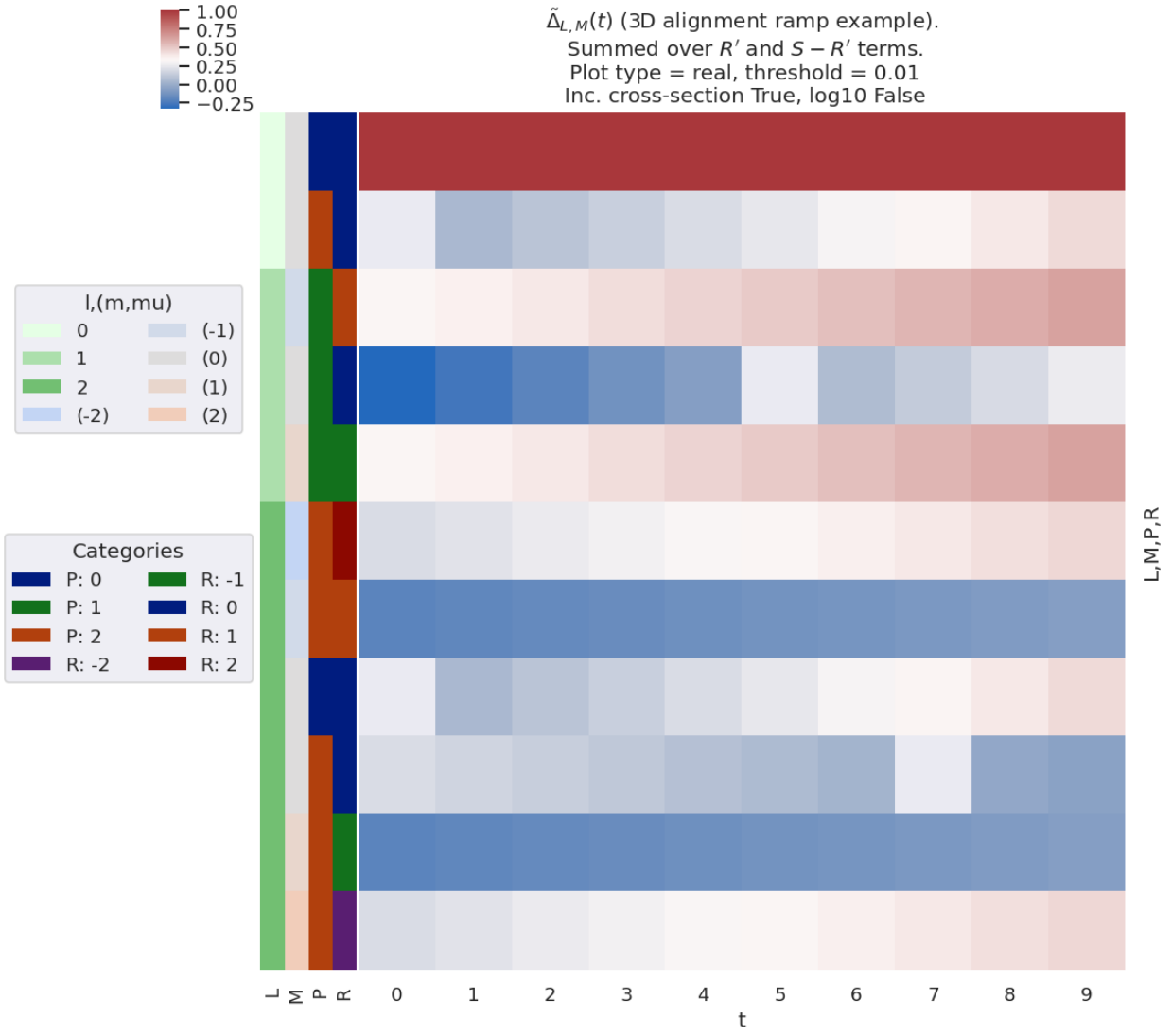


Fig. 7.9: Example $\tilde{\Delta}_{L,M}(t)$ basis values for various choices of "3D" alignment, i.e. including some $K \neq 0$ and $S \neq 0$ terms. Note, in particular, the presence of $M \neq 0$ terms in general, and a complicated dependence of the allowed terms on the alignment (*ADMs*), which may 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 *AF*-β_{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. 7.3.8 are illustrated in Fig. 7.11.

```
# 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)
```

7.4 Density matrix representation

7.4.1 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* [88], 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 [88]:

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

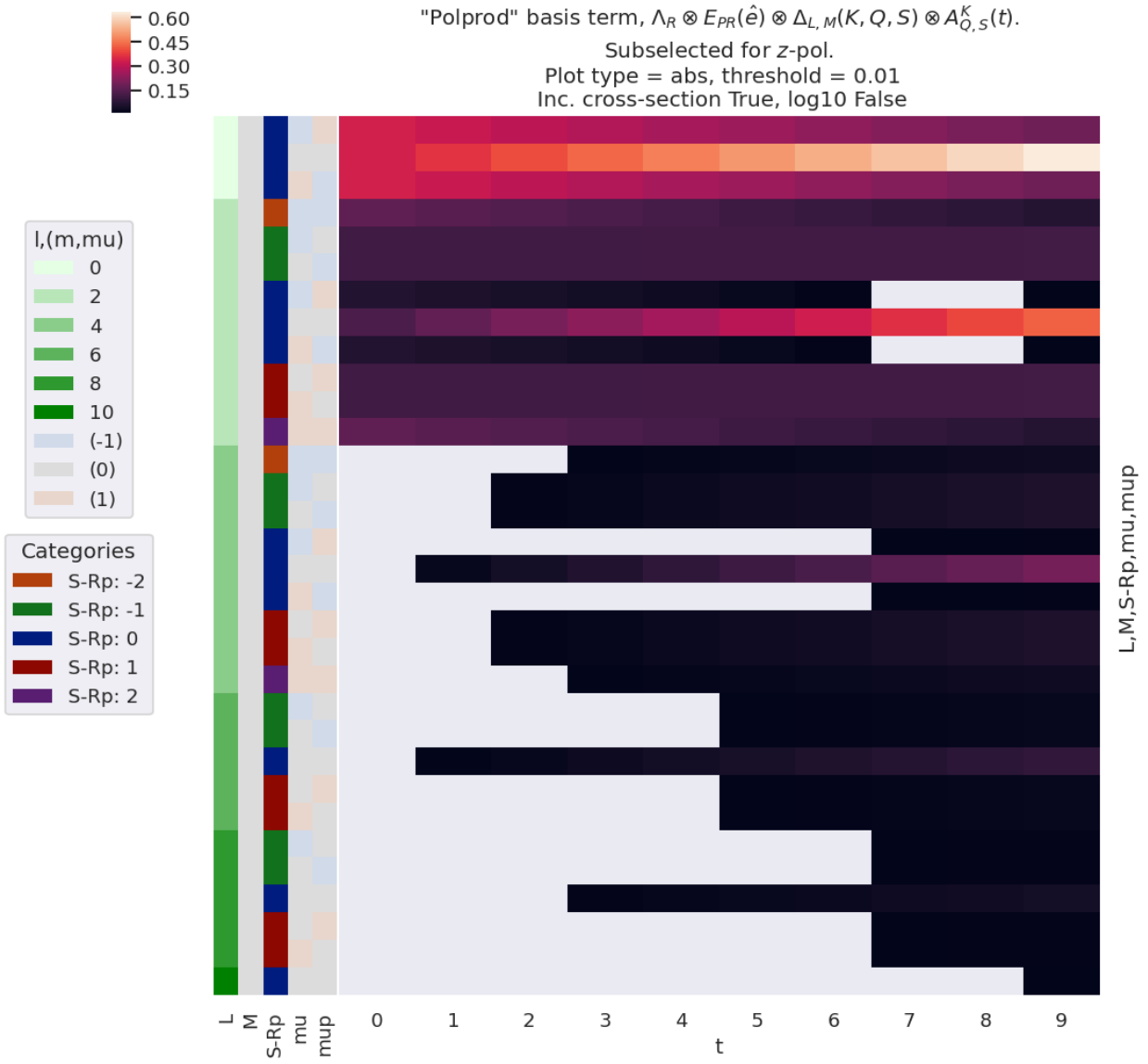


Fig. 7.10: Example product basis function for the polarisation and ADM terms, as given by $\Lambda_R \otimes E_{PR}(\hat{e}) \otimes \Delta_{L,M}(K,Q,S) \otimes A^K_{Q,S}(t)$. Shown for z-pol case only ($p = 0$).

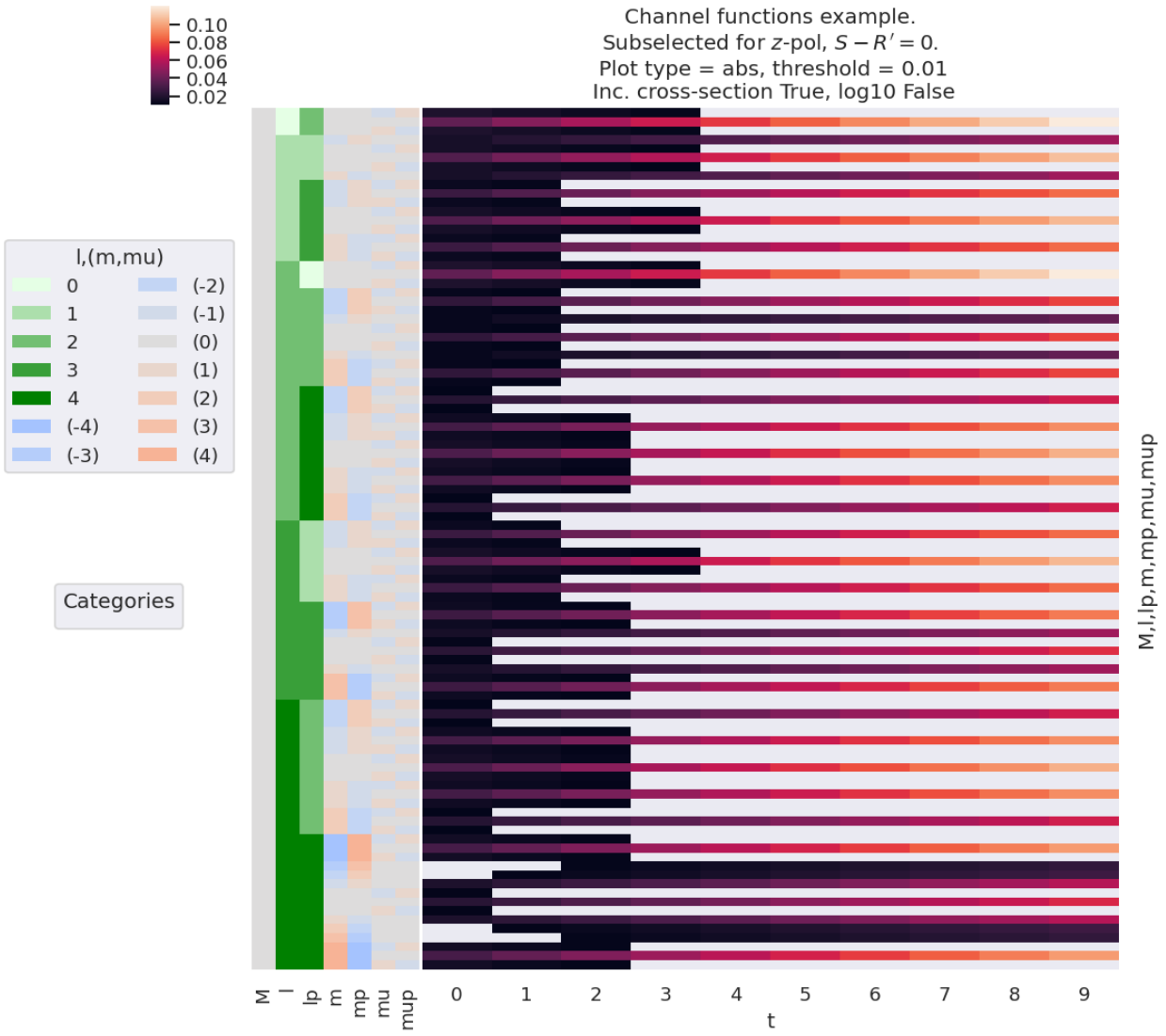


Fig. 7.11: Example of $\bar{\Upsilon}^{u,\zeta\zeta'}_{L,M}$ basis values for various choices of alignment (as per Fig. 7.7), shown for $L = 2$ and z-pol case only ($p = 0$). The basis essentially shows the obsevable terms if the ionization matrix elements are neglected, hence the sensitivity of the 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 [88]:

$$|\psi_n\rangle = \sum_{m'} a_{m'}^{(n)} |\phi_{m'}\rangle$$

$$\hat{\rho} = \sum_{n} \sum_{mm'} W_n a_{m'}^{(n)} a_{m}^{(n)*} |\phi_{m'}\rangle\langle\phi_m| \tag{7.29}$$

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

$$\rho_{i,j} = \langle\phi_i|\hat{\rho}|\phi_j\rangle = \sum_{n} W_n a_i^{(n)} a_j^{(n)*} \tag{7.30}$$

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.

7.4.2 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. (7.1)) 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 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. 7.3, 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:

$$\rho_{L,M}^{u,\zeta\zeta'} = \Upsilon_{L,M}^{u,\zeta\zeta'} \mathbb{I}^{\zeta,\zeta'} \tag{7.31}$$

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. (7.14)):

$$\rho^{\zeta\zeta'} = \mathbb{I}^{\zeta,\zeta'} \tag{7.32}$$

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. (7.31) includes any additional effects incorporated via the channel functions. For reconstruction problems, it is usually the reduced form of Eq. (7.32) 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. (7.31)) 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. (7.13)):

$$\beta_{L,M}^{u} = \sum_{\zeta,\zeta'} \rho_{L,M}^{u,\zeta\zeta'} \tag{7.33}$$

In fact, this form arises naturally since the $\beta^{u}_{L,M}$ 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. [88], Chpt. 4.).

In a more traditional notation (following Eq. (7.1), see also Ref. [90]), the density operator can be expressed as:

$$\rho(t) = \sum_{LM} \sum_{KQS} A^{K}_{QS}(t) \sum_{\zeta\zeta'} \Upsilon^{u,\zeta\zeta'}_{L,M} |\zeta, \Psi_{+}\rangle\langle\zeta, \Psi_{+}|\mu_q \rho_i \mu^{*}_{q'}|\zeta', \Psi_{+}\rangle\langle\zeta', \Psi_{+}| \tag{7.34}$$

This is, effectively, equivalent to an expansion in the various tensor operators defined in the channel function notation above (Eq. (7.31)), 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 $A^{K}_{Q,S}(t)$ term. (For further discussion of the use of density matrices in other specific cases, see *Quantum Metrology* Vol. 1 [2], 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 [26], as well as in the literature, see, e.g., Ref. [88] for general discussion, Ref. [75] 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 [91]. 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. [88], 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. [92, 93] for recent discussions in this context). A brief numerical diversion in this direction is given in Sect. 7.4.6, which illustrates the use of the the `QuTiP` (*Quantum Toolkbox in Python*) library [94, 95, 96] with the density matrix results derived herein.

