Quantum Metrology with Photoelectrons Vol. 3 *Analysis methodologies*

Paul Hockett

Mar 22, 2023

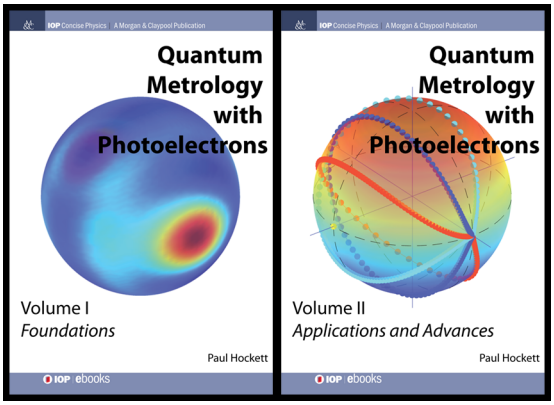
CONTENTS

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

Series abstract

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

About the books



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

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

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

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

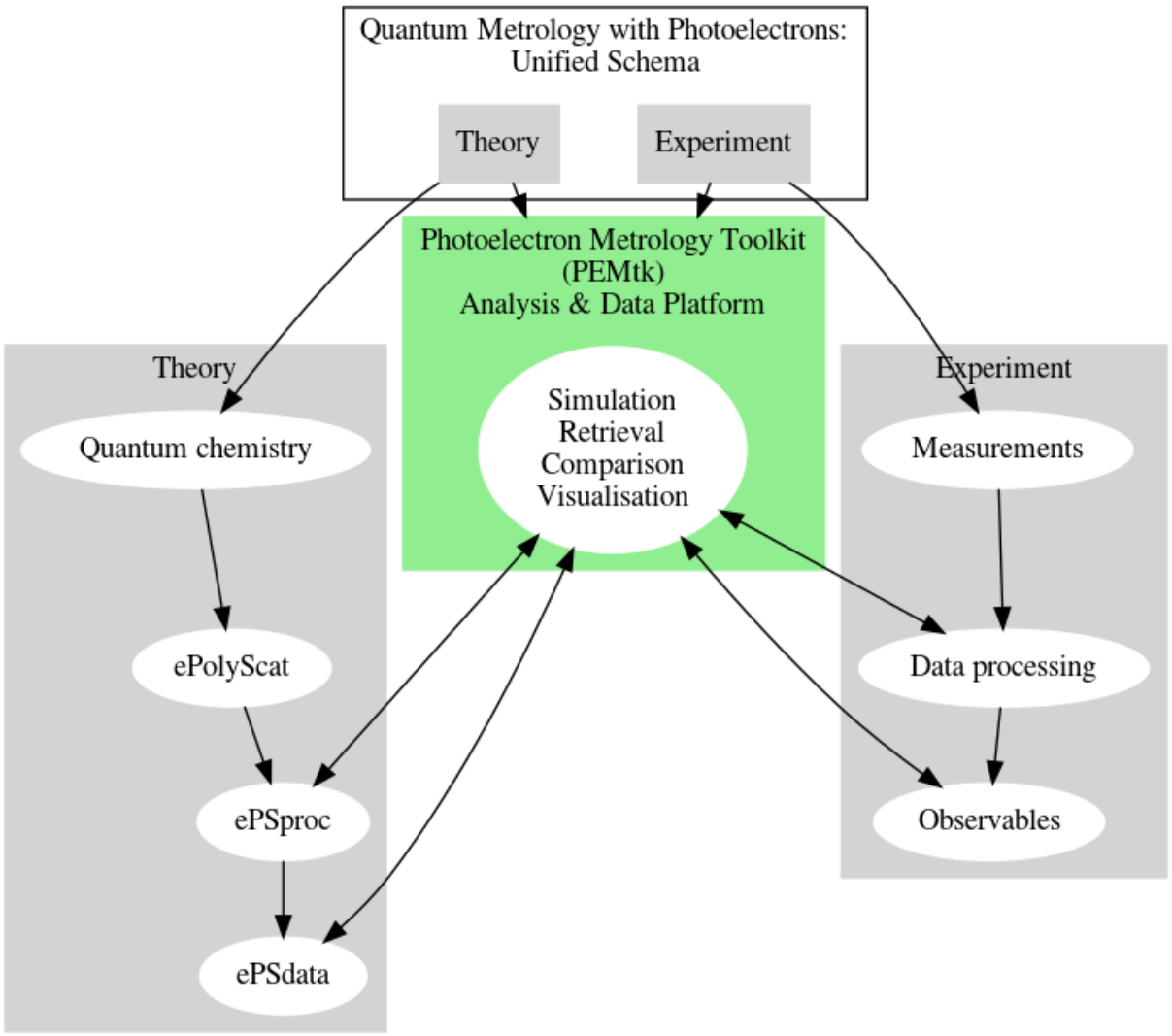
Technical details

This repository contains:

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

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

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



Running code examples

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

Building the book

The full book can also be built from source:

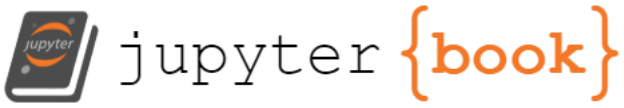
1. Clone this repository
2. Run `pip install -r requirements.txt` (it is recommended you do this within a virtual environment)
3. (Optional) Edit the books source files located in the `doc-source/` directory
4. Run `jupyter-book clean doc-source/` to remove any existing builds
5. For an HTML build:
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

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

Credits

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

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



Part I

Frontmatter

CHAPTER

ONE

OVERVIEW

1.1 General overview

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

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

1.2 Provisional contents

1.2.1 Part 1: theory & software

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

1. Introduction

   a. Topic overview.

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

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

2. Quantum metrology software platform/ecosystem overview

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

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

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

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

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

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

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

4. Numerical implementation & analysis platform tools

   a. Tensor methods implementation in ePSproc/PEMtk.

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

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

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

1.2.2 Part 2: numerical examples

Open-source worked examples using the new software platform.

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

   a. Experimental data overview & simulation.

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

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

   d. Comparison of methods.

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

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

   a. Experimental data overview & simulation.

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

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

   d. Comparison of methods.

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

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

   a. Experimental data overview & simulation.

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

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

   d. Comparison of methods.

   e. Information content/quantum information analysis.

4. Future directions & outlook

5. Summary & conclusions

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

Part II

Theory & software

CHAPTER

TWO

INTRODUCTION

The overall aim of *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 computational examples which may be used without extensive background knowledge, a brief introduction to the core physics and some recent extensions is also presented herein (Sect. XX BELOW). The unfamiliar reader is referred to *Quantum Metrology* Vol. 1 [1] 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 3), recent theory developments (Chapter 4), and concludes with a general overview for approaching matrix element retrieval numerically.

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

2.1 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 [1] are reproduced below; the reader is referred to the introductory chapters of *Quantum Metrology* Vol. 1 [1] for a lengthier treatment, and an introductory video to *Phase-sensitive Photoelectron Metrology* can be found online.

2.1.1 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 [1], 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 [1], Chpt. 1

2.1.2 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 [1, 2]). 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 [2]). 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 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 [2] (in particular Chpt. 11), and see also Ref. [3] for an experimental demonstration, and Ref. [] for a recent review in the context of molecular frame reconstruction.

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

> 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 in figure 8.3, and…] 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 [7]; 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 into the analysis.