7.4.3 Numerical setup

This follows the setup in Sect. 7.3 *Tensor formulation of photoionization*, using a symmetry-based set of basis functions for demonstration purposes. (Repeated code is hidden in PDF version.)

```
# 19/07/23 - this needs debugging, but skipped for now!
# Not sure what has changed - might be issue with dim names?
# ep.geomFunc.afblmXprod(data.data[data.subKey]['matE'], basisReturn = 'Full',␣
↪selDims={}, sqThres=False)
```

```
# Now run in script above

# # 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.
```

(continues on next page)

(continued from previous page)

```
# # TODO: consider different labelling here, can set at init e.g. dims = ['C', 'h',
 ↪'muX', 'l', 'm'] - 25/11/22 code currently fails for mu mapping, remap below instead
# symObj = symHarm(sym,lmax)
# # symObj = symHarm(sym,lmax,dims = ['Cont', 'h', 'muX', 'l', 'm'])

# # 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
# dimMap = {'C':'Cont','mu':'muX'}
# symObj.toePSproc(dimMap=dimMap)

# # Run conversion with a different dimMap & dataType
# dataType = 'matE'
# # symObj.toePSproc(dimMap = {'C':'Cont','h':'it', 'mu':'muX'}, dataType=dataType)
# symObj.toePSproc(dimMap = dimMap, dataType=dataType)
# # symObj.toePSproc(dimMap = {'C':'Cont','h':'it'}, dataType=dataType)    # Drop mu >␣
 ↪muX mapping for now
# # symObj.coeffs[dataType]

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

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

# for dataType in ['matE','BLM']:
#     data.data[dataKey][dataType] = symObj.coeffs[dataType]['b (comp)'].sum(['h','muX
 ↪'])  # Select expansion in complex harmonics, and sum redundant dims
#     data.data[dataKey][dataType].attrs = symObj.coeffs[dataType].attrs
```

```
# # Compute basis functions for given matrix elements

# # Set data
# data.subKey = dataKey

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

# # Using ePSproc directly - this includes full basis return if specified
# BetaNormX2, basisFull = ep.geomFunc.afblmXprod(data.data[data.subKey]['matE'],␣
 ↪basisReturn = 'Full', selDims={}, sqThres=False)  #, BLMRenorm = BLMRenorm,␣
 ↪**kwargs)

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

7.4.4 Compute a density matrix

A basic density matrix computation routine is implemented in the ePSproc codebase [24, 25, 26]. 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. (7.29), (7.30), 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 defined matrix element is given below; for more examples see the ePSproc documentation [26].

```
# 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/matE.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:
```

(continues on next page)

(continued from previous page)

```
#   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)

# Renormlise output?
if normDen=='max':
    daOut = daOut/daOut.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)
```

7.4.5 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-denMatD2hCompExample`; 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-denMatD2hCompExample`(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.

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

if normME:
    matE = matE/matE.max()

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

# Compute density matrix
daOut_noise, *_ = density.densityCalc(matE_noise, denDims = denDims, selDims =␣
 ↪selDims, thres = thres)

# Renormlise output?
if normDen=='max':
    daOut_noise = daOut_noise/daOut_noise.max()
elif normDen=='trace':
    daOut_noise = daOut_noise/(daOut_noise.sum('Sym').pipe(np.trace)**2)
```

(continues on next page)

(continued from previous page)

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

# 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}')

#*** 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'"))
```

7.4.6 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 Toolkbox in Python*) library [94, 95, 96] 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 [96] for more possibilities.

Convert numerical arrays to QuTiP objects

Fidelity metric

Fidelity between two density matrices ρ_a, ρ_b can be defined as per Refs. [97, 98]:

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

This is implemented by the `fidelity` function in the `QuTiP` (*Quantum Toolkbox in Python*) library [94, 95, 96]. 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-denMatD2hCompExample`(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()}")
```

```
# 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())}")
```

```
# 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()
```

7.5 Molecular alignment

7.5.1 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. 5.1). 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. 7.3. 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. [99, 100, 101, 102] for reviews and further introductory materials, and further discussion in the current context can be found in *Quantum Metrology* Vols. 1 & 2 [2, 5] and Refs. [86, 87, 103, 104, 105] 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. 7.3.8 and Sect. 7.7). 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. 8.1.1.

7.5.2 Alignment distribution moments (ADMs)

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

$$P(\Omega,t)=\sum_{K,Q,S}A^{K}_{Q,S}(t)D^{K}_{Q,S}(\Omega) \tag{7.35}$$

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

$$P(\theta,\phi,t)=\sum_{K,Q}A^{K}_{Q,0}(t)D^{K}_{Q,0}(\Omega)=\sum_{K,Q}A^{K}_{Q}(t)Y_{K,Q}(\Omega) \tag{7.36}$$

In the examples given in Sect. 7.3, some arbitrary choices of $A^K_{Q,S}(t)$ 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 $A^K_{Q,S}(t)$ 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 $A^K_{Q,S}(t)$, see Ref. [] for further introductory discussion on this point.) Note that, as discussed in Sect. 6.5, computation of molecular alignment is not yet implemented in the Photoelectron Metrology Toolkit [3], so values must be obtained from other codes. *ADMs* used herein were all computed with codes developed by V. Makhija [106], and are available from the ePSproc [25] repo on Github.

7.5.3 Numerical setup

For illustrative purposes, the *ADMs* used for the OCS fitting example are here loaded and used to compute $P(\Omega, t)$.

```
# 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)
data.data['subset']['ADM'].unstack().where(data.data['subset']['ADM'].unstack().K>0) \
        .real.hvplot.line(x='t').overlay(['K','Q','S'])
```

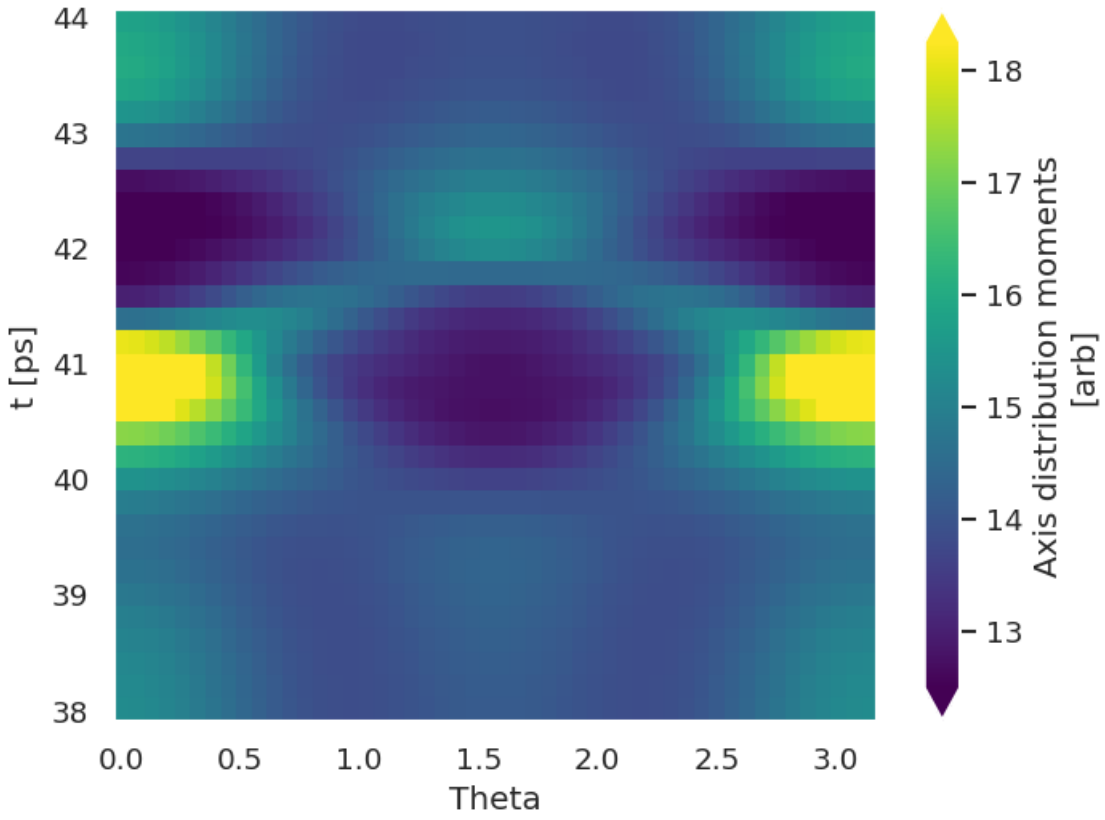
```
:NdOverlay   [S,Q,K]
   :Curve   [t]   (ADM)
```

7.5.4 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 [3] 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.

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

data.padPlot(keys = dataKey, dataType='ADM', Etype = 't', pType='a',
             returnFlag = True, selDims={'t':tPlot}, backend='pl')
```

```
Using default sph betas.
Summing over dims: set()
Plotting from self.data[subset][ADM], facetDims=['t', None], pType=a with␣
 ↪backend=pl.
Set plot to self.data['subset']['plots']['ADM']['polar']
```

7.6 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. 5.1) - 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.

7.6.1 Spherical harmonics

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

$$\bar{I}(\epsilon, t, \theta, \phi) = \sum_{L=0}^{2n} \sum_{M=-L}^{L} \bar{\beta}_{L,M}(\epsilon, t) Y_{L,M}(\theta, \phi) \tag{7.37}$$

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 $\bar{\beta}_{L,M}(\epsilon, t)$ are generically termed *anisotropy paramters*. 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. 5.1. Some arbitrary examples are given in Fig. 7.12, 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. 7.13.

Numerically, there are some choices and conventions which apply to the spherical harmonics. As noted in Sect. 6.3: "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 [50]." For further details, including conversion routines, see the `pySHtools` [51, 52, 53, 54] 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)]
# 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()
```

```
# 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

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

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

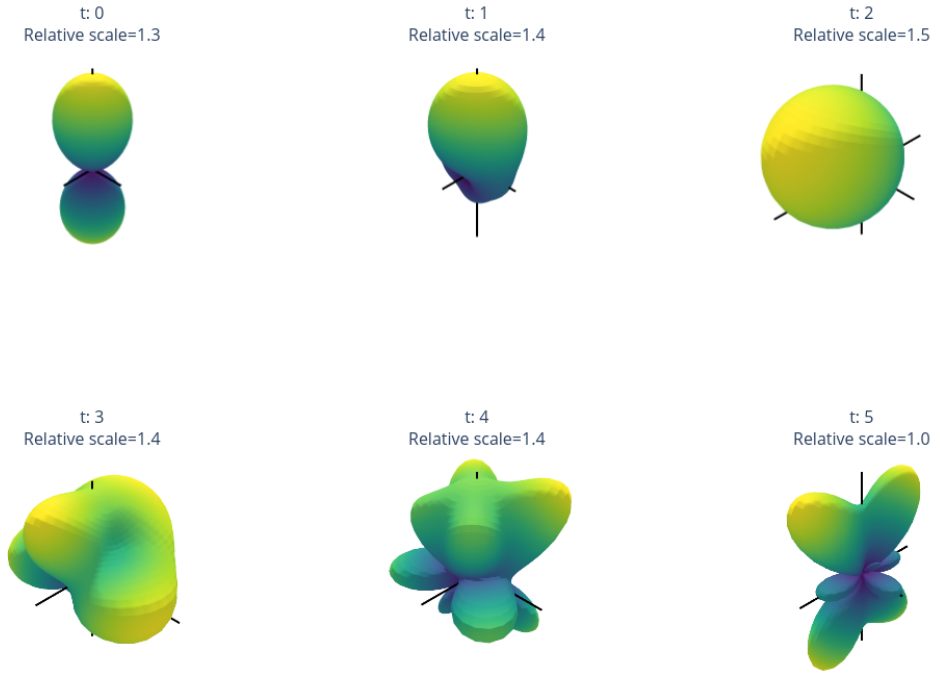


Fig. 7.12: Examples of angular distributions (expansions in spherical harmonics $Y_{L,M}$), for a range of cases indexed by t. Note that up-down asymmetry is associated with odd-l contributions (e.g. $t = 1, 2$), breaking of cylindrical symmetry with $m \neq 0$ terms (all $t > 0$), and asymmetries in 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 ($t = 3, 4, 5$).

In general, the spherical harmonic rank and order (L, M) of Eq. (7.37) 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 [107]), but in the *MF* is defined by the maximum continuum angular momentum $n = l_{max}$ imparted by the scattering event [108] (note lower-case l here refers specifically to the continuum photoelectron wavefunction, see Eq. (7.5)).

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

Fig. 7.13: Values used for the plots in Fig. 7.12.

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. [2] for more discussion and complex examples.

In general, these observables may also be dependent on various other parameters; in Eq. (7.37) 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. 7.1, the origin of such dependencies 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. 7.12.

7.6.2 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 [109, 110, 111]:

$$X_{hl}^{\Gamma\mu*}(\theta,\phi) = \sum_{\lambda} b_{hl\lambda}^{\Gamma\mu} Y_{l,\lambda}(\theta,\phi) \tag{7.37}$$

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. 7.3.2 - 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. (7.37), a general expansion of an observable in the symmetrized harmonic basis set can then be defined as:

$$\bar{I}^{\Gamma}(\epsilon,t,\theta,\phi) = \sum_{\Gamma\mu hl} \bar{\beta}_{hl}^{\Gamma\mu}(\epsilon,t) X_{hl}^{\Gamma\mu*}(\theta,\phi) \tag{7.38}$$

Alternatively, by substitution into Eq. (7.37), and assigning $l = L$ and $\lambda = M$, a general symmetrized expansion may also be defined as:

$$\bar{I}(\epsilon,t,\theta,\phi) = \sum_{\Gamma\mu h}\sum_{L=0}^{2n}\sum_{M=-L}^{L} b_{hLM}^{\Gamma\mu}\bar{\beta}_{L,M}(\epsilon,t)Y_{L,M}(\theta,\phi) \tag{7.39}$$

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:

$$\bar{I}^{\Gamma}(\epsilon,t,\theta,\phi) = \sum_{L=0}^{2n}\sum_{M=-L}^{L} \bar{\beta}_{L,M}^{\Gamma}(\epsilon,t)Y_{L,M}(\theta,\phi) \tag{7.39}$$

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. (7.37) (e.g. for cases with degenerate components). Further details and usage in relation to channel functions are also discussed in Sect. 7.3 (see, in particular, Eq. (7.13) 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. [111, 112]. Numerical routines for the generation of symmetrized harmonics are implemented in Photoelectron Metrology Toolkit [3]: point-groups, character table generation and symmetrization (computing $b_{hl\lambda}^{\Gamma\mu}$ parameters) is handled by `libmsym` [55, 56]; additional handling also makes use of `pySHtools` [51, 52, 53, 54].

A brief numerical example is given below, and more details can be found in the PEMtk documentation [14]. In this case, full tabulations of the parameters list all $b_{hLM}^{\Gamma\mu}$ for each irreducible representation, and the corresponding PADs are illustrated in Fig. 7.15.

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()
```

```
# 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...
```

(continues on next page)

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. 7.14: Example character table for Td symmetry generated with the Photoelectron Metrology Toolkit [3] wrapper for `libmsym` [55, 56].