---*Quantum Metrology* Vol. 2 [2], 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 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: a. 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. These calculations are required in order to determine the rotational wavepacket quantitatively, and in order to determine the corresponding *ADMs* from/for the experiment. b. 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. c. (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.

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 comparison of measured and simulated observables which may be integrated over many degrees of freedom of the system.

As detailed in the following section (Sect. 2.2), 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.

2.2 Context & aims for Vol. 3

2.2.1 Scientific aims

The work in the current volume primarily addresses recent developments towards a generalised bootstrapping protocol (i.e. the analysis of the data time-resolved photoelectron images from a rotationally-excited system), as previously outlined in *Quantum Metrology* Vol. 2 [2] Sect. 12.3; in particular the new Photoelectron Metrology Toolkit [4] 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, along with the theory and numerics implemented towards this end; Part II provides multiple demonstrations of the new codebase, 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 rotational wavepacket 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 [1, 2], e.g. use of shaped laser pulses, 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 Chpt. XX.

2.2.2 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 [1, 2] % Vols. 1 & 2 [1, 2] are more traditional publications. The material presented in this volume aims to continue the exploration of quantum metrology with photoelectrons, with a focus on numerical analysis techniques, forging a closer link between experimental and theoretical results, and making the methodologies discussed directly accessible via a new software platform/ecosystem, Photoelectron Metrology Toolkit [4]. %, introduced in more detail in Chapter 3. In order to fulfill this aim, Vol. 3 is a 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.[1]
    - These are `.ipynb` files, usually running a python kernel, 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 project platform,[2] which includes build tools and specifications for the specific flavour of Markdown (MyST) used for the written text.
    - 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 [4].
    - 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 3: *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. 3.4: *Docker deployments*. % Open Photoionization Docker Stacks [5]).
    - Once configured, any code examples from the book can be executed locally by the user/reader, and modified as desired.
3. The book can be regarded as, essentially, a manual and introduction to the Photoelectron Metrology Toolkit [4], 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.
    - 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.

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.

[1] For more information on the Jupyter Project and ecosystem, see jupyter.org and Refs. [6, 7, 8].)
[2] For more information see jupyterbook.org and Refs. [9, 10].

2.2.3 Open science, open source software and reproducibility

CHAPTER

THREE

QUANTUM METROLOGY SOFTWARE PLATFORM/ECOSYSTEM OVERVIEW

STUB

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

As of late 2022, the new parts of the platform - primarily the Photoelectron Metrology Toolkit [4] library - implement general data handling (although not a full experimental analysis toolchain), matrix element handling and retrieval, which will be the main topic of this volume. In the future, it is hoped that the platform will be extended to other theoretical and experimental methods, including full experimental data handling.

3.1 Analysis components

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

- The Photoelectron Metrology Toolkit [4] (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 [11].
- The ePSproc codebase [12, 13, 14] aims to provide methods for post-processing with *ab initio* radial dipole matrix elements from ePolyScat (ePS) [15, 16, 17, 18], or equivalent matrix elements from other sources (dedicated support for R-matrix results from the RMT suite [19, 20] is in development). The core functionality includes the computation of AF and MF observables. Manual computation without known matrix elements is also possible, e.g. for investigating limiting cases, or data analysis and fitting - hence these routines also provide the backend functionality for PEMtk fitting routines. Again more technical details can be found in the ePSproc documentation [14].

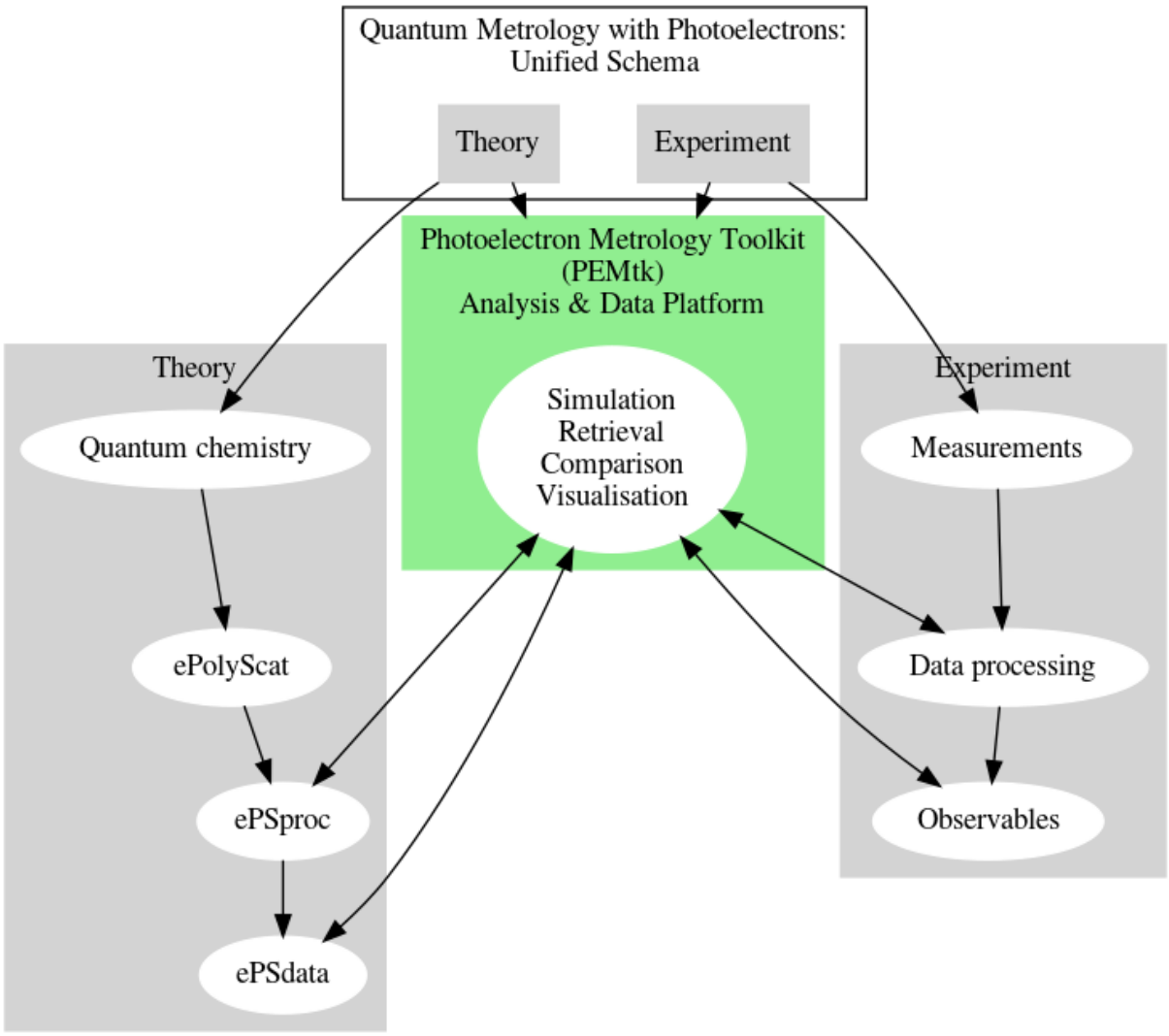


Fig. 3.1: Quantum metrology with photoelectrons ecosystem overview.

3.2 Additional tools

Other tools listed in Fig. 3.1 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") [21, 22] 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 [13] 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) [15, 16, 17, 18] 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 [18] and Refs. [15, 16, 17].
- ePSdata [23] 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 [23] Github project repository, and notebooks + datasets via ePSdata Zenodo [24]. 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.