(continued from previous page)

```
# 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
    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]  # Explict 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)
```

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

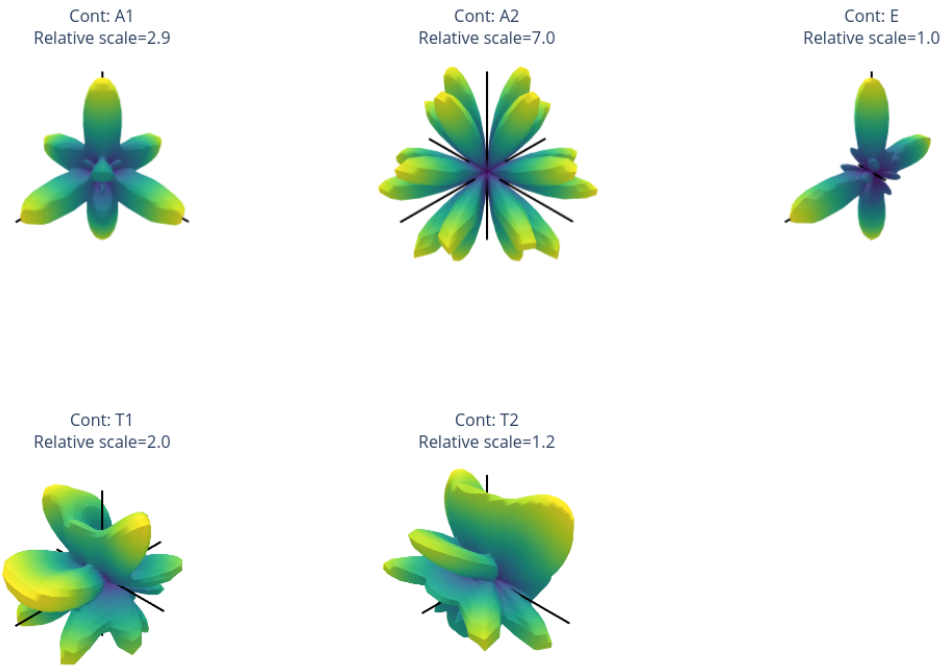


Fig. 7.15: Examples of angular distributions from expansions in symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta,\phi)$, for all irreducible representations in Td symmetry (l_{max} =6). (Note A_2 only has components for $l \geq 6$.)

can be defined as (per the Wikipaedia definitions [113]):

$$
\begin{aligned}
Y_{\ell m} &= \begin{cases} \frac{i}{\sqrt{2}}\left(Y_\ell^m - (-1)^m Y_\ell^{-m}\right) & \text{if } m0 \\ Y_\ell^0 & \text{if } m=0 \\ \frac{1}{\sqrt{2}}\left(Y_\ell^{-m} + (-1)^m Y_\ell^m\right) & \text{if } m>0. \end{cases} \\
&= \begin{cases} \frac{i}{\sqrt{2}}\left(Y_\ell^{-|m|} - (-1)^m Y_\ell^{|m|}\right) & \text{if } m0 \\ Y_\ell^0 & \text{if } m=0 \\ \frac{1}{\sqrt{2}}\left(Y_\ell^{-|m|} + (-1)^m Y_\ell^{|m|}\right) & \text{if } m>0. \end{cases} \\
&= \begin{cases} \sqrt{2}\,(-1)^m\,\Im[Y_\ell^{|m|}] & \text{if } m0 \\ Y_\ell^0 & \text{if } m=0 \\ \sqrt{2}\,(-1)^m\,\Re[Y_\ell^m] & \text{if } m>0. \end{cases}
\end{aligned}
\tag{7.39}
$$

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 [24, 25, 26] either directly or via the `pySHtools` [51, 52, 53, 54] 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 [14] for further details and examples, and the "Working with real spherical harmonics" note from the ePSproc documentation [26], 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'
csphase = -1
lmax = 6
error_kind = None
header = None
header2 = None
name = None
units = None
```

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

7.6.3 Legendre polynomials

Finally, it is of note that Legendre polynomial expansions are also in common use in the description of photoionization obsevables. These are suitable for cylindrically symmetric cases only, and form a subset of the general spherical harmonic case. Using the same notation as Eq. (7.37), the 1D expansion can be given as:

$$\bar{I}(\epsilon,t,\theta) = \sum_{L=0}^{2n} \bar{\beta}_L(\epsilon,t) P_L(\cos(\theta)) \tag{7.40}$$

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:

$$\beta_{L,0}^{Sph} = \sqrt{(2L+1)/4\pi}\beta_L^{Lg} \tag{7.41}$$

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

Herein only spherical harmonic expansions are used, but the ePSproc codebase [24, 25, 26] does include a conversion routine to convert expansion parameters as required. Again more information can be found in the ePSproc documentation [26], 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
```

7.7 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. 7.3, 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. 7.3.9, including the full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ ((7.18) and (7.19)), 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 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. [103, 114]).

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.

7.7.1 Numerical setup

This follows the setup in Sect. 7.3 *Tensor formulation of photoionization*, using a symmetry-based set of basis functions for demonstration purposes. (Repeated code is hidden in PDF version.)

7.7.2 Experimental information content

As discussed in *Quantum Metrology* Vol. 2 [5], the information content of a single observable might be regarded as simply the number of contributing $\beta_{L,M}$ parameters. In set notation:

$$M = \mathrm{n}\{\beta_{L,M}\} \tag{7.42}$$

where M is the information content of the measurement, defined as $\mathrm{n}\{...\}$ 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:

$$M_u = \sum_u \mathrm{n}\{\beta^u_{L,M}\}$$

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:

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

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 $\{\beta^u_{L,M}(k,t)\}$. 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. (7.13)), 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 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', 'BLM')
```

```
<xarray.DataArray 'BLM' (BLM: 3)>
array([(0, 0), (1, 0), (2, 0)], dtype=object)
Coordinates:
  * BLM      (BLM) MultiIndex
  - l        (BLM) int64 0 1 2
  - m        (BLM) int64 0 0 0
```

```
# 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
# BetaNorm.BLM.where(np.abs(BetaNorm.BLM.m)<=BetaNorm.BLM.l,drop=True)
# BetaNorm.where(np.abs(BetaNorm.m)<=BetaNorm.l,drop=True)

cleanBLMs(BetaNorm).BLM
```

```
<xarray.DataArray 'BLM' (BLM: 3)>
array([(0, 0), (1, 0), (2, 0)], dtype=object)
Coordinates:
  * BLM      (BLM) MultiIndex
  - l        (BLM) int64 0 1 2
  - m        (BLM) int64 0 0 0
```

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

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

```
# 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={BetaNorm.BLM.where(np.abs(BetaNorm.BLM.m)<=BetaNorm.BLM.l,drop=True).size}")
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=3
Cleaned array M=3
Thresholded array (thres=0.0001), M=0
```

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()   #([pd.unique])   #(['nunique'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[10], line 7
      3 BetaNormLinearADMsPD, _ = ep.util.multiDimXrToPD(BetaNormLinearADMs.
 ↪squeeze().real, thres=1e-4, colDims='t')
      5 # Basic describe with Pandas, see https://pandas.pydata.org/docs/user_
 ↪guide/basics.html#summarizing-data-describe
      6 # This will give properties per t
----> 7 BetaNormLinearADMsPD.describe()   #([pd.unique])   #(['nunique'])

File /opt/conda/lib/python3.10/site-packages/pandas/core/generic.py:10940, in␣
 ↪NDFrame.describe(self, percentiles, include, exclude, datetime_is_numeric)
  10691 @final
  10692 def describe(
  10693     self: NDFrameT,
   (...)
  10697     datetime_is_numeric: bool_t = False,
  10698 ) -> NDFrameT:
  10699     """
  10700     Generate descriptive statistics.
  10701
   (...)
  10938     max            NaN      3.0
  10939     """
> 10940     return describe_ndframe(
  10941         obj=self,
  10942         include=include,
  10943         exclude=exclude,
  10944         datetime_is_numeric=datetime_is_numeric,
  10945         percentiles=percentiles,
  10946     )

File /opt/conda/lib/python3.10/site-packages/pandas/core/describe.py:94, in␣
 ↪describe_ndframe(obj, include, exclude, datetime_is_numeric, percentiles)
     89     describer = SeriesDescriber(
     90         obj=cast("Series", obj),
     91         datetime_is_numeric=datetime_is_numeric,
     92     )
     93 else:
---> 94     describer = DataFrameDescriber(
     95         obj=cast("DataFrame", obj),
     96         include=include,
     97         exclude=exclude,
     98         datetime_is_numeric=datetime_is_numeric,
     99     )
```

(continues on next page)

(continued from previous page)

```
    101 result = describer.describe(percentiles=percentiles)
    102 return cast(NDFrameT, result)

File /opt/conda/lib/python3.10/site-packages/pandas/core/describe.py:171, in␣
 ↪DataFrameDescriber.__init__(self, obj, include, exclude, datetime_is_numeric)
    168 self.exclude = exclude
    170 if obj.ndim == 2 and obj.columns.size == 0:
--> 171     raise ValueError("Cannot describe a DataFrame without columns")
    173 super().__init__(obj, datetime_is_numeric=datetime_is_numeric)

ValueError: Cannot describe a DataFrame without columns
```

```
# 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()
```

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 1 d.p., then apply statistical methods
BetaNormLinearADMsPD.round(1).agg(['min','max','var','count','nunique'])
```

```
# 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')  #.round(1).agg(['min','max',
 ↪'var','count','nunique'])   # OK, matches above case.
```

```
# Apply statistical functions to differences from mean.
BetaNormLinearADMsPD.transform(demean,axis='columns').round(1).agg(['min','max','var',
 ↪'count','nunique'])
```

In this case the analysis suggests that $t = 4, 5$ contain minimal, and redundant, information, whilst $t = 0, 3, 6$ are also of low total information content. 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 $K_{i,j} = \mathrm{cov}[X_i, X_j] = \langle (X_i - \langle X_i \rangle)(X_j - \langle X_j \rangle) \rangle$. For the linear ramp case this analysis is not particularly useful, but will become more informative for more complicated cases.

```
# import pandas as pd
# import holoviews as hv
# import hvplot.pandas
# hv.extension('bokeh')

# 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()
```

(continues on next page)

(continued from previous page)

```
# covMat.name = "Covariance"
# Plot with holoviews
# hv.HeatMap(covMat, kdims='t')
# import hvplot.pandas
figObj = covMat.hvplot.heatmap(cmap='viridis')

# hvds = hv.Dataset(daPlot)
# hvmap = hvds.to(hv.HeatMap, kdims=kdims)

# Import routines for density calculation and plotting - FAILS, needs da with␣
 ↪specific props
# from epsproc.calc import density
# density.matPlot(covMat)
```

7.7.3 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:

$$M_u = \mathrm{n}\{\Upsilon^u_{L,M}(\epsilon,t)\}$$

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. [115, 116, 117] for details.)

In the numerical examples below, this is considered in terms of the full channel (response) functions $\Upsilon^{u,\zeta\zeta'}_{L,M}$ as defined in (7.18) and (7.19) (see Sect. 7.3.9). 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}")
```

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')

# Basic describe with Pandas, see https://pandas.pydata.org/docs/user_guide/basics.
↪html#summarizing-data-describe
# This will give properties per t
# channelFuncsSubsetPD.describe()   #([pd.unique])   #(['nunique'])

# Round values to 1 d.p., then apply statistical methods
# channelFuncsSubsetPD.round(1).agg(['min','max','var','count','nunique'])  # Compute␣
↪per t

channelFuncsSubsetPD.T.round(2).agg(['min','max','var','count','nunique']).T
↪#[0:100]  # Compute per basis index and display


# Plotting tests with hvplot wrapper - interesting, but lacking fine control so far...
# channelFuncsSubsetPD.T.round(1).agg(['min','max','var','count','nunique']).T.hvplot.
↪hist('nunique', by=['L','M'], subplots=True).cols(1)
# channelFuncsSubsetPD.T.round(3).agg(['min','max','var','count','nunique']).T.hvplot.
↪hist('nunique', by=['l','lp'])
# channelFuncsSubsetPD.T.round(3).agg(['min','max','var','count','nunique']).T.hvplot.
↪hist('nunique', by=['l','lp'], subplots=True).cols(2)
# channelFuncsSubsetPD.T.round(2).agg(['min','max','var','count','nunique']).T.hvplot.
↪hist('nunique', by=['L','l','lp'], subplots=True)
# channelFuncsSubsetPD.T.round(2).agg(['min','max','var','count','nunique']).T.hvplot.
↪heatmap(x='nunique',y='L', by=['L','l','lp'], subplots=True)

# Holomap version - not working in notebook currently? Gives controls, but plot not␣
↪responding...
# channelFuncsSubsetPD.T.round(2).agg(['min','max','var','count','nunique']).T.hvplot.
↪hist('nunique', by=['L','l','lp'], groupby=['L','l','lp'])
```

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. 7.3.9, 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');
```

```
# Full covariance mapping along all dims
sns.clustermap(channelFuncsSubsetPD.T.cov().fillna(0))
```

7.7.4 Information content from density matrices

CHAPTER

EIGHT

NUMERICAL METHODOLOGIES FOR EXTRACTING MATRIX ELEMENTS

Following the tensor notation outline in Sect. 7.3, the complete quantum metrology of a photoionization event (aka. a "complete" photoionization experiment) can be characterized as recovery of the matrix elements $I^{\zeta}(\epsilon)$ (per Eqs. (7.13), (7.14)) from the experimental measurements or, equivalently, the density matrix $⍰^{\zeta\zeta'}$ (Eqs. (7.31) - (7.33)). (For further discussion and background, see Refs. [83, 85] and *Quantum Metrology* Vol. 1 [2].) This may be possible provided the channel functions are known, and the information content of the measurements is sufficient.

For schemes making use of molecular alignment, or other control methods, the contribution of these parameters to the channel functions must, therefore, be accounted for. In general, these contributions may be computed and/or obtained from experiment. For the rotational wavepacket case, this can be considered as the requirement for the determination of the molecular axis distributions (*ADMs*): this can be treated as a reduced-dimensionality MF signal retrieval problem, with sets of computed *ADMs* forming the basis set for the fitting, and this forms the first step in both the generalised bootstrapping method explored herein. (Note here that the 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 [5] for further discussion on this point.)

Of particular import for matrix element retrieval is the phase-sensitive nature of the observables (Sect. 7.6), which is required in order to obtain partial wave phase information. *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 [118], and general quantum tomography [91].

8.1 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. 8.1. Firstly, a linear fitting stage to retrieve the molecular axis distribution, characterised by a set of *ADMs* (see Sect. 7.3.8 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:

$$\bar{\beta}^{u}_{L,M}(\epsilon,t) = \sum_{K,Q,S} A^{K}_{Q,S}(t)\bar{C}^{LM}_{KQS}(\epsilon) \tag{8.1}$$

And the 2nd stage as per Eqs. (7.13) and (7.17) for the AF case:

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

In these forms, the terms are:

1. $A^{K}_{Q,S}(t)$, the set of *ADMs* defining the molecular alignment, and associated parameters $\bar{C}^{LM}_{KQS}(\epsilon)$.