3.3 Python ecosystem (backends, libraries and packages)

The core analysis tools, which constitute the Photoelectron Metrology Toolkit [4] 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` for general numerical methods and data types.
  - `pandas` for statistical methods, and various tabulation and sorting tasks.
  - `Scipy` for some special functions and computational routines, particularly spherical harmonics and fitting routines (see below).
- General tensor handling and manipulation makes use of the Xarray library [25, 26].
- Angular momentum functions (Wigner D and 3js) are currently implemented directly, or via the Spherical Functions library [27], and have been tested for consistency with the definitions in Zare (for details see the ePSproc docs [14]). The Spherical Functions library also uses `numpy_quaternion` 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 [28]. Further manipulation and conversion between different normalisations can be readily implemented with the SHtools library [29, 30].

- Non-linear optimization (fitting) is handled via the lmfit library, which implements and/or wraps a range of non-linear fitting routines in Python [31, 32]; for the Levenberg-Marquardt least-squares minimization method used herein this wraps Scipy's `least_squares` functionality, which therefore constituted the core minimization routine [28] for the demonstration cases.
- Symmetry functionality, specifically computing symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta,\phi)$ (see (4.1)), makes use of `libmsym` [33, 34] (symmetry coefficients) and `SHtools` [29, 30] (general spherical harmonic handling and conversion).
- 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 [13]: 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.
  - For general plotting a range of tools are used, or can be used, including `Matplotlib` (basic plotting, including `Xarray` plotters), `Holoviews` (data handling and interactive plotting, wraps various backends), `Bokeh` (implemented via Holoviews), `Plotly` (mainly used for spherical polar plotting), and `Seaborn` (for statistical and some specialist plots).
  - `Numba` 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).

Further comments, including conventions and numerical examples, can be found in Chpt. XX.

3.4 Docker deployments

A Docker-based distribution of various codes for tackling photoionization problems is also available from the Open Photoionization Docker Stacks [5] 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 `ePS`, `ePSproc` and `PEMtk`.

3.5 General discussion

Note that, at the time of writing, rotational wavepacket simulation is not yet implemented in the PEMtk suite, and these must be obtained via other codes. An intial build of the `limapack` suite for rotational wavepacket simulations is currently part of the Open Photoionization Docker Stacks [5], but has yet to be tested.

CHAPTER

FOUR

THEORY

- *Observables: photoelectron flux in the LF and MF*
- *Photoionization dynamics*
- *Tensor formulation of photoionization*
- *Density matrix representation*
- *Information content & sensitivity*

4.1 Observables: photoelectron flux in the LF and MF

The observables of interest - the photoelectron flux as a function of energy, ejection angle, and time - 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.

4.1.1 Spherical harmonics

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

$$\bar{I}(\epsilon, t, \theta, \phi) = \sum_{L=0}^{2n} \sum_{M=-L}^{L} \bar{\beta}_{L,M}(\epsilon, t) Y_{L,M}(\theta, \phi) \tag{4.1}$$

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*. Some arbitrary examples are given in Fig. 4.1, 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. 4.2.

```
# Plot some distributions from specified BLMs

# Set specific LM coeffs by list with setBLMs, items are [l,m,value]
from epsproc.sphCalc import setBLMs

# BLM = setBLMs([[0,0,1],[1,1,1],[2,2,1]])
# BLM = setBLMs([[0,0,1,1,1],[1,1,1,0.5,0.2],[2,2,1,1,0.2]])   # Note different index
```

(continues on next page)

(continued from previous page)

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

# Set the backend to 'pl' for an interactive surface plot with Plotly
# NOTE PL FIG RETURN BROKEN FOR THIS CASE (ePSproc v1.3.1), so run sphSumPlotX too.
dataPlot, figObj = ep.sphFromBLMPlot(BLM, facetDim='t', plotFlag = False, backend =␣
 ↪plotBackend);
figObj = ep.sphSumPlotX(dataPlot,facetDim='t', plotFlag = False, backend =␣
 ↪plotBackend);

# And GLUE for display later with caption
# from myst_nb import glue
# glue("padExamplePlot", figObj[0], display=False);
# Glue with Plotly wrapper.
# gluePlotly("padExamplePlot", figObj[0])   # Working in Render test notebook, but␣
 ↪not here? Issue with subplots?

# Test in separate cell...
# gluePlotly("padExamplePlot", figObj[0])   # Working in Render test notebook, but␣
 ↪not here? Issue with subplots?

# With additional layout settings - defaults give cropped subplots?
gluePlotly("padExamplePlot", figObj[0].update_layout(height=1400, width=1400))
```

In general, the spherical harmonic rank and order (L, M) of Eq. (4.1) 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 [35]), but in the *MF* is defined by the maximum continuum angular momentum $n = l_{max}$ imparted by the scattering event [36] (note lower-case l here refers specifically to the continuum photoelectron wavefunction, see Eq. (4.7)).

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. [1] for more discussion and complex examples.

In general, these observables may also be dependent on various other parameters; in Eq. (4.1) 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 below (Sect. 4.2), 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. 4.1.

4.1.2 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 (Refs. [37, 38, 39] as below):

$$X_{hl}^{\Gamma\mu*}(\theta,\phi) = \sum_{\lambda} b_{hl\lambda}^{\Gamma\mu} Y_{l,\lambda}(\theta,\phi) \tag{4.1}$$

where:

- Γ is an irreducible representation;

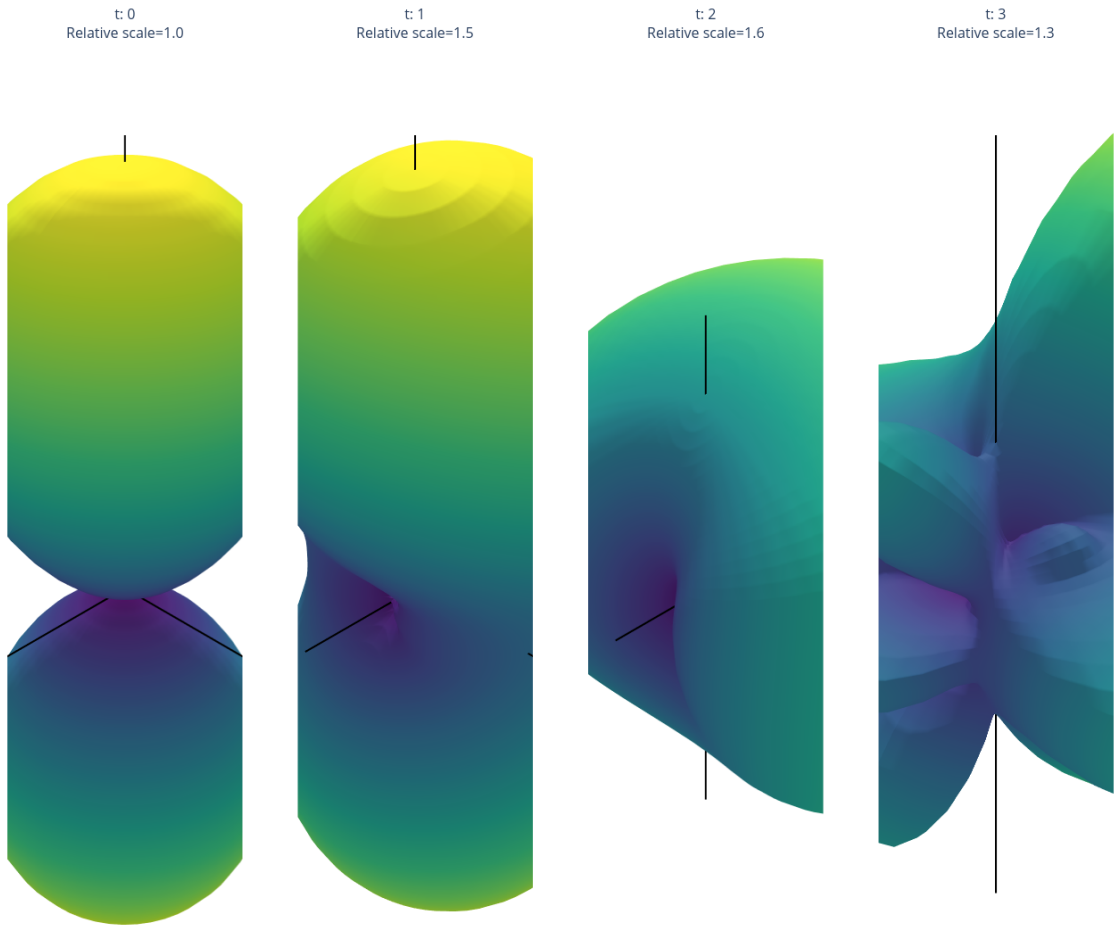


Fig. 4.1: Examples of angular distributions (expansions in spherical harmonics $Y_{L,M}$), for a range of cases. Note that up-down asymmetry is associated with odd-l contributions, and breaking of cylindrical symmetry with $m \neq 0$ terms.

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. 4.2: Values used for the plots in Fig. 4.1.

- (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. 4.3.2 - in ambiguous cases the symmetrization term will instead be labelled 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. (4.1), a general expansion of an observable in the symmetrized harmonic basis set can then be defined as:

$$\bar{I}^{\Gamma}(\epsilon, t, \theta, \phi) = \sum_{\Gamma\mu hl} \bar{\beta}_{hl}^{\Gamma\mu}(\epsilon, t) X_{hl}^{\Gamma\mu *}(\theta, \phi) \tag{4.2}$$

Alternatively, by substitution into Eq. (4.1), and assigning $l = L$ and $\lambda = M$, a general symmetrized expansion may also be defined as:

$$\bar{I}(\epsilon, t, \theta, \phi) = \sum_{\Gamma\mu h} \sum_{L=0}^{2n} \sum_{M=-L}^{L} b_{hLM}^{\Gamma\mu} \bar{\beta}_{L,M}(\epsilon, t) Y_{L,M}(\theta, \phi) \tag{4.3}$$

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:

$$\bar{I}^{\Gamma}(\epsilon, t, \theta, \phi) = \sum_{L=0}^{2n} \sum_{M=-L}^{L} \bar{\beta}_{L,M}^{\Gamma}(\epsilon, t) Y_{L,M}(\theta, \phi) \tag{4.3}$$

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. (4.1) (e.g. for cases with degenerate components). Further details and usage in relation to channel functions are also discussed in Sect. 4.3 (see, in particular, Eq. (4.12) 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. [39, 40]. Numerical routines for the generation of symmetrized harmonics are implemented in Photoelectron Metrology Toolkit [4]: point-groups, character table generation and symmetrization (computing $b_{hl\lambda}^{\Gamma\mu}$ parameters) is handled by libmsym [33, 34]; additional handling also makes use of pySHtools [29, 30].

A brief numerical example is given below, for Td symmetry (l_{max} =6), and more details can be found in the PEMtk documentation [11]. In this case, full tabulations of the parameters lists all $b_{hLM}^{\Gamma\mu}$ for each irreducible representation, and the corresponding PADs are illustrated in Fig. 4.4.

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()

# Glue items for later
glue("symHarmPG", sym, display=False)
glue("symHarmLmax", lmax, display=False)
glue("charTab",symObj.printCharacterTable(returnPD=True), display=False)  # As above,␣
↪but with PD object return and glue.
```

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. 4.3: Example character table for Td symmetry generated with the Photoelectron Metrology Toolkit [4] wrapper for libmsym [33, 34].

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

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

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

```
# To plot using ePSproc/PEMtk class, these values can be converted to ePSproc BLM␣
↪data type...

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

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

(continues on next page)

(continued from previous page)

```
for dataType in ['BLM']:  #['matE','BLM']:
    data.data['symHarm'][dataType] = symObj.coeffs[dataType]['b (comp)']  # Select␣
 ↪expansion in complex harmonics
    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)

rc = [2,3]  # Explict layout setting
data.padPlot(keys='symHarm',dataType='BLM', facetDims = ['Cont'], squeeze = True,␣
 ↪backend=plotBackend, rc = rc, plotFlag=False, returnFlag=True)  # Working
figObj = data.data['symHarm']['plots']['BLM']['polar'][0]

# And GLUE for display later with caption
# from myst_nb import glue
# glue("padExamplePlot2", figObj, display=False);
gluePlotly("symHarmPADs", figObj)
```

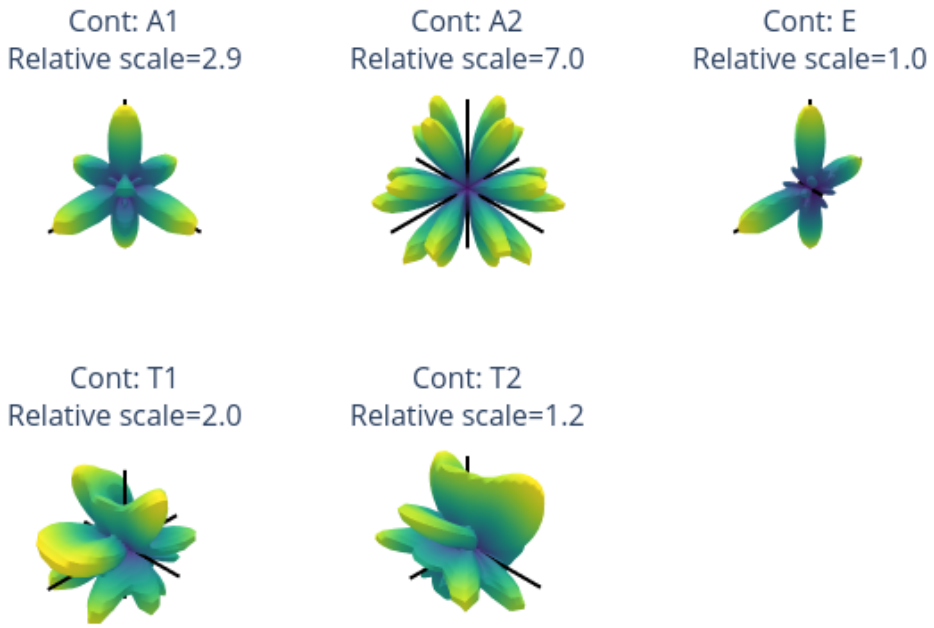


Fig. 4.4: Examples of angular distributions from expansions in symmetrized harmonics $X_{hl}^{\Gamma\mu*}(\theta,\phi)$, for all irreducible representations in Td symmetry (l_{max} =6). (Note A_2 only has components for $l \geq 6$.)

4.1.3 Real harmonics

4.1.4 Legendre polynomials