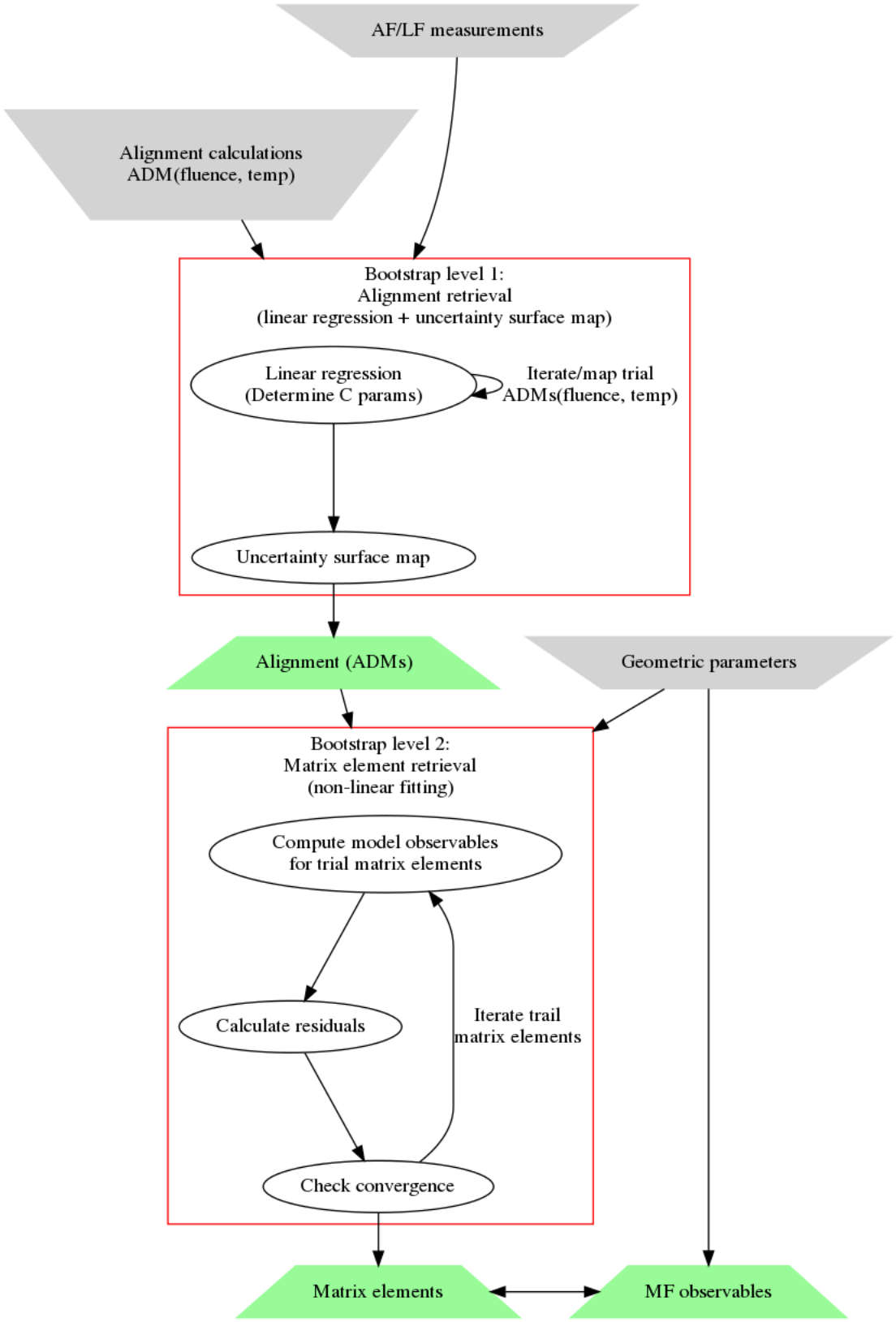


Fig. 8.1: Outline of the 2-stage generalised bootstrap matrix element retrieval protocol.

2. $\bar{\Upsilon}^{u,\zeta\zeta'}_{L,M}(t)$, the channel functions in the *AF* (Eq. (7.17), 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 $A^K_{Q,S}(t)$ 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. XX. 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. XX.

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 [2, 5]. Although only rotational wavepacket cases are illustrated in this work (see Chpt. 9), by suitable choice of dataset and channel functions many other experimental schemes may be modelled and analysed; the Photoelectron Metrology Toolkit [3] is designed with this flexibility in mind.

8.1.1 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 [REFS]. 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 molecular frame. Since in this case the time resolved measurement constitutes a set of measurements of the same quantity from a variety of molecular axes distributions, 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 molecular frame 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,

$$S(\theta,\chi) = \sum_{jk} C_{jk} D^j_{0k}(\theta,\chi), \tag{8.2}$$

where θ and χ are the MF spherical polar and azimuthal angles of the linearly polarized electric field vector generating the signal; C_{jk} are unknown expansion coefficients; and D^j_{0k} are the Wigner D-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,

$$\langle S\rangle(t) = \sum_{jk} C_{jk} \langle D^j_{0k}\rangle(t). \tag{8.3}$$

Since the rotational wavepacket can be accurately simulated, the $\langle D^j_{0k}\rangle(t)$ are considered known. The time resolved signal $\langle S\rangle(t)$ being measured, the unknown coefficients C_{jk} can be determined by linear regression, and the molecular frame signal in Eqn.~\ref{eq:mfrealsig} constructed. In this form the method was initially applied to strong field ionization and dubbed Orientation Reconstruction through Rotational Coherence Spectroscopy (ORRCS) [119, 120]. It has since been applied to strong field ionization of various molecules [121, 122, 123], strong field dissociation [124] and few-photon ionization [125].

(A large range of other experimental methods have also addressed alignment and orientation dependence and retrieval, other recent examples include Coulomb-explosion imaging [126], high-harmonic spectroscopy [127, 128], optical imaging [129] and rotational echo spectroscopy [130], see Refs. [101, 104] for further discussion.)

The case of PADs is a more challenging one, since they are not generally described by Eqn.~\ref{eq:mfrealsig}. Instead, both *AF* and *MF PADs* are determined by the radial dipole matrix elements. However, the correspondence of the problem with an equation of the form of Eqn. \ref{eq:St-Cjk} - essentially a convolution - can be made. This is discussed in detail

in Ref. [87]. In the current case Eqs. (7.13) and (7.17) can be rewritten in a similar form to Eq. (8.3) by explicitly separating out the axis distribution moments $A^K_{Q,S}(t)$ and collapsing all other terms. The case of photoionization from a time-dependent ensemble can then be reparameterized as indicated in Eq. (8.1).

Here the set of axis distribution moments can thus be viewed as modulating all observables $\beta^u_{L,M}(t)$. The unknowns, $\bar{C}^{LM}_{KQS}$ and axis distribution moments $A^K_{Q,S}(t)$, can be retrieved in a similar manner to that discussed for the simpler scalar observable case above, i.e. via linear regression with simulated rotational wavepackets.

In practice this equates to (accurately) simulating rotational wavepackets, hence obtaining the corresponding $A^K_{Q,S}(t)$ 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 $A^K_{Q,S}(t)$. 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 $A^K_{Q,S}(t)$) and $\bar{C}^{LM}_{KQS}(\epsilon)$. 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 $\bar{C}^{LM}_{KQS}$, but details of the matrix elements are not obtained directly.

```
# Numerical examples of ADMs and rotational wavepackets here? Some also now in␣
↪fitting basis set chpt.
```

8.1.2 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:

$$\chi^2(\mathbb{I}^{\zeta\zeta'}) = \sum_u \left[\beta^u_{L,M}(\epsilon,t;\mathbb{I}^{\zeta\zeta'}) - \beta^u_{L,M}(\epsilon,t)\right]^2 \tag{8.4}$$

where $\beta^u_{L,M}(\epsilon,t;\mathbb{I}^{\zeta\zeta'})$ denotes the values from a model function, computed for a given set of (complex) matrix elements $\mathbb{I}^{\zeta\zeta'}$, and $\beta^u_{L,M}(\epsilon,t)$ the experimentally-measured parameters, for a given configuration u. Implicit in the notation is that 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. (7.15); see also discussion in Sect. 7.3.1)

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, and how well they will scale to larger problems (more matrix elements/partial waves). Exploration of these questions for various exemplar systems is the topic of the latter part of this book (see Chpt. 9).

```
# Numerical example here? May skip this, since it's in latter chpts.
```

8.1.3 Adaptation and fitting for different reconstruction protocols

8.2 Fitting strategies

Notes on stats, repeated fitting etc. here? Now illustrated in Part 2, but no formalism given. See also QM1/2…?

Basis set choices are also introduced later.

Part III

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

CHAPTER

NINE

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 HTML or source notebooks for readers that wish to perform their own calculations.

The layout for this part is as follows:

- Technical chapters
  - Chapter 10: *Basis sets for fitting*: introduces methods for setting the basis set used for fitting, defined in terms of symmetrized harmonics.
  - Chapter 11: *Basic fit setup and numerics*: introduces methods for setting up the data to fit, and running fits in various ways.
- Case studies
  - Chapter 12: *Case study: Generalised bootstrapping for a homonuclear diatomic scattering system, N_2~(D_{\infty h})*: A "simple" 1D case, here the $D_{\infty h}$ molecular symmetry matches the rotational wavepacket and detection symmetry.
  - Chapter 13: *Case study: Generalised bootstrapping for a linear heteronuclear scattering system, OCS~(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 14: *Case study: Generalised bootstrapping for a general asymmetric top scattering system, C_2H_4~(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.

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.

CHAPTER

TEN

BASIS SETS FOR FITTING

In order to 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 fitting.

Numerically, there are two main methods to do this explored 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, specifically ePolyScat (ePS) [27, 28, 29, 30] matrix elements, in order to determine allowed components.

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

10.1 Symmetry-defined basis sets

As illustrated in Sect. 7.2, 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.

10.2 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 [39]), and - potentially - may bias the matrix element analysis/retrieval in cases where the experimental results and computational results are significantly different.

10.3 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.

10.3.1 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 irrep - physically this corresponds to the arbitrary orientation of these axes in the *MF*.

Following Eq. (7.12), the allowed components (irreducible representations) can be determined by hand making use of character and direct product tables:

$$\Gamma^+ \otimes \Gamma^e \otimes \Gamma_{\text{dipole}} \otimes \Gamma^i \supseteq \Sigma_g^+ \tag{10.1}$$

$$\Sigma_g^+ \otimes \Gamma^e \otimes \begin{matrix}\Pi_u\,(x,y)\\ \Sigma_u^+\,(z)\end{matrix} \otimes \Sigma_g^+ \supseteq \Sigma_g^+ \tag{10.2}$$

$$\Sigma_g^+ \otimes \Gamma^e \otimes \begin{matrix}\Pi_u\,(x,y)\\ \Sigma_u^+\,(z)\end{matrix} \supseteq \Sigma_g^+ \tag{10.3}$$

$$\Gamma^e \otimes \begin{matrix}\Pi_u\,(x,y)\\ \Sigma_u^+\,(z)\end{matrix} \supseteq \Sigma_g^+ \tag{10.4}$$

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. [131, 132, 133]. 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. Katzers are useful.

Hence the allowed continuum components are given by:

$$\Gamma^e = \begin{matrix}\Pi_u\,(x,y)\\ \Sigma_u^+\,(z)\end{matrix} \tag{10.5}$$

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. 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 $\Gamma_{\text{scat}} = \Gamma^+ \otimes \Gamma^e$:

$$\Gamma_{\text{scat}} = \Sigma_g^+ \otimes \begin{matrix}\Pi_u\,(x,y)\\ \Sigma_u^+\,(z)\end{matrix} = \begin{matrix}\Pi_u\,(x,y)\\ \Sigma_u^+\,(z)\end{matrix} \tag{10.6}$$

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

10.3.2 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:

$$\Gamma^e = \begin{matrix} \Sigma_g^+ + \Delta_g\,(x,y) \\ \Pi_g\,(z) \end{matrix} \tag{10.7}$$

And total scattering state symmetries:

$$\Gamma_{\text{scat}} = \Pi_u \otimes \begin{matrix} \Sigma_g^+ + \Delta_g\,(x,y) \\ \Pi_g\,(z) \end{matrix} = \begin{matrix} \Pi_u + \Delta_u + \Phi_u\,(x,y) \\ \Sigma_u^+ + \Sigma_u^- + \Delta_u\,(z) \end{matrix} \tag{10.8}$$

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.

10.3.3 Defining symmetrized harmonics

In the examples above, the allowed irreducible representations are defined by hand from direct product tables for illustrative purposes. But - by hand - it is tedious to categorize/define the allow spherical harmonics and linear combinations (see Sect. 7.6.2 and references therein for more details). With the Photoelectron Metrology Toolkit [3], 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. 7.2, and is extended here for the example cases above.

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
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']}}
```

```
/home/jovyan/github/PEMtk/pemtk/sym/_dipoleTerms.py:102: FutureWarning: iteritems␣
↪is deprecated and will be removed in a future version. Use .items instead.
  for (col, vals) in dipolePD.iteritems():
```

```
# 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. 10.1: 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
symObjE1u.displayXlm(symFilter = True, YlmType='comp')
```

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. 10.2: Dipole-allowed continuum symmetries ("Target") for D_{10h}, E_{1u} ionization.

```
/opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:344:␣
↪FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise␣
↪the base implementation of `Styler.to_latex` for formatting and rendering. The␣
↪arguments signature may therefore change. It is recommended instead to use␣
↪`DataFrame.style.to_latex` which also contains additional functionality.
  return method()
```

Character (Γ)	SALC (h)	PFIX (μ)	l m	b 0	1	2	3	4
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			(0.7071067811865475-0j)		
	1	0	-1					(0.707106781186
			1					(0.707106781186
		1	-1					(-0.70710678118
			1					(0.707106781186
E2g	0	0	-2			(0.7071067811865475+0j)		
			2			(0.7071067811865475+0j)		
		1	-2			(-0.7071067811865475+0j)		
			2			(0.7071067811865475+0j)		
	1	0	-2					(0.707106781186
			2					(0.707106781186
		1	-2					(-0.70710678118
			2					(0.707106781186

10.3.4 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 paramters. The matrix elements are currently defined in the Photoelectron Metrology Toolkit [3] and the ePSproc codebase [24, 25, 26] following the definitions in ePolyScat (ePS) [27, 28, 29, 30], and the symmetry-defined cases can be remapped to this format. (Further information can be found in the PEMtk documentation [14].) For a comparison with *ab initio* matrix elements, see Sect. 10.4.

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. 10.3.4.
2. From a given basis set, the parameters used for fitting data are determined. This converts the parameters to `lmfit` objects in magnitude-phase form, and (optionally) sets various symmetry relations and constraints. This is explored in Sect. 10.3.4.

Further examples can be found in the remainder of this text, and also in the PEMtk documentation [14].

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 indicies 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) [27, 28, 29, 30], 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
↪'}  # TBC - decide on default here.
# 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 produts, 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('')
```