```
!date
```

```
Wed 22 Mar 2023 01:29:54 PM EDT
```

4.2 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 Vol. 1 [1] (and refs. 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 $\langle\Psi_i|\hat{\Gamma}(\mathbf{E})|\Psi_f\rangle$, 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 purturbative), dipolar regime, resulting in a single photoelectron. (For more discussion of various approximations in photoionzation, see Refs. [41, 42].) 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 a well-defined problem, making use of analytic terms in combination with fitting methodologies as explored herein. Again, more extensive background and discussion can be found in *Quantum Metrology* Vol. 1 [1], and references therein. % [TODO: Add some more refs here?]

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,

$$\left|\Psi_f\right\rangle = \sum \int \left|\Psi_+;\mathbf{k}\right\rangle \left\langle\Psi_+;\mathbf{k}|\Psi_f\right\rangle d\mathbf{k}, \tag{4.3}$$

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

$$\hat{\Gamma}(\mathbf{E}) = \hat{\mu}.\mathbf{E} \tag{4.4}$$

Hence,

$$\left\langle\Psi_+;\mathbf{k}|\Psi_f\right\rangle = \langle\Psi_+;\,\mathbf{k}|\hat{\mu}.\mathbf{E}|\Psi_i\rangle \tag{4.5}$$

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 [43], which includes a sum over all electrons s defined in position space as $\mathbf{r_s}$:

$$\hat{\mu} = -e \sum_s \mathbf{r_s} \tag{4.6}$$

The position space photoelectron wavefunction is typically expressed in the "partial wave" basis, expanded as (asymptotic) continuum eignstates of orbital angular momentum, with angular momentum components (l, m) (note lower case notation for the partial wave components, distinct from upper-case for the similar terms (L, M) in the observables),

$$\Psi_{\mathbf{k}}(r) \equiv \langle r|\mathbf{k}\rangle = \sum_{lm} Y_{lm}(\hat{\mathbf{k}})\psi_{lm}(r, k) \tag{4.7}$$

where r are MF electronic coordinates and $Y_{lm}(\hat{\mathbf{k}})$ are the spherical harmonics.

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. [44] for general discussion), and written generally as (using notation similar to [45]):

$$\langle \Psi_+; \mathbf{k}|\hat{\mu}.\mathbf{E}|\Psi_i\rangle = \sum_{lm} \gamma_{l,m}\mathbf{r}_{k,l,m} \tag{4.8}$$

Provided that 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. (4.8) and other angular-momentum coupling terms - are know, knowledge of the so-called radial (or reduced) dipole matrix elements, at a given k thus equates to a full description of the system dynamics (and, hence, the observables).

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:

$$\mathbf{r}_{k,l,m} = \int \psi_{lm}(r, k) r \Psi_i(r) dr \tag{4.9}$$

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. 4.3.2 for details).

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

$$I(\epsilon, \theta, \phi) = \langle \Psi_f | \Psi_f \rangle \tag{4.10}$$

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 [1], 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 [1]):

$$I(\theta, \phi;\, k) = \sum_{ll'} \sum_{\lambda\lambda'} \sum_{mm'} \gamma_{\alpha\alpha_+ l\lambda m l'\lambda' m'} r_{kl\lambda} r_{kl'\lambda'} e^{i(\eta_{l\lambda}(k) - \eta_{l'\lambda'}(k))} Y_{lm}(\hat{k}) Y^*_{l'm'}(\hat{k}) \tag{4.11}$$

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 terms denotes a general set of geometric paramters arising from the coherent square. A tensor form is also given herein, see Sect. 4.3.2, including a full breakdown of these terms and numerical implementation. Comparison with Eq. (4.1) then indicates that the amplitudes in Eq. (4.8) also determine the observable anisotropy parameters $\beta_{L,M}(\epsilon, t)$ (Eqn. (4.1)), which basically collect all the terms in Eq. (4.11) and the product over spherical harmonics, into a result set of (L, M). (Note that the photoelectron energy ϵ and momentum k are used somewhat interchangeably herein, with the former usually preferred in reference to observables.)

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. 4.3.2). The radial matrix elements are a sensitive function of molecular geometry and electronic configuration in general, hence may be considered to be responsive to molecular dynamics, although they are formally time-independent in a Born-Oppenheimer basis - for further general discussion and examples see Ref. [46] and *Quantum Metrology* Vol. 1 [1]; discussions of more complex cases with electronic and nuclear dynamics can be found in Refs. [42, 47, 48, 49].

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 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 matrix elements - as emphasised in the magnitude-phase form of Eq. (4.11) - and can be considered as angular interferograms.

4.3 Tensor formulation of photoionization

A number of authors have treated MFPADs and related problems [REFS]; herein, a geometric tensor based formalism is developed, which is close in spirit to the treatments given by Underwood and co-workers [49, 50, 51], 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. Furthermore, the tensors can readily be converted to a density matrix representation [44, 52], 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 [52] and Zare [44].

Within this treatment, the observables can be defined in a series of simplified forms, emphasizing the quantities of interest for a given problem. Some details are defined in the following subsections,

4.3.1 Channel functions

A simple form of the equations (cf. the general form of Eq. (4.11), see also *Quantum Metrology* Vol. 2 [2] Chpt. 12), 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),

$$\beta^{u}_{L,M} = \sum_{\zeta,\zeta'} \Upsilon^{u,\zeta\zeta'}_{L,M} \mathbb{I}^{\zeta\zeta'} \tag{4.12}$$

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:

$$\mathbb{I}^{\zeta,\zeta} = I^{\zeta}(\epsilon) I^{\zeta'*}(\epsilon) \tag{4.13}$$

This is effectively a convolution equation (cf. refs. [50, 53]) with channel functions $\Upsilon^{u,\zeta\zeta'}_{L,M}$, 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. [15, 16, 17]),

these matrix elements are essentially identical to the simplified (radial) forms $\mathbf{r}_{k,l,m}$ defined in Eqn. (4.8), in the case where $\zeta = k, l, m$. Note, also, that the matrix elements used herein are usually assumed to be symmetrized (unless explicitly stated), i.e. expanded in spherical harmonics per Eq. (4.1) with any additional terms $b^{\Gamma\mu}_{hl\lambda}$ incorporated into the value of the matrix elements.

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

$$I^{\zeta}(\epsilon) \equiv \mathbf{r}_{\zeta} \equiv r_{\zeta} e^{i\phi_{\zeta}} \tag{4.14}$$

This tensorial form is numerically implemented in the ePSproc [13] codebase, and is in contradistinction to standard numerical routines in which the requisite terms are usually computed from vectorial and/or nested summations, which can be somewhat opaque to detailed interpretation, and typically implement the full computation of the observables in one monolithic computational routine. The Photoelectron Metrology Toolkit [4] codebase implements matrix element 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. (4.12), with terms $I^{\zeta}(\epsilon)$ as the unknowns (in magnitude, phase form per (4.14)). 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 investigation of the properties of the channel functions for a given problem in general terms, as well as for specific experimental cases.

4.3.2 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. (4.12) in terms of the relevant tensors. Further details and numerical examples are given in the following sub-sections.

For the MF:

$$\begin{aligned}\beta_{L,-M}^{\mu_i,\mu_f}(\epsilon) = & (-1)^M \sum_{P,R',R} [P]^{\frac{1}{2}} E_{P-R}(\hat{e};\mu_0) \\ \times & \sum_{l,m,\mu} \sum_{l',m',\mu'} (-1)^{(\mu'-\mu_0)} \Lambda_{R',R}(R_{\hat{n}};\mu,P,R,R') B_{L,-M}(l,l',m,m') \\ \times & I_{l,m,\mu}^{p_i\mu_i,p_f\mu_f}(\epsilon) I_{l',m',\mu'}^{p_i\mu_i,p_f\mu_f *}(\epsilon)\end{aligned} \tag{4.15}$$

And the LF/AF as:

$$\begin{aligned}\bar{\beta}_{L,-M}^{\mu_i,\mu_f}(E,t) = & (-1)^M \sum_{P,R',R} [P]^{\frac{1}{2}} E_{P-R}(\hat{e};\mu_0) \\ \times & \sum_{l,m,\mu} \sum_{l',m',\mu'} (-1)^{(\mu'-\mu_0)} \bar{\Lambda}_{R'}(\mu,P,R') B_{L,S-R'}(l,l',m,m') \\ \times & I_{l,m,\mu}^{p_i\mu_i,p_f\mu_f}(\epsilon) I_{l',m',\mu'}^{p_i\mu_i,p_f\mu_f *}(\epsilon) \sum_{K,Q,S} \Delta_{L,M}(K,Q,S) A_{Q,S}^{K}(t)\end{aligned} \tag{4.16}$$

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 (*AF* only).
- $A_{Q,S}^{K}(t)$: ensemble alignment described as a set of *axis distribution moments* (*ADMs*, *AF* only).
- Square-brackets indicate degeneracy terms, e.g. $[P]^{\frac{1}{2}} = (2P+1)^{\frac{1}{2}}$.

And $I_{l,m,\mu}^{p_i\mu_i,p_f\mu_f}(\epsilon)$ are the (radial) dipole ionization matrix elements, as a function of energy ϵ. These matrix elements are essentially identical to the simplified forms $r_{k,l,m}$ defined in Eqn. (4.8), except 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) [15, 16, 17, 18], 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. (4.12) for the *AF* and *MF* can be written as:

$$\begin{aligned}\Upsilon_{L,M}^{u,\zeta\zeta'} = \quad & (-1)^M[P]^{\frac{1}{2}}E_{P-R}(\hat{e};\mu_0)(-1)^{(\mu'-\mu_0)}\Lambda_{R',R}(R_{\hat{n}};\mu,P,R,R') \\ \times \quad & B_{L,-M}(l,l',m,m')\end{aligned} \tag{4.17}$$

$$\begin{aligned}\bar{\Upsilon}_{L,M}^{u,\zeta\zeta'} = \quad & (-1)^M[P]^{\frac{1}{2}}E_{P-R}(\hat{e};\mu_0)(-1)^{(\mu'-\mu_0)}\bar{\Lambda}_{R'}(\mu,P,R') \\ \times \quad & B_{L,S-R'}(l,l',m,m')\Delta_{L,M}(K,Q,S)A_{Q,S}^{K}(t)\end{aligned} \tag{4.18}$$

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

It should be emphasized, however, that the underlying physical quantities are essentially identical in all the theoretical approaches, with a set of coupled angular-momenta defining the geometrical 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 [12, 13, 14], and further wrapped for fitting cases in the Photoelectron Metrology Toolkit [4]. 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 [14], including extended discussion of each tensor and complete function references in the geomCalc submodule documentation.

4.3.3 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 some of the geometric terms and are discussed where required.

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 (4.1)) have the property that $\mu = 0$ only, and the h index is redundant, since it maps uniquely to l - see Fig. 4.5 - 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
```

(continues on next page)

(continued from previous page)

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

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

# Glue items for later
glue("symHarmPGmatE", sym, display=False)
glue("symHarmLmaxmatE", lmax, display=False)
glue("symHarmBasislmaxPlot", lmaxPlot, display=False)

# 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
```

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

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

```
# ep.matEleSelector(data.data[dataKey]['matE'], thres = None, inds = {}, dims = 'Eke',
 ↪ sq = True, drop = True)
```

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. 4.5: Symmetrized harmonics coefficients ($b_{hl\lambda}^{\Gamma\mu}$, see (4.1)) for D2h symmetry (l_{max} =4) generated with the Photoelectron Metrology Toolkit [4] wrapper for libmsym [33, 34]. Note that, in this case, the coeffcients have the property that $\mu = 0$ only, and the h index is redundant (maps uniquely to l).

```
# 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
```

```
Product basis elements: dict_keys(['BLMtableResort', 'polProd', 'phaseConvention',
↪'BLMRenorm'])
Full basis elements: dict_keys(['QNs', 'EPRX', 'lambdaTerm', 'BLMtable',
↪'BLMtableResort', 'AFterm', 'AKQS', 'polProd', 'phaseConvention', 'BLMRenorm'])
```

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

The coupling of the partial wave 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.

$$B_{L,M} = (-1)^m \left(\frac{(2l+1)(2l'+1)(2L+1)}{4\pi}\right)^{1/2} \begin{pmatrix} l & l' & L \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} l & l' & L \\ -m & m' & M \end{pmatrix} \tag{4.19}$$

Note that this term is equivalent, effectively, to a triple integral over spherical harmonics (e.g. Eq. 3.119 in Zare [44]):

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

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 [52]) - 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 [52]:

$$\beta^{\mu_i,\mu_f}_{L,M} Y^*_{LM}(\theta_{\hat{k}},\phi_{\hat{k}}) = \beta^{\mu_i,\mu_f}_{L,-M}(-1)^M Y_{L,-M}(\theta_{\hat{k}},\phi_{\hat{k}}) \tag{4.20}$$

In the current Photoelectron Metrology Toolkit [4] 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. Note for the *AF* case the terms may be reindexed by $M = S - R'$ - this allows for all *MF* projections to contribute, rather than just a single specified polarization geometry. However, in many typical cases, this term is nonetheless 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. 4.6 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. (4.19)), 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 (use ep.multiDimXrToPD, or other? Should have wrappers...?)

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
pd, _ = ep.multiDimXrToPD(basisPlot, colDims=stackDims)

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

```
#*** Plot BLM terms for basis set - basic
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

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

labelRound = 1
catLegend=False
titleString=''
titleDetails=False
labelCols = [1,1]

# With global formatting args
# ep.lmPlot(basis[basisKey].where((basis[basisKey].l<=lmaxPlot) & (basis[basisKey].lp
 ↪<=lmaxPlot)).rename({'S-Rp':'M'}).stack({'LM':['L','M']}), xDim={'LM':['L','M']},
#           cmap=cmap, labelRound = labelRound, catLegend=catLegend,␣
 ↪titleString=titleString, titleDetails=titleDetails, labelCols = labelCols);
```

(continues on next page)

(continued from previous page)

```
# With fig return - NEEDS WORK, displays but doesn't return fig object?
# UPDATE 28/11/22: still displays, but now have glue() working correctly.
# daPlot, daPlotpd, legendList, gFig = ep.lmPlot(basis[basisKey].
↪where((basis[basisKey].l<=lmaxPlot) & (basis[basisKey].lp<=lmaxPlot)).rename({'S-Rp
↪':'M'}).stack({'LM':['L','M']}), xDim={'LM':['L','M']},
#           pType = 'r', cmap=cmap, labelRound = labelRound, catLegend=catLegend,␣
↪titleString=titleString, titleDetails=titleDetails, labelCols = labelCols);