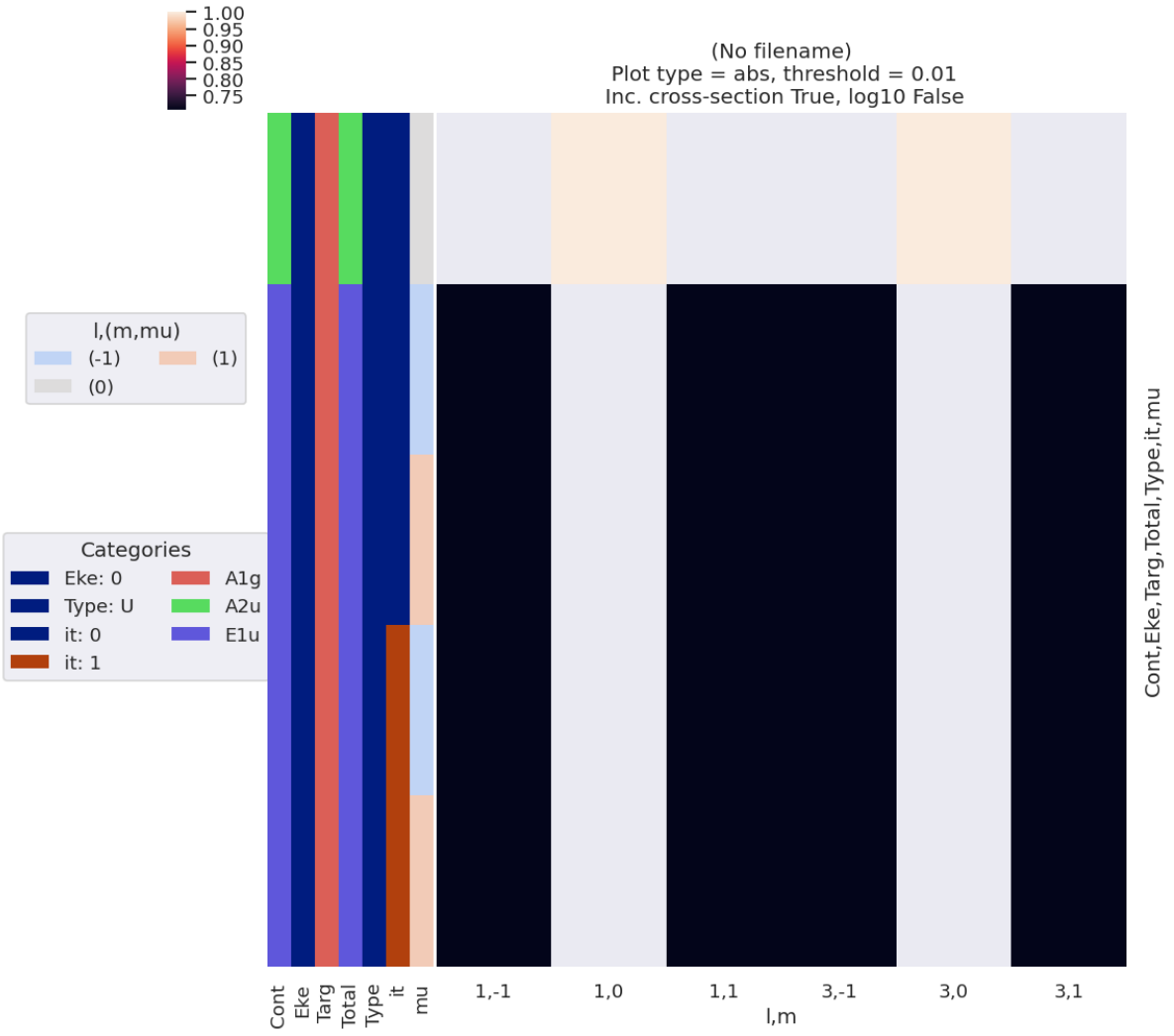
```
*** Mapping coeffs to ePSproc dataType = matE
Remapped dims: {'C': 'Cont', 'mu': 'it'}
Added dim Eke
Added dim Targ
Added dim Total
Added dim mu
Added dim Type
*** Updated self.coeffs['matE'] with new coords.
Assigned 'Total' from A1g x A1g = ['A1g']
Assigned 'Total' from A2u x A1g = ['A2u']
Assigned '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['matE'] with new coords.
Assigned dipole-allowed terms for dim = 'Cont' to self.coeffs['symAllowed']
```

```
/opt/conda/lib/python3.10/site-packages/IPython/core/formatters.py:344:␣
↪FutureWarning: In future versions `DataFrame.to_latex` is expected to utilise␣
↪the base implementation of `Styler.to_latex` for formatting and rendering. The␣
↪arguments signature may therefore change. It is recommended instead to use␣
↪`DataFrame.style.to_latex` which also contains additional functionality.
  return method()
```

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

```
# Plot values
%matplotlib inline
daPlot, daPlotpd, legendList, gFig =  ep.lmPlot(symObjA1g.coeffs['symAllowed']['XR'][
 ↪'b (comp)'], xDim={'LM':['l','m']}, sumDims='h')    #, cmap=cmap, mDimLabel='m');  #␣
 ↪linear ADM case
```

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



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 detailed in **SECT XX**, 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.

The automated routine currently checks for the following relationships: identity (equal complex values), magnitude and phase equality, complex rotations by $\pm\pi$, and 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. 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 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
```

(continues on next page)

(continued from previous page)

```
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

# General case
# for dataType in ['matE']:   #,'BLM']:

#     # Special cases...
#     if sym=='D2h':
#         data.data[dataKey][dataType] = symObj.coeffs[dataType]['b (comp)'].sum(['h',
 ↪'muX'])  # Select expansion in complex harmonics, and sum redundant dims

#     # General case
#     else:
#         data.data[dataKey][dataType] = symObj.coeffs[dataType]['b (comp)']

#     data.data[dataKey][dataType].attrs = symObj.coeffs[dataType].attrs

# Specific case
dataType = 'matE'
data.data[dataKey][dataType] = symObjA1g.coeffs['symAllowed']['XR']['b (comp)'].sum('h
 ↪')
```

(continues on next page)

(continued from previous page)

```
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'}}
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()
```

```
display(data.params)
```

```
Parameters([('m_A2u_0_A1g_A2u_1_0_0_0', <Parameter 'm_A2u_0_A1g_A2u_1_0_0_0',␣
↪value=1.0, bounds=[0.0001:5.0]>), ('m_A2u_0_A1g_A2u_3_0_0_0', <Parameter 'm_A2u_
↪0_A1g_A2u_3_0_0_0', value=1.0, bounds=[0.0001:5.0]>), ('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]>), ('m_E1u_0_A1g_E1u_1_n1_n1_1', <Parameter 'm_E1u_0_A1g_E1u_1_n1_n1_1
↪', value=0.7071067811865475, bounds=[0.0001:5.0]>), ('m_E1u_0_A1g_E1u_1_n1_1_0',
↪<Parameter 'm_E1u_0_A1g_E1u_1_n1_1_0', value=0.7071067811865475, bounds=[0.
↪0001:5.0], expr='m_E1u_0_A1g_E1u_1_n1_n1_0'>), ('m_E1u_0_A1g_E1u_1_n1_1_1',
↪<Parameter 'm_E1u_0_A1g_E1u_1_n1_1_1', value=0.7071067811865475, bounds=[0.
↪0001:5.0], expr='m_E1u_0_A1g_E1u_1_n1_n1_0'>), ('m_E1u_0_A1g_E1u_1_1_n1_0',
↪<Parameter 'm_E1u_0_A1g_E1u_1_1_n1_0', value=0.7071067811865475, bounds=[0.
↪0001:5.0], expr='m_E1u_0_A1g_E1u_1_n1_n1_0'>), ('m_E1u_0_A1g_E1u_1_1_n1_1',
↪<Parameter 'm_E1u_0_A1g_E1u_1_1_n1_1', value=0.7071067811865475, bounds=[0.
↪0001:5.0], expr='m_E1u_0_A1g_E1u_1_n1_n1_0'>), ('m_E1u_0_A1g_E1u_1_1_1_0',
↪<Parameter 'm_E1u_0_A1g_E1u_1_1_1_0', value=0.7071067811865475, bounds=[0.0001:5.
↪0], expr='m_E1u_0_A1g_E1u_1_n1_n1_0'>), ('m_E1u_0_A1g_E1u_1_1_1_1', <Parameter
↪'m_E1u_0_A1g_E1u_1_1_1_1', value=0.7071067811865475, bounds=[0.0001:5.0], expr=
↪'m_E1u_0_A1g_E1u_1_n1_n1_0'>), ('m_E1u_0_A1g_E1u_3_n1_n1_0', <Parameter 'm_E1u_0_
↪A1g_E1u_3_n1_n1_0', value=0.7071067811865475, bounds=[0.0001:5.0]>), ('m_E1u_0_
↪A1g_E1u_3_n1_n1_1', <Parameter 'm_E1u_0_A1g_E1u_3_n1_n1_1', value=0.
↪7071067811865475, bounds=[0.0001:5.0]>), ('m_E1u_0_A1g_E1u_3_n1_1_0', <Parameter
↪'m_E1u_0_A1g_E1u_3_n1_1_0', value=0.7071067811865475, bounds=[0.0001:5.0], expr=
↪'m_E1u_0_A1g_E1u_3_n1_n1_0'>), ('m_E1u_0_A1g_E1u_3_n1_1_1', <Parameter 'm_E1u_0_
↪A1g_E1u_3_n1_1_1', value=0.7071067811865475, bounds=[0.0001:5.0], expr='m_E1u_0_
↪A1g_E1u_3_n1_n1_0'>), ('m_E1u_0_A1g_E1u_3_1_n1_0', <Parameter 'm_E1u_0_A1g_E1u_3_
↪1_n1_0', value=0.7071067811865475, bounds=[0.0001:5.0], expr='m_E1u_0_A1g_E1u_3_
↪n1_n1_0'>), ('m_E1u_0_A1g_E1u_3_1_n1_1', <Parameter 'm_E1u_0_A1g_E1u_3_1_n1_1',␣
↪value=0.7071067811865475, bounds=[0.0001:5.0], expr='m_E1u_0_A1g_E1u_3_n1_n1_0'>
↪), ('m_E1u_0_A1g_E1u_3_1_1_0', <Parameter 'm_E1u_0_A1g_E1u_3_1_1_0', value=0.
↪7071067811865475, bounds=[0.0001:5.0], expr='m_E1u_0_A1g_E1u_3_n1_n1_0'>), ('m_
↪E1u_0_A1g_E1u_3_1_1_1', <Parameter 'm_E1u_0_A1g_E1u_3_1_1_1', value=0.
↪7071067811865475, bounds=[0.0001:5.0], expr='m_E1u_0_A1g_E1u_3_n1_n1_0'>), ('p_
↪A2u_0_A1g_A2u_1_0_0_0', <Parameter 'p_A2u_0_A1g_A2u_1_0_0_0', value=0.0 (fixed),␣
↪bounds=[-3.141592653589793:3.141592653589793]>), ('p_A2u_0_A1g_A2u_3_0_0_0',
↪<Parameter 'p_A2u_0_A1g_A2u_3_0_0_0', value=0.0, bounds=[-3.141592653589793:3.
↪141592653589793]>), ('p_E1u_0_A1g_E1u_1_n1_n1_0', <Parameter 'p_E1u_0_A1g_E1u_1_
↪n1_n1_0', value=0.0, bounds=[-3.141592653589793:3.141592653589793]>), ('p_E1u_0_
↪A1g_E1u_1_n1_n1_1', <Parameter 'p_E1u_0_A1g_E1u_1_n1_n1_1', value=3.
```

(continues on next page)

(continued from previous page)

```
data.params._repr_html_()
```

```
'<table class="jp-toc-ignore"><caption>Parameters</caption><tr><th style=\'text-
↪align:left\'>name</th><th style=\'text-align:left\'>value</th><th style=\'text-
↪align:left\'>initial value</th><th style=\'text-align:left\'>min</th><th style=\
↪'text-align:left\'>max</th><th style=\'text-align:left\'>vary</th><th style=\
↪'text-align:right\'>expression</th></tr><tr><td style=\'text-align:left\'>m_A2u_
↪0_A1g_A2u_1_0_0_0</td><td style=\'text-align:left\'> 1.00000000</td><td style=\
↪'text-align:left\'>1.0</td><td style=\'text-align:left\'> 1.0000e-04</td><td␣
↪style=\'text-align:left\'> 5.00000000</td><td style=\'text-align:left\'>True</td>
↪<td style=\'text-align:right\'></td></tr><tr><td style=\'text-align:left\'>m_A2u_
↪0_A1g_A2u_3_0_0_0</td><td style=\'text-align:left\'> 1.00000000</td><td style=\
↪'text-align:left\'>1.0</td><td style=\'text-align:left\'> 1.0000e-04</td><td␣
↪style=\'text-align:left\'> 5.00000000</td><td style=\'text-align:left\'>True</td>
↪<td style=\'text-align:right\'></td></tr><tr><td style=\'text-align:left\'>m_E1u_
↪0_A1g_E1u_1_n1_n1_0</td><td style=\'text-align:left\'> 0.70710678</td><td style=\
↪'text-align:left\'>0.7071067811865475</td><td style=\'text-align:left\'> 1.0000e-
↪04</td><td style=\'text-align:left\'> 5.00000000</td><td style=\'text-align:left\
↪'>True</td><td style=\'text-align:right\'></td></tr><tr><td style=\'text-
↪align:left\'>m_E1u_0_A1g_E1u_1_n1_n1_1</td><td style=\'text-align:left\'> 0.
↪70710678</td><td style=\'text-align:left\'>0.7071067811865475</td><td style=\
↪'text-align:left\'> 1.0000e-04</td><td style=\'text-align:left\'> 5.00000000</td>
↪<td style=\'text-align:left\'>True</td><td style=\'text-align:right\'></td></tr>
↪<tr><td style=\'text-align:left\'>m_E1u_0_A1g_E1u_1_n1_1_0</td><td style=\'text-
↪align:left\'> 0.70710678</td><td style=\'text-align:left\'>0.7071067811865475</
↪td><td style=\'text-align:left\'> 1.0000e-04</td><td style=\'text-align:left\'>␣
↪5.00000000</td><td style=\'text-align:left\'>False</td><td style=\'text-
↪align:right\'>m_E1u_0_A1g_E1u_1_n1_n1_0</td></tr><tr><td style=\'text-align:left\
↪'>m_E1u_0_A1g_E1u_1_n1_1_1</td><td style=\'text-align:left\'> 0.70710678</td><td␣
↪style=\'text-align:left\'>0.7071067811865475</td><td style=\'text-align:left\'>␣
↪1.0000e-04</td><td style=\'text-align:left\'> 5.00000000</td><td style=\'text-
↪align:left\'>False</td><td style=\'text-align:right\'>m_E1u_0_A1g_E1u_1_n1_n1_0</
↪td></tr><tr><td style=\'text-align:left\'>m_E1u_0_A1g_E1u_1_1_n1_0</td><td␣
↪style=\'text-align:left\'> 0.70710678</td><td style=\'text-align:left\'>0.
↪7071067811865475</td><td style=\'text-align:left\'> 1.0000e-04</td><td style=\
↪'text-align:left\'> 5.00000000</td><td style=\'text-align:left\'>False</td><td␣
↪style=\'text-align:right\'>m_E1u_0_A1g_E1u_1_n1_n1_0</td></tr><tr><td style=\
↪'text-align:left\'>m_E1u_0_A1g_E1u_1_1_n1_1</td><td style=\'text-align:left\'> 0.
↪70710678</td><td style=\'text-align:left\'>0.7071067811865475</td><td style=\
↪'text-align:left\'> 1.0000e-04</td><td style=\'text-align:left\'> 5.00000000</td>
↪<td style=\'text-align:left\'>False</td><td style=\'text-align:right\'>m_E1u_0_
↪A1g_E1u_1_n1_n1_0</td></tr><tr><td style=\'text-align:left\'>m_E1u_0_A1g_E1u_1_1_
↪1_0</td><td style=\'text-align:left\'> 0.70710678</td><td style=\'text-
↪align:left\'>0.7071067811865475</td><td style=\'text-align:left\'> 1.0000e-04</
↪td><td style=\'text-align:left\'> 5.00000000</td><td style=\'text-align:left\'>
↪False</td><td style=\'text-align:right\'>m_E1u_0_A1g_E1u_1_n1_n1_0</td></tr><tr>
↪<td style=\'text-align:left\'>m_E1u_0_A1g_E1u_1_1_1_1</td><td style=\'text-
↪align:left\'> 0.70710678</td><td style=\'text-align:left\'>0.7071067811865475</
↪td><td style=\'text-align:left\'> 1.0000e-04</td><td style=\'text-align:left\'>␣
↪5.00000000</td><td style=\'text-align:left\'>False</td><td style=\'text-
↪align:right\'>m_E1u_0_A1g_E1u_1_n1_n1_0</td></tr><tr><td style=\'text-align:left\
↪'>m_E1u_0_A1g_E1u_3_n1_n1_0</td><td style=\'text-align:left\'> 0.70710678</td>
↪<td style=\'text-align:left\'>0.7071067811865475</td><td style=\'text-align:left\
↪'> 1.0000e-04</td><td style=\'text-align:left\'> 5.00000000</td><td style=\'text-
↪align:left\'>True</td><td style=\'text-align:right\'></td></tr><tr><td style=\
↪'text-align:left\'>m_E1u_0_A1g_E1u_3_n1_n1_1</td><td style=\'text-align:left\'>␣
↪0.70710678</td><td style=\'text-align:left\'>0.7071067811865475</td><td style=\
↪'text-align:left\'> 1.0000e-04</td><td style=\'text-align:left\'> 5.00000000</td>
↪<td style=\'text-align:left\'>True</td><td style=\'text-align:right\'></td></tr>
↪<tr><td style=\'text-align:left\'>m_E1u_0_A1g_E1u_3_n1_1_0</td><td style=\'text-
↪align:left\'> 0.70710678</td><td style=\'text-align:left\'>0.7071067811865475</
```

(continues on next page)

(continued from previous page)

```
None
```

Fig. 10.3: 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.

Modifying fitting basis parameters

A brief illustration of defining constraints is given below, for more details see the PEMtk documentation [14], particularly the basic fitting guide. For more details on the base lmfit parameters class that is used here, see the `lmfit` library [57, 58], particularly the documentation on parameters and constraints.