# 30/11/22 simplified version
daPlot, daPlotpd, legendList, 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);

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

4.3.5 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. (4.11)) can be written as a tensor contraction:

$$E_{PR}(\hat{e}) = [e \otimes e^*]^P_R = [P]^{\frac{1}{2}} \sum_p (-1)^R \begin{pmatrix} 1 & 1 & P \\ p & R-p & -R \end{pmatrix} e_p e^*_{R-p} \tag{4.21}$$

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. $[P]^{\frac{1}{2}} = (2P+1)^{\frac{1}{2}}$;
- the symbol μ_0 is conventionally used to denote the *LF* field projection definition, given in full as $(1, \mu_0)$ for the 1-photon case, **in general for the LF/AF case $\mu_0 = p$, while for the MF case all projection terms are allowed, and are usually labelled by μ or q.** TO FIX

(To derive this result, one can start from, e.g., Eq. 5.40 in Zare

$$[A^{(1)} \otimes B^{(1)}]^k_q = \sum_m \langle 1m, 1q-m | kq \rangle A(1,m) B(1,q-m) \tag{4.22}$$

Convert to $3j$ form:

$$[A^{(1)} \otimes B^{(1)}]^k_q = \sum_m (-1)^q [k]^{1/2} \begin{pmatrix} 1 & 1 & k \\ m & q-m & -q \end{pmatrix} A(1,m) B(1,q-m) \tag{4.23}$$

And substitute in appropriate terms.)

As before, we can visualise these values…

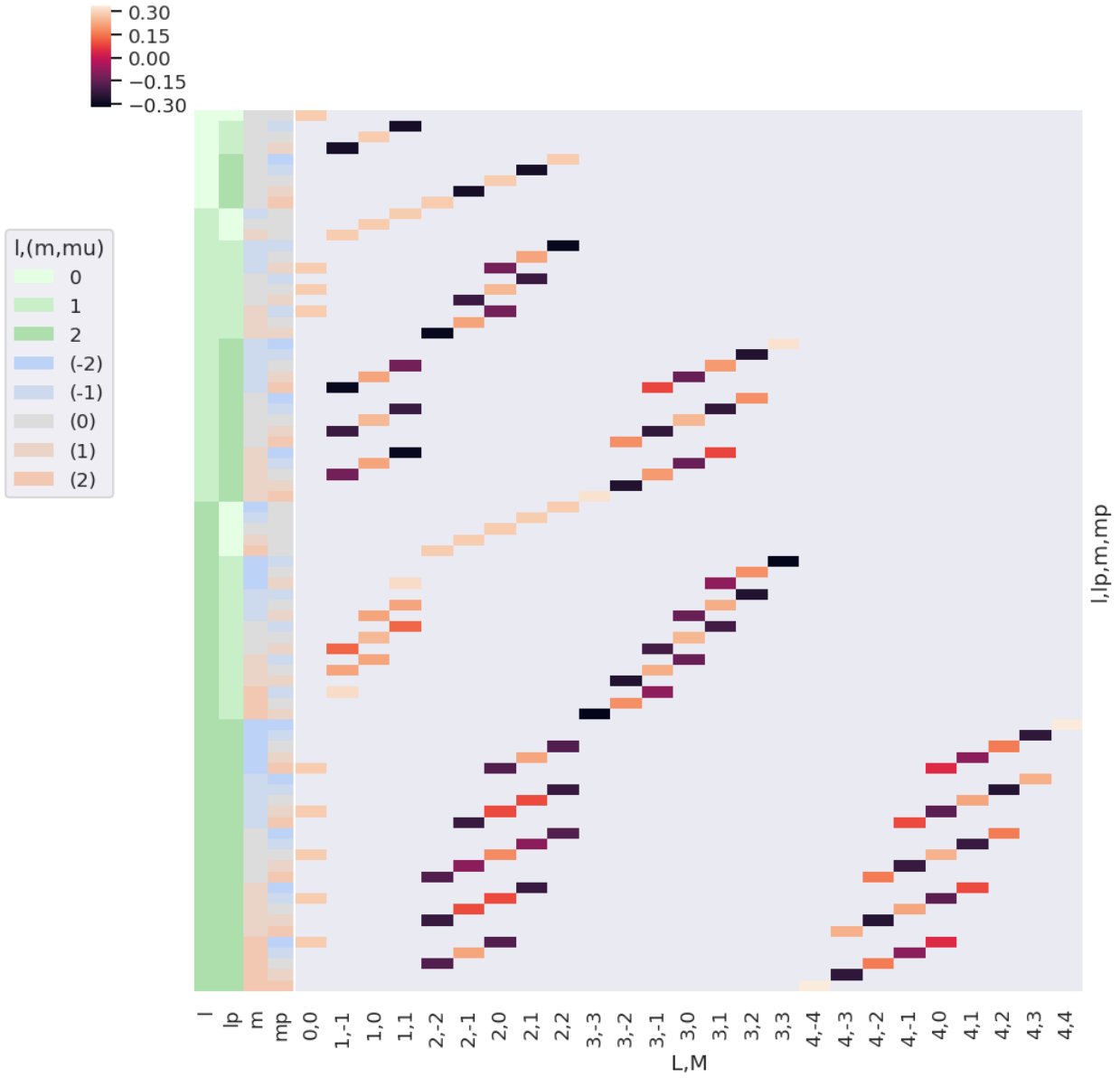


Fig. 4.6: Example $B_{L,M}$ basis functions for D2h symmetry. Note figure is truncated to $l_{max} = l'_{max} = 2$ for clarity.

```
# 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 = ['l', 'p', 'lp', 'R-p']
plotDimsRed = ['p', 'R-p']
xDim = {'PR':['P','R']}

# Plot with ep.lmPlot(), real values
daPlot, daPlotpd, legendList, gFig = ep.lmPlot(EPRX, plotDims=plotDimsRed, xDim=xDim,␣
↪pType = 'r')
# Version summed over l,m
# daPlot, daPlotpd, legendList, gFig = ep.lmPlot(EPRX.unstack().sum(['l','lp','R-p']),
↪ xDim=xDim, pType = 'r')

# For glue
glue("lmPlot_EPR_basis", gFig.fig, display=False)
```

4.3.6 Molecular frame projection term Λ

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

$$\Lambda_{R',R}(R_{\hat{n}}) = (-1)^{(R')} \begin{pmatrix} 1 & 1 & P \\ \mu & -\mu' & R' \end{pmatrix} D^{P}_{-R',-R}(R_{\hat{n}}) \tag{4.24}$$

This is similar to the E_{PR} 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 $R_{\hat{n}}$.

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

$$\bar{\Lambda}_{R'} = (-1)^{(R')} \begin{pmatrix} 1 & 1 & P \\ \mu & -\mu' & R' \end{pmatrix} \equiv \Lambda_{R',R'}(R_{\hat{n}} = 0) \tag{4.25}$$

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. 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_{PR} definition.

TODO: frame rotation illustration as well as term plots? Cf. QM1?