```
# 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, and properties modified using lmfit
↪'s `.set()` method.

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]>
```

```
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 [14] for more information.

```
# TODO: needs container rebuild for updated PEMtk code 25/04/23

# # Manual configuration of matrix elements
# # from pemtk.fit.fitClass import pemtkFit

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

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

10.4 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) [27, 28, 29, 30] 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 N_2. Further ePolyScat (ePS) [27, 28, 29, 30] datasets are available from the ePSdata [39], 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. (7.3)) versus the symmetry-allowed harmonics and associated $b_{hl\lambda}^{\Gamma\mu}$ parameters (Eq. (7.37)). 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.

```
# Pull N2 data from ePSproc Github repo
import wget
from pathlib import Path

# 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"}

# Set data dir
dataPath = Path(Path.cwd(), 'n2Data')

# Create and pull files if dir not present (NOTE doesn't check per file here)
if not dataPath.is_dir():
    dataPath.mkdir()
```

(continues on next page)

(continued from previous page)

```
    # Pull files with wget
    for k,v in urls.items():
        wget.download(v+'?raw=true',out=dataPath.as_posix())  # For Github add '?
↪raw=true' to URL


# List files
list(dataPath.glob('*.out'))
```

```
[PosixPath('/home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↪n2Data/n2_1pu_0.1-50.1eV_A2.inp.out'),
 PosixPath('/home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↪n2Data/n2_3sg_0.1-50.1eV_A2.inp.out')]
```

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

# from epsproc.classes.multiJob import ePSmultiJob
# from epsproc.classes.base import ePSbase
# from pemtk.fit.fitClass import pemtkFit

# 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].

*** 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'}
```

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

Here the symmetry-defined basis has two degenerate continuua, `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 for comparison if comparison with computational results is required.
3. Add degeneracy factors if required (otherwise will be subsumed into matrix element values).

CHAPTER

ELEVEN

BASIC FIT SETUP AND NUMERICS

11.1 Init and pulling data

Here the setup is mainly handled by some basic scripts, these follow the outline in the PEMtk documentation [14], see in particular the intro to fitting.

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

# from epsproc.util.io import getFilesFromGithub

# fDict, fAll = getFilesFromGithub(subpath='data/alignment/OCS_ADMs_28K_VM_070722',␣
 ↪ref='dev')   # OK

# 26/05/23 - Monkeypatch version for debug
# Above should be fine after source updates
import requests
from epsproc.util import io
io.requests = requests

dataName = 'n2fitting'
fDictMatE, fAllMatE = io.getFilesFromGithub(subpath='data/photoionization/n2_multiorb
 ↪', dataName=dataName)  #, download=False)   # N2 matrix elements
fDictADM, fAllMatADM = io.getFilesFromGithub(subpath='data/alignment',␣
 ↪dataName=dataName)  #, download=False)   # N2 alignment data
# Note this is missing script - should consolidate all to book repo?

# Alternatively supply URLs directly for file downloader
# 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",
        '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"}

fList, fDict = io.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
```

11.2 Setup with options

Following the PEMtk documentation [14], the fitting workspace can be configured by setting:

1. A fitting basis set, either from computational matrix elements, from symmetry constraints, or manually. (See Sect. 10 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.

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

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

```
*** Job subset details
Key: subset
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).
```

(continues on next page)

(continued from previous page)

```
{   'batch': 'ePS n2, batch n2_3sg_0.1-50.1eV, orbital A2',
    'event': ' N2 X-state (3sg-1)',
    'orbE': -17.34181645456815,
    'orbLabel': '3sg-1'}
```

11.2.1 Alignment distribution moments (ADMs)

The class wraps ep.setADMs(). This returns an isotropic distribution by default, or values can be set explicitly from a list. Values are set in `self.data['ADM']`.

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

```
# Default case
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
```

```
# Load time-dependent ADMs for N2 case
# Adapted from ePSproc_AFBLM_testing_010519_300719.m

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)
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],
```

(continues on next page)

(continued from previous page)

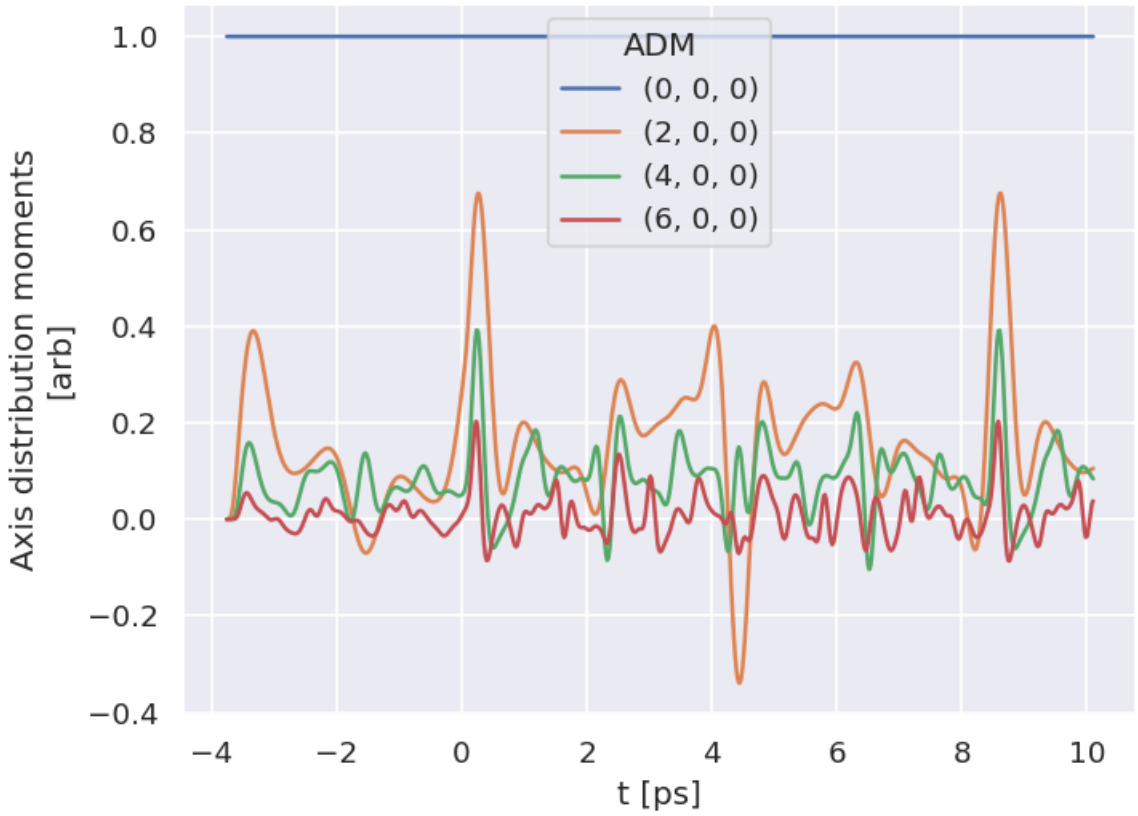
```
       [-2.26243113e-17+0.00000000e+00j,  2.43430608e-08+1.04125246e-20j,
         9.80188266e-08+6.89166168e-20j, ...,
         1.05433798e-01-1.62495135e-18j,  1.05433798e-01-1.62495135e-18j,
         1.05433798e-01-1.62495135e-18j],
       [ 1.55724057e-16+0.00000000e+00j, -3.37021111e-10-6.81416260e-20j,
         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
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)
```

```
# Wrapper OK with Matplotlib, but needs work for hv case (see below)
%matplotlib inline
data.ADMplot(keys = 'ADM')
```

```
Dataset: ADM, ADM
```



11.2.2 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
```

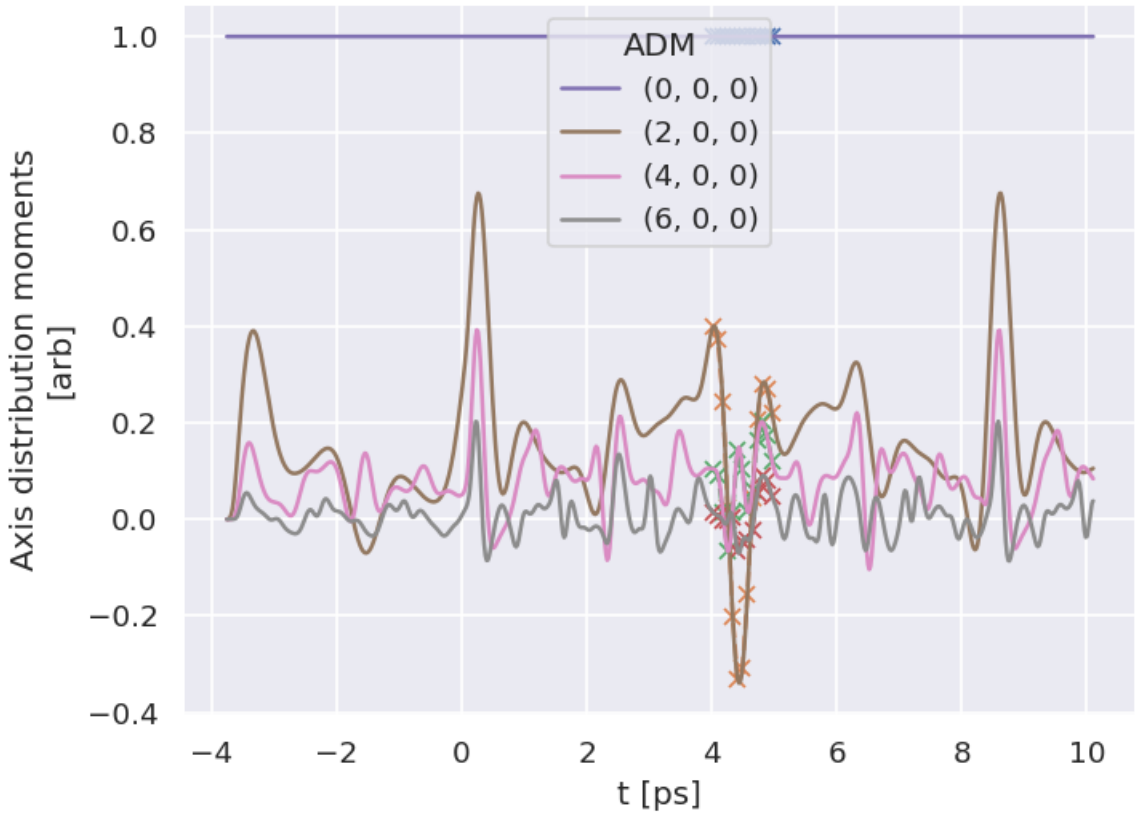
11.2.3 Subselect data

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]`, 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

# Show subselected data
# data.data['subset']['matE']

# Tabulate the matrix elements
# Not showing as nice table for singleton case - pd.series vs. dataframe?
# data.matEtoPD(keys = 'subset', xDim = 'Sym', drop=False)

# 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%)
```

```
# Plot from Xarray vs. full dataset
# data.data['subset']['ADM'].where(ADMX['K']>0).real.squeeze().plot.line(x='t');
data.data['subset']['ADM'].real.squeeze().plot.line(x='t', marker = 'x', linestyle=
↪'dashed');
data.data['ADM']['ADM'].real.squeeze().plot.line(x='t');
```



11.3 Compute AF-β_{LM} and simulate data

With all the components set, some observables can be calculated. For testing, we'll also use this to simulate an experiemental trace…

Here we'll use `self.afblmMatEfit()`, which is also the main fitting routine, and essentially wraps `epsproc.afblmXprod()` to compute AF-β_{LM}s (for more details, see the ePSproc method development docs).

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

11.3.1 Compute AF-β_{LM}s

```
# data.afblmMatEfit(data = None)  # OK
BetaNormX, basis = data.afblmMatEfit()  # OK, uses default polarizations & ADMs as␣
↪set in data['subset']
# BetaNormX, basis = data.afblmMatEfit(ADM = data.data['subset']['ADM'])  # OK, but␣
↪currently using default polarizations
# BetaNormX, basis = data.afblmMatEfit(ADM = data.data['subset']['ADM'], pol = data.
↪data['pol']['pol'].sel(Labels=['x']))
# BetaNormX, basis = data.afblmMatEfit(ADM = data.data['subset']['ADM'], pol = data.
↪data['pol']['pol'].sel(Labels=['x','y']))  # This fails for a single label...?
# BetaNormX, basis = data.afblmMatEfit(RX=data.data['pol']['pol'])  # This currently␣
↪fails, need to check for consistency in ep.sphCalc.WDcalc()
```

(continues on next page)

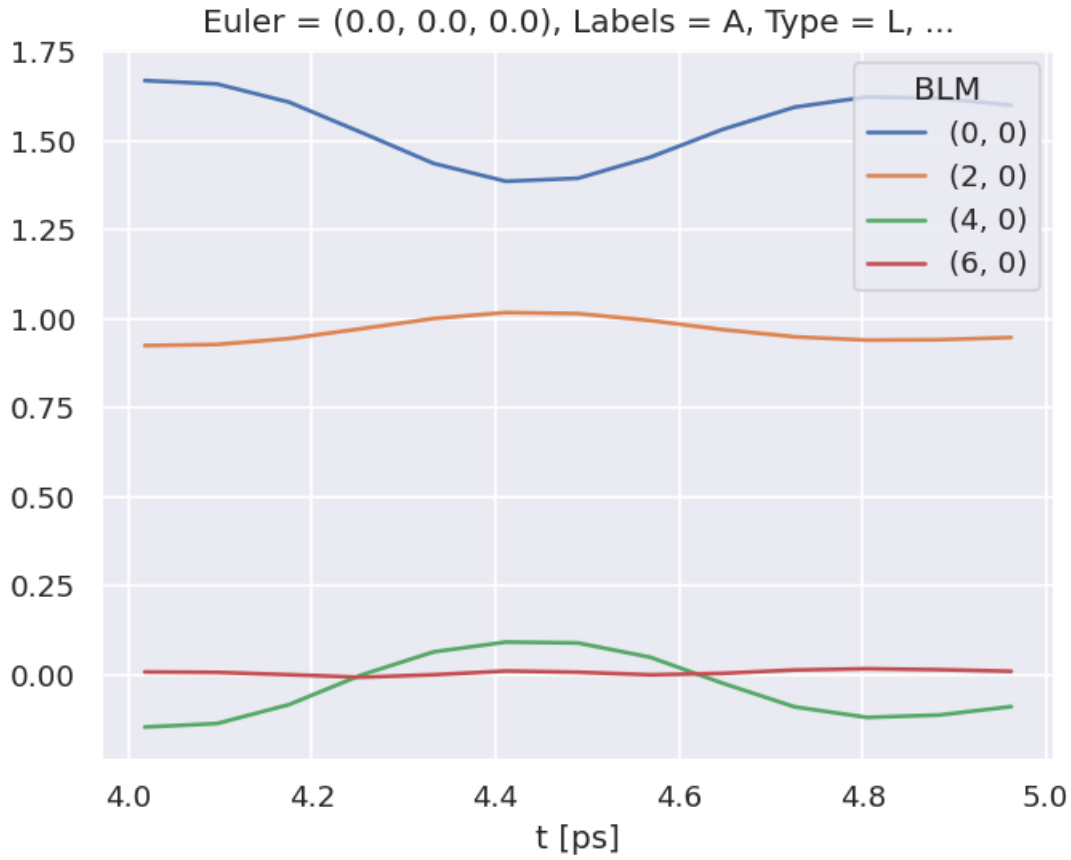
(continued from previous page)

```
                                                                    # - looks like␣
↪set values and inputs are not consistent in this case? Not passing angs correctly,␣
↪or overriding?
                                                                    # - See also␣
↪recently-added sfError flag, which may cause additional problems.
```

11.3.2 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.

```
basis.keys()
```

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

11.4 Fitting the data

In order to fit data, and extract matrix elements from an experimental case, we'll use the lmfit library. This wraps core Scipy fitting routines with additional objects and methods, and is further wrapped for this specific class of problems in `pemtkFit` class we're using here.

11.4.1 Set the data to fit

Here we'll use the values calculated above as our test data. This currently needs to be set as `self.data['subset']['AFBLM']` for fitting.

```
# data.data['subset']['AFBLM'] = BetaNormX  # Set manually

data.setData('sim', BetaNormX)  # Set simulated data to master structure as "sim"
data.setSubset('sim','AFBLM')   # Set to 'subset' to use for fitting.
```

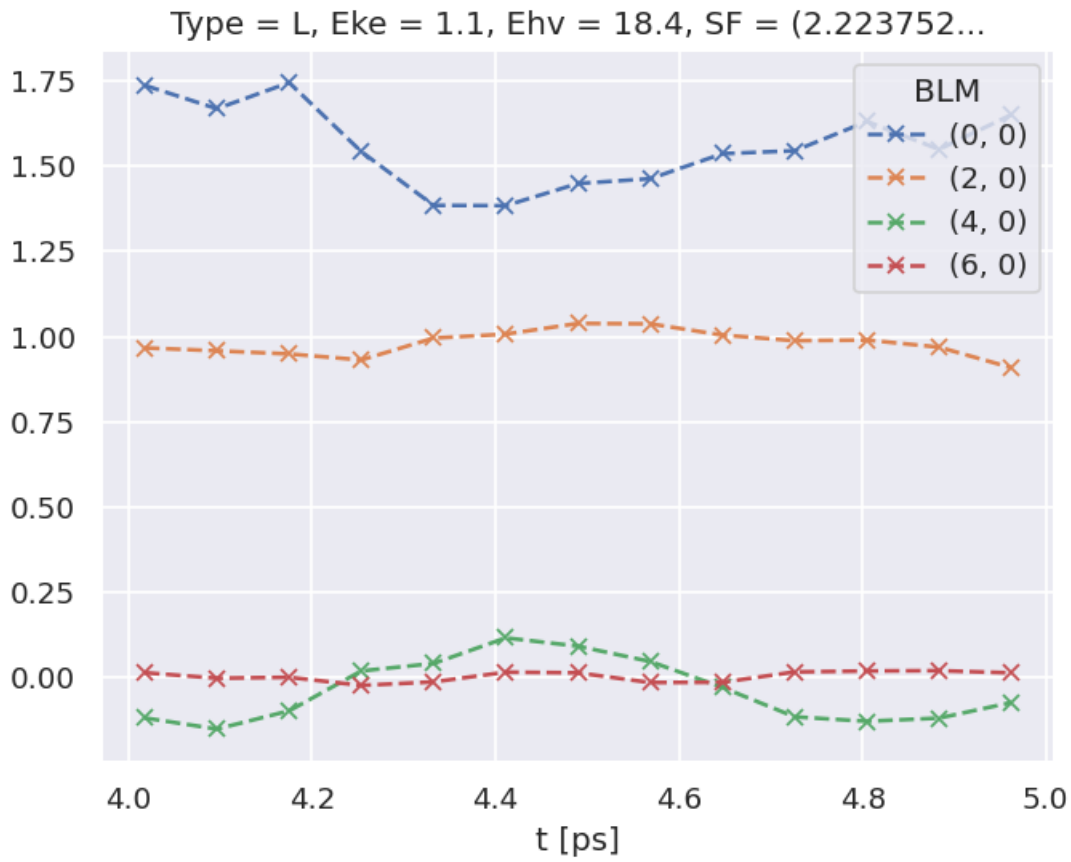
```
Subselected from dataset 'sim', dataType 'AFBLM': 52 from 52 points (100.00%)
```

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

11.4.2 Adding noise

```
# Add noise with np.random.normal
# https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html
# data.data['subset']['AFBLM']

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
# data.data['subset']['AFBLM']['noise'] = ((data.data['subset']['AFBLM'].t, data.data[
 ↪'subset']['AFBLM'].l), noise)
# xr.where(noiseXR.l>0, noiseXR/noiseXR.l, noiseXR)
noiseXR = noiseXR.where(noiseXR.l<1, noiseXR/(noiseXR.l))  # Scale by L

data.data['subset']['AFBLM'] = data.data['subset']['AFBLM'] + noiseXR
data.data['subset']['AFBLM'] = data.data['subset']['AFBLM'].where(data.data['subset'][
 ↪'AFBLM'].m == 0, 0)

data.BLMfitPlot()
```

```
Dataset: subset, AFBLM
```



11.4.3 Setting up the fit parameters

In this case, we can work from the existing matrix elements to speed up parameter creation, although in practice this may need to be approached ab initio - nonetheless, the method will be the same, and the ab initio case detailed later.

```
# Input set, as defined earlier
# data.data['subset']['matE'].pd
```

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

```
Set 6 complex matrix elements to 12 fitting params, see self.params for details.
Auto-setting parameters.
```

```
Parameters([('m_PU_SG_PU_1_n1_1_1', <Parameter 'm_PU_SG_PU_1_n1_1_1', value=1.
↪784615753610107, bounds=[0.0001:5.0]>), ('m_PU_SG_PU_1_1_n1_1', <Parameter 'm_PU_
↪SG_PU_1_1_n1_1', value=1.784615753610107, bounds=[0.0001:5.0], expr='m_PU_SG_PU_
↪1_n1_1_1'>), ('m_PU_SG_PU_3_n1_1_1', <Parameter 'm_PU_SG_PU_3_n1_1_1', value=0.
↪802904951323892, bounds=[0.0001:5.0]>), ('m_PU_SG_PU_3_1_n1_1', <Parameter 'm_PU_
↪SG_PU_3_1_n1_1', value=0.802904951323892, bounds=[0.0001:5.0], expr='m_PU_SG_PU_
↪3_n1_1_1'>), ('m_SU_SG_SU_1_0_0_1', <Parameter 'm_SU_SG_SU_1_0_0_1', value=2.
↪686062120382649, bounds=[0.0001:5.0]>), ('m_SU_SG_SU_3_0_0_1', <Parameter 'm_SU_
↪SG_SU_3_0_0_1', value=1.109153108617096, bounds=[0.0001:5.0]>), ('p_PU_SG_PU_1_
↪n1_1_1', <Parameter 'p_PU_SG_PU_1_n1_1_1', value=-0.8610414024232178 (fixed),
↪bounds=[-3.141592653589793:3.141592653589793]>), ('p_PU_SG_PU_1_1_n1_1',
↪<Parameter 'p_PU_SG_PU_1_1_n1_1', value=-0.8610414024232179, bounds=[-3.
↪141592653589793:3.141592653589793], expr='p_PU_SG_PU_1_n1_1_1'>), ('p_PU_SG_PU_3_
↪n1_1_1', <Parameter 'p_PU_SG_PU_3_n1_1_1', value=-3.1204444620772467, bounds=[-3.
↪141592653589793:3.141592653589793]>), ('p_PU_SG_PU_3_1_n1_1', <Parameter 'p_PU_
↪SG_PU_3_1_n1_1', value=-3.1204444620772467, bounds=[-3.141592653589793:3.
↪141592653589793], expr='p_PU_SG_PU_3_n1_1_1'>), ('p_SU_SG_SU_1_0_0_1',
```

(continues on next page)

(continued from previous page)

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<mag<5, -pi<phase<pi.
- Relationships between the parameters are set by default, but can be set manually, see section below, or pass `paramsCons=None` to skip.

11.4.4 Running fits

Single fit

With the parameters and data set, just call `self.fit()`!

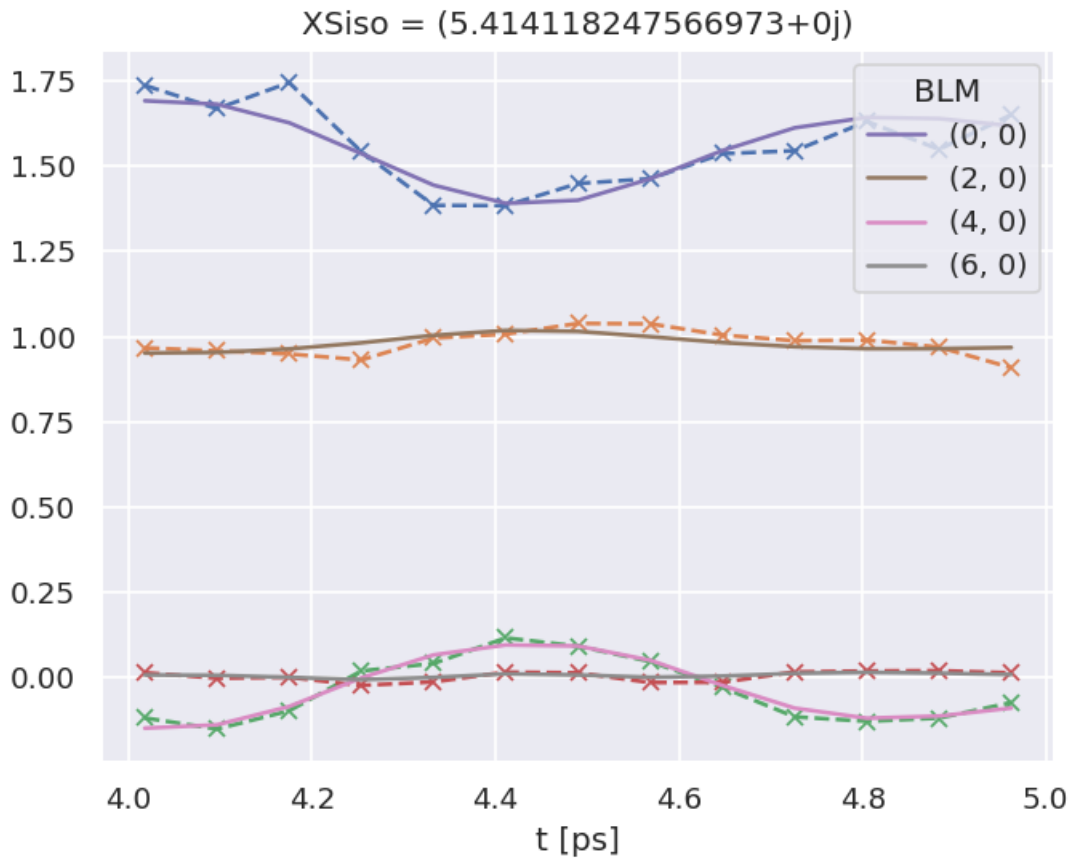
Statistics and outputs are handled by lmfit, which includes uncertainty estimates and correlations in the fitted parameters.

```
# data.randomizeParams()  # Randomize input parameters if desired
                          # For method testing using known initial params is also␣
↪useful
data.fit()
```

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

```
# Plot results with data
# data.BLMfitPlot(backend='hv')
data.BLMfitPlot()
```

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



Extended execution methods, including parallel and batched execution

See https://pemtk.readthedocs.io/en/latest/fitting/PEMtk_fitting_demo_multi-fit_tests_130621-para_010922.html

(1) serial execution

Either:

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

```
import time

start = time.time()

# Maual 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 = [0,100], parallel = False)
```

```
0.796954071521759
```

```
# We now have 10 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

Updated version including parallel fitting routine with `self.multiFit()` method.

This currently uses the XYZpy library for quick parallelization, although there is some additional setup overhead in the currently implementation due to class init per fit batch. The default aims to set ~90% CPU usage, based on core-count.

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

```
Number of processors:  64
Running pool on:  20
* 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.
* 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.
* 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.
* 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.
* 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.
* 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.
```

(continues on next page)

(continued from previous page)

```
* Set Holoviews with bokeh.
* pyevtk not found, VTK export not available.
* 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.
* 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.
* 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.
```

```
  0%|                                                                              ␣
↪                                                               | 0/10 [00:00<?, ?
↪it/s]OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_
↪active_levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
OMP: Info #276: omp_set_nested routine deprecated, please use omp_set_max_active_
↪levels instead.
100%|##########################################################################
↪####################################################| 10/10 [01:09<00:00,  6.
↪91s/it]
```

(c) Dump data

Various options are available. The most complete is to use Pickle (default case), although this is not suggested for archival use. For details see https://epsproc.readthedocs.io/en/dev/dataStructures/ePSproc_dataStructures_demo_070622.html

```
outStem = 'dataDump_N2'  # Set for file save later

# data.writeFitData(fName='N2_datadump')  # Use 'fName' to supply a filename
```

(continues on next page)

(continued from previous page)

```
data.writeFitData(dataPath = dataPath, outStem=outStem)  # Use 'outStem' to define a␣
↪filename which will be appended with a timestamp
                                                         # Set dataPath if desired,␣
↪otherwise will use working dir
```

```
Dumped self.data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/
↪part2/n2fitting/dataDump_N2_220723_14-10-38.pickle with pickle.
Dumped data to /home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↪n2fitting/dataDump_N2_220723_14-10-38.pickle with pickle.
```

```
PosixPath('/home/jovyan/jake-home/buildTmp/_latest_build/pdf/doc-source/part2/
↪n2fitting/dataDump_N2_220723_14-10-38.pickle')
```

11.5 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.

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

CHAPTER

TWELVE

CASE STUDY: GENERALISED BOOTSTRAPPING FOR A HOMONUCLEAR DIATOMIC SCATTERING SYSTEM, N_2 $(D_{\infty H})$

CHAPTER

THIRTEEN

CASE STUDY: GENERALISED BOOTSTRAPPING FOR A LINEAR HETERONUCLEAR SCATTERING SYSTEM, $OCS\ (C_{\infty V})$

CHAPTER

FOURTEEN

CASE STUDY: GENERALISED BOOTSTRAPPING FOR A GENERAL ASYMMETRIC TOP SCATTERING SYSTEM, C_2H_4 (D_{2H})

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. 7.3.3 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. 7.3.3 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. 7.3.3 for further details.

PADs
: Photoelectron angular distributions (PADs), often with a prefix denoting the reference frame, e.g. LFPADs, MFPADs (sometimes also hypenated, 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. (7.37). Often referred to simply as "beta parameters", and may be dependent on various properties, e.g. $\beta_{L,M}(\epsilon, t...)$. 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 $A^K_{Q,S}(t)$ 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. 7.5 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 [80] 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 ($> 10^{12}$~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. [134, 135, 136]. 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 paramters*).

partial-waves
: See *partial-wave expansion*.

channel functions
: Geometric (angular-momentum) coupling parameters in the tensor formulation of photoionzation, denoted by $\Upsilon^{u,\zeta\zeta'}_{L,M}$ herein. See Eq. (7.13). These can be regarded as an alternative form of the more traditional geometric coupling parameters (Eq. (7.9)). 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. (7.6)), and $\gamma_{\alpha\alpha_+ l\lambda m l'\lambda' m'}$ for the coherent square of these terms (Eq. (7.9)). 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. (7.6), (7.7)), and also appear as $\mathbb{I}^{\zeta\zeta'}$ for the coherent square of these terms in the *channel functions* (tensor) form, see Eq. (7.13). These complex matrix elements are the unknowns to be determined in quantum metrology with photoelectons fitting or reconstruction problems.

symmetrized harmonics
: A basis set of spherical harmonics expanded/defined for a given point-group symmetry. See Sect. 7.6.2, particularly Eq. (7.37), 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 $b^{\Gamma\mu}_{hl\lambda}$ 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](https://en.wikipedia.org/wiki/Euler_angles)[] and Zare, Chpt. 3 [74].

Wigner rotation matrix elements
: A function defining a basis set for rotations in three-dimensions ($SO(3)$). Also known as "Wigner D-matrix el-

ements". For discussion see Wikipedia](https://en.wikipedia.org/wiki/Wigner_D-matrix)[] and Zare, Chpt. 3 [74].

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 $A^K_{Q,S}(t)$. See Sect. 7.5 for details.

Part V

Test pages

CHAPTER

SEVENTEEN

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: Sat Jul 22 13:56:29 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-Parser       : 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
661a657a9ca8031ebab3a1f267273c95c073222a
```

```
# Check current remote commits
!git ls-remote --heads https://github.com/phockett/Quantum-Metrology-with-
 ↪Photoelectrons-Vol3
```

```
5fd6ba727637a5e727ae414a38fd5344c8740d93        refs/heads/gh-pages
c0fcec831b5683882b15ac1473fc6d732310da48        refs/heads/main
59edb777677b25f06f137421339d5bf4e971f366        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
80e15ac8e83e07bb8a52b5fb89a915c3ff4f52bb
```

```
# Check current remote commits
!git ls-remote --heads https://github.com/phockett/ePSproc
```

```
80e15ac8e83e07bb8a52b5fb89a915c3ff4f52bb        refs/heads/3d-AFPAD-dev
897d73392a7b32ffba4ca6b6b4755c61e7c1c8d7        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/certifi-2022.12.7
457f8cd85d89bd6474296b6c01e5165a4a7ce7fc        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/cryptography-39.0.1
2855573d0f088b45d19acf2fd9a71eeb7af0a29b        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/ipython-8.10.0
92c661789a7d2927f2b53d7266f57de70b3834fa        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/mistune-2.0.3
fe1e9540c7b91fe571f60562acd31d8e489d491e        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/nbconvert-6.5.1
70b80a1e3a54de91c2bfe3b6be82d611fcfd5f43        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/pillow-9.3.0
92fc79b09aafedadcb645f88bb7ed771c96d5b52        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/setuptools-65.5.1
fa33ed8d63a5c4a4043cc4c261059cc09e4c2bf7        refs/heads/dependabot/pip/notes/
 ↪envs/envs-versioned/wheel-0.38.1
41cdfe43750e08c510f98b05e024a9c62da42771        refs/heads/dependabot/pip/
 ↪setuptools-65.5.1
```

(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
```

```
d3693888600a5eb4d0e2281c4bb24e4fce221230        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           2023.5.7             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                   2023.5.7           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
ipympl                    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               h5cdd6b5_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
libllvm11                 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
libstdcxx-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                 h9c3ff4c_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
smmap                     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                      py_1    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-libsm                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. *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.

[3] Paul Hockett. Photoelectron Metrology Toolkit (PEMtk) Github repository. 2021. URL: https://github.com/phockett/PEMtk (visited on 2022-02-18).

[4] **missing journal in hockett2016PresentationsArchiveUltrafast**

[5] 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.

[6] **missing journal in Makhija2016a**

[7] Project Jupyter. URL: https://jupyter.org (visited on 2023-01-16).

[8] Python.org. July 2023. URL: https://www.python.org/ (visited on 2023-07-07).

[9] Jupyter Book Project. URL: https://jupyterbook.org.

[10] Executable Books Community. Jupyter Book. Zenodo, February 2020. URL: https://zenodo.org/record/4539666 (visited on 2023-01-16).

[11] MyST Markdown - Tools for the future of technical communication. URL: https://mystmd.org/ (visited on 2023-07-07).

[12] Sphinx documentation. URL: https://www.sphinx-doc.org (visited on 2023-07-07).

[13] Paul Hockett. Open Photoionization Docker Stacks. URL: https://github.com/phockett/open-photoionization-docker-stacks (visited on 2022-08-04).

[14] Paul Hockett. PEMtk - the Photoelectron Metrology Toolkit - documentation. 2021. URL: https://pemtk.readthedocs.io (visited on 2022-02-18).

[15] 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).

[16] Jake VanderPlas. Python Data Science Handbook. July 2023. URL: https://github.com/jakevdp/PythonDataScienceHandbook (visited on 2023-07-07).

[17] 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).

[18] Marcia McNutt. Taking up TOP. *Science*, 2016. URL: http://science.sciencemag.org/content/352/6290/1147.full (visited on 2017-04-04).

[19] 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.sciencemag.org.proxy.bib.uottawa.ca/content/348/6242/1422.full (visited on 2017-04-04).

[20] 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).

[21] **missing author in wikipeadiaOpenScience**

[22] Thomas Kluyver, Benjamin Ragan-Kelley, P&#233, Fernando Rez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Dami&#225 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).

[23] Brian E. Granger and Fernando Pérez. Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering*, 23(2):7–14, March 2021.

[24] 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.

[25] Paul Hockett. ePSproc: Post-processing for ePolyScat (Github repository). Github, 2016. URL: https://github.com/phockett/ePSproc.

[26] Paul Hockett. ePSproc: Post-processing for ePolyScat documentation. 2020. URL: https://epsproc.readthedocs.io (visited on 2022-02-18).

[27] 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).

[28] 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).

[29] Alexandra P P Natalense and Robert R Lucchese. Cross section and asymmetry parameter calculation for sulfur 1s photoionization of SF[sub 6]. *The Journal of Chemical Physics*, 111(12):5344, 1999. URL: http://link.aip.org/link/JCPSA6/v111/i12/p5344/s1&Agg=doi (visited on 2012-07-18).

[30] R R Lucchese. ePolyScat User's Manual. URL: https://epolyscat.droppages.com/ (visited on 2022-04-26).

[31] Andrew C. Brown, Gregory S. J. Armstrong, Jakub Benda, Daniel D. A. Clarke, Jack Wragg, Kathryn R. Hamilton, Zdeněk Mašín, Jimena D. Gorfinkiel, 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.

[32] Andrew C. Brown, Gregory S. J. Armstrong, Jakub Benda, Daniel D. A. Clarke, Jack Wragg, Kathryn R. Hamilton, Zdeněk Mašín, Jimena D. Gorfinkiel, 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.

[33] 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).

[34] 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.

[35] Mark S. Gordon. Gamess website. URL: http://www.msg.ameslab.gov/gamess/.

[36] AMOSGateway. URL: https://amosgateway.org/ (visited on 2023-07-10).

[37] Barry I. Schneider, Klaus Bartschat, Oleg Zatsarinny, Kathryn R. Hamilton, Igor Bray, Armin Scrinzi, Fernando Martin, Jesus Gonzalez Vasquez, Jonathan Tennyson, Jimena D. Gorfinkiel, 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).

[38] Barry I. Schneider, Klaus~Bartschat, Oleg Zatsarinny, Igor Bray, Armin Scrinzi, Fernando Martin, Markus Klinker, Jonathan Tennyson, Jimena D. Gorfinkiel, 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.

[39] Paul Hockett. ePS data: Photoionization calculations archive. 2019. URL: https://phockett.github.io/ePSdata/ (visited on 2022-02-16).

[40] Paul Hockett. ePSdata repositories on Zenodo. 2019. URL: https://zenodo.org/search?page=1&size=20&q=hockett&keywords=Data.

[41] NumPy. URL: https://numpy.org/ (visited on 2023-07-10).

[42] Pandas - Python Data Analysis Library. URL: https://pandas.pydata.org/ (visited on 2023-07-10).

[43] SciPy. URL: https://scipy.org/ (visited on 2023-07-10).

[44] 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).

[45] Xarray documentation. URL: https://docs.xarray.dev/en/latest/index.html (visited on 2022-08-03).

[46] Mike Boyle. Spherical Functions Github. April 2022. URL: https://github.com/moble/spherical_functions (visited on 2022-08-03).

[47] 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).

[48] Mike Boyle, Blair Bonnett, Jon Long, Martin Ling, stiiin, 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).

[49] Moble/quaternion Github. URL: https://github.com/moble/quaternion (visited on 2023-07-10).

[50] SciPy documentation. URL: https://docs.scipy.org/doc/scipy/index.html (visited on 2022-08-03).

[51] Mark A. Wieczorek and Matthias Meschede. SHtools Github. SHTOOLS, August 2022. URL: https://github.com/SHTOOLS/SHTOOLS (visited on 2022-08-03).

[52] 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).

[53] 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).

[54] Mark A. Wieczorek and Matthias Meschede. SHtools Docs. SHTOOLS, August 2022. URL: https://shtools.github.io/SHTOOLS/ (visited on 2022-08-03).

[55] 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).

[56] Marcus Johansson. Libmsym Github. July 2022. URL: https://github.com/mcodev31/libmsym (visited on 2022-08-03).

[57] LMFIT documentation. URL: https://lmfit.github.io/lmfit-py/intro.html (visited on 2022-08-03).

[58] 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).

[59] Xyzpy documentation. URL: https://xyzpy.readthedocs.io (visited on 2023-07-10).

[60] Matplotlib — Visualization with Python. URL: https://matplotlib.org/ (visited on 2023-07-10).

[61] HoloViews documentation. URL: https://holoviews.org/ (visited on 2023-07-10).

[62] hvPlot documentation. URL: https://hvplot.holoviz.org/index.html (visited on 2023-07-10).

[63] Bokeh. URL: https://bokeh.org/ (visited on 2023-07-10).

[64] Plotly. URL: https://plotly.com/python/ (visited on 2023-07-10).

[65] Seaborn documentation. URL: https://seaborn.pydata.org (visited on 2023-07-10).

[66] 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).

[67] Numba: A High Performance Python Compiler. URL: https://numba.pydata.org/ (visited on 2023-07-10).

[68] Docker: Accelerated, Containerized Application Development. May 2022. URL: https://www.docker.com/ (visited on 2023-07-10).

[69] Jupyter Docker Stacks documentation. URL: https://jupyter-docker-stacks.readthedocs.io/en/latest/index.html (visited on 2023-07-10).

[70] Jonathan Underwood. Limapack. March 2021. URL: https://github.com/jonathanunderwood/limapack (visited on 2023-07-10).

[71] 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.

[72] 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).

[73] Christopher Gerry and Peter Knight. *Introductory Quantum Optics*. Cambridge University Press, 2005.

[74] Richard N Zare. *Angular Momentum: Understanding Spatial Aspects in Chemistry and Physics*. John Wiley & Sons, 1988.

[75] 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.

[76] 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.

[77] Yasuki Arasaki, Kazuo Takatsuka, Kwanghsi 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.

[78] 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).

[79] 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).

[80] P R Bunker and P Jensen. *Molecular Symmetry and Spectroscopy*. NRC Research Press, Ottawa, 2nd edition, 1998.

[81] R Signorell and F Merkt. General symmetry selection rules for the photoionization of polyatomic molecules. *Molecular Physics*, 92(5):793–804, 1997.

[82] 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).

[83] 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).

[84] **missing booktitle in Kleinpoppen2005**

[85] 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.

[86] 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/JCPSA6/v112/i8/p3643/s1&Agg=doi.

[87] 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/JCPSA6/v113/i3/p1067/s1&Agg=doi (visited on 2012-07-17).

[88] 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.

[89] 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.

[90] 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).

[91] 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).

[92] 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).

[93] 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).

[94] 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).

[95] 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).

[96] QuTiP - Quantum Toolbox in Python. URL: https://qutip.org/ (visited on 2023-02-25).

[97] 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).

[98] 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.

[99] 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).

[100] Hirokazu Hasegawa and Yasuhiro Ohshima. Nonadiabatic Molecular Alignment and Orientation. In Kaoru Yamanouchi, 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).

[101] Christiane P Koch, Mikhail Lemeshko, and Dominique Sugny. Quantum control of molecular rotation. *Reviews of Modern Physics*, 91(3):035005, 2019.

[102] 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.

[103] 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).

[104] 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).

[105] Paul Hockett. General phenomenology of ionization from aligned molecular ensembles. *New Journal of Physics*, 17(2):023069, February 2015. URL: http://dx.doi.org/10.1088/1367-2630/17/2/023069http://stacks.iop.org/1367-2630/17/i=2/a=023069?key=crossref.bdbb6f53e1f801f11c6bfeca01330fde.

[106] 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.

[107] 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).

[108] 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).

[109] 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).

[110] \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).

[111] 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.

[112] Katharine L. Reid and Ivan Powis. Symmetry considerations in molecular photoionization: Fixed molecule photoelectron 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/100/2/10.1063/1.466638.

[113] **missing author in wikiSphericalHarmonics**

[114] 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.

[115] 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.

[116] 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.

[117] 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.

[118] 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).

[119] Varun Makhija, Xiaoming Ren, Drue Gockel, Anh-Thu Le, and Vinod Kumarappan. Orientation resolution through rotational coherence spectroscopy. *arXiv preprint arXiv:1611.06476*, 2016.

[120] 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.

[121] 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.

[122] 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.

[123] 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.

[124] Huynh Van Sa Lam, Suresh Yarlagadda, Anbu Venkatachalam, Tomthin Nganba Wangjam, Rajesh K Kushawaha, Chuan Cheng, Peter Svihra, 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.

[125] 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.

[126] 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.

[127] 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).

[128] 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).

[129] 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).

[130] 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).

[131] 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/.

[132] Achim Gelessus. Character tables for chemically important point groups. URL: http://symmetry.jacobs-university.de/ (visited on 2023-04-23).

[133] 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).

[134] L D Landau and E M Lifshitz. *Quantum Mechanics (Non-relativistic Theory)*. Pergamon Press, 3 edition, 1977.

[135] Albert Messiah. *Quantum Mechanics Volume I*. North-Holland Publishing Company, 1970.

[136] Jun John Sakurai. *Modern Quantum Mechanics*. Addison-Wesley, Reading, MA, revised edition edition, 1994. URL: https://cds.cern.ch/record/1167961.

INDEX