4.3.7 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*) and the photoionization multipole terms into the final observable.

$$\Delta_{L,M}(K, Q, S) = (2K + 1)^{1/2}(-1)^{K+Q} \begin{pmatrix} P & K & L \\ R & -Q & -M \end{pmatrix} \begin{pmatrix} P & K & L \\ R' & -S & S - R' \end{pmatrix} \tag{4.26}$$

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

$$\tilde{\Delta}_{L,M}(t) = \sum_{K,Q,S} \Delta_{L,M}(K, Q, S) A^{K}_{Q,S}(t) \tag{4.27}$$

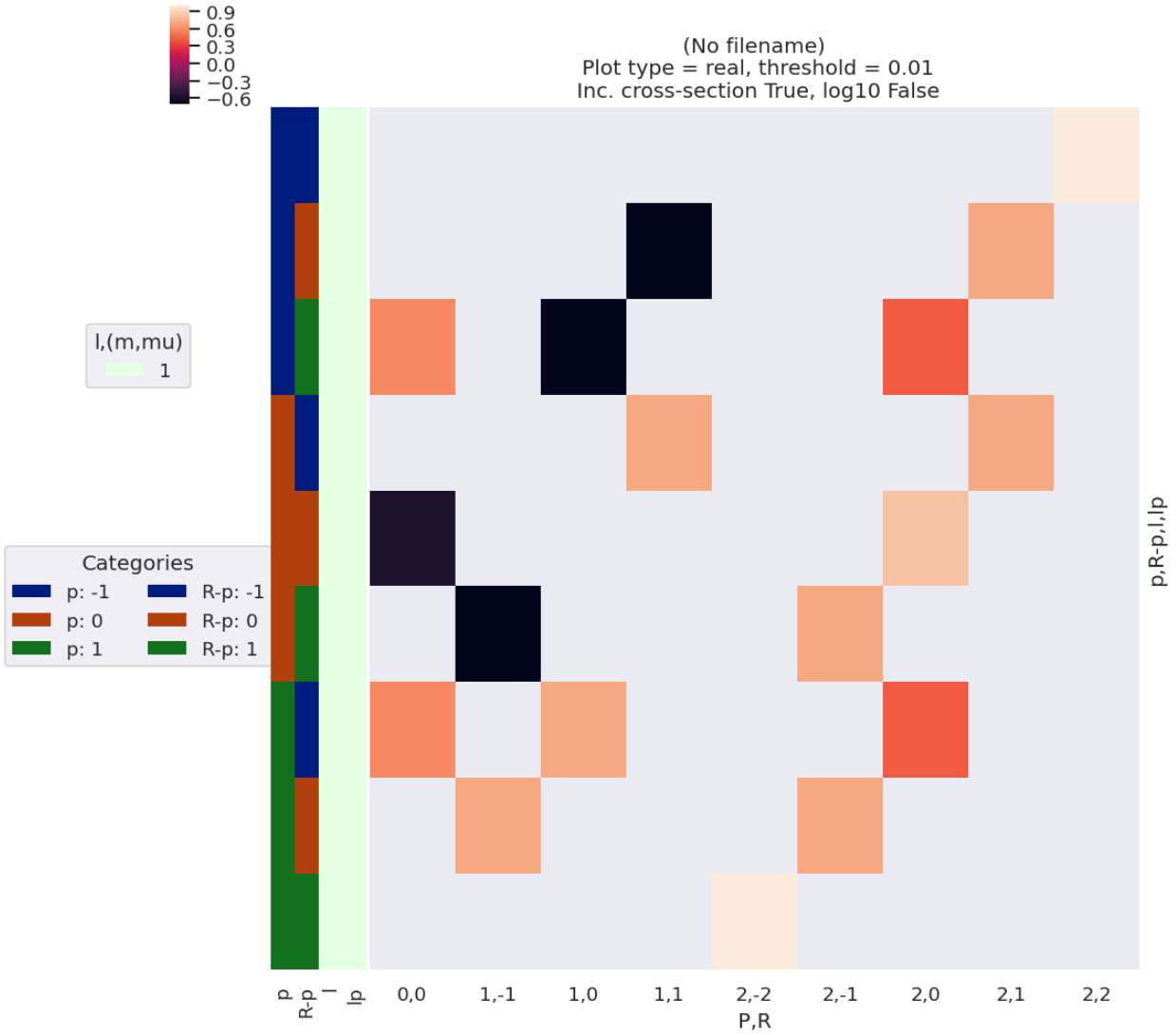


Fig. 4.7: Example $E_{P,R}$ basis functions. Note that for linearly polarised light $p = R - p = 0$ only, hence only the terms $E_{0,0}$ and $E_{2,0}$ are non-zero in this case. For non-linearly polarised cases many other terms are allowed.

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. [1, 51]); 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. 4.8 illustrates the alignment tensor $\Delta_{L,M}(K,Q,S)$ for some basic cases, and values are also tabulated in Fig. 4.9. 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 elliptically polarized light, mixing of terms with different p allows for non-zero R terms, hence non-zero M is allowed, signifying breaking of cylindrical symmetry in the observable.

```
#*** Set range of ADMs for test as time-dependent values - single call
# AKQS = ep.setADMs(ADMs = [[0,0,0,1,1,1,1],
#                           [2,0,0,0,0.5,1,1],
#                           [4,0,0,0,0,0,0.3]])  #, t=[0,1,2])    # Nested list or np.
 ↪array OK.
# AKQS = ep.setADMs(ADMs = np.array([[0,0,0,1,1],[2,0,0,0,0.5]]), t=[0,1])

#*** Alternative form with ADM inds separate
# ADMinds = np.array([[0,0,0],[2,0,0],[4,0,0]])
# AKQS = ep.setADMs(ADMs = np.stack((np.ones(4),np.linspace(0,1,4),np.linspace(0,0.2,
 ↪4))),
#                   KQSLabels=ADMinds)


# For computation set ADMs and EPR term first.
# 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.

#*** Neater version
# Set ADMs for increasing alignment...
tPoints = 10
inputADMs = [[0,0,0, *np.ones(tPoints)],     # * np.sqrt(4*np.pi)],  # Optional␣
 ↪multiplier for normalisation
             [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)]]

AKQS = ep.setADMs(ADMs = inputADMs)  # TODO WRAP TO CLASS (IF NOT ALREADY!)


#*** Plot ADMs
# daPlot, daPlotpd, legendList, gFig = ep.lmPlot(AKQS, xDim = 't', pType = 'r',␣
 ↪squeeze = False, thres=None, cmap='vlag')  # Note squeeze = False required for 1D␣
 ↪case (should add this to code!)
```

(continues on next page)

(continued from previous page)

```
# daPlotpd

# Example alignment term, assuming all other terms allow/unity valued.

# 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
# xDim = {'KQ':['K','Q']}
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) #, cmap='vlag')
 ↪# , fillna=True)  # Note squeeze = False required for 1D case (should add this to␣
 ↪code!)
# daPlotpd.fillna('')

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

```
#*** Plot ADMs
daPlot, daPlotpd, legendList, ADMFig = ep.lmPlot(AKQS, xDim = 't', pType = 'r',␣
 ↪squeeze = False, cmap='vlag')  # Note squeeze = False required for 1D case (should␣
 ↪add this to code!)
# daPlotpd

#*** Plot subsection
daPlot, daPlotpd, legendList, AFFig = ep.lmPlot(AFterm.sel(R=0).sel(Rp=0).sel({'S-Rp
 ↪':0}),
                                                xDim = 't', pType = 'r', squeeze =␣
 ↪False, cmap='vlag')  # Note squeeze = False required for 1D case (should add this␣
 ↪to code!)

# 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 [2], and in the current case the focus is purely on rotational wavepackets.

Fig. 4.11 shows $\tilde{\Delta}_{L,M}(t)$ for various choices of alignment (as per the *ADMs* shown in Fig. 4.10), and illustrates some of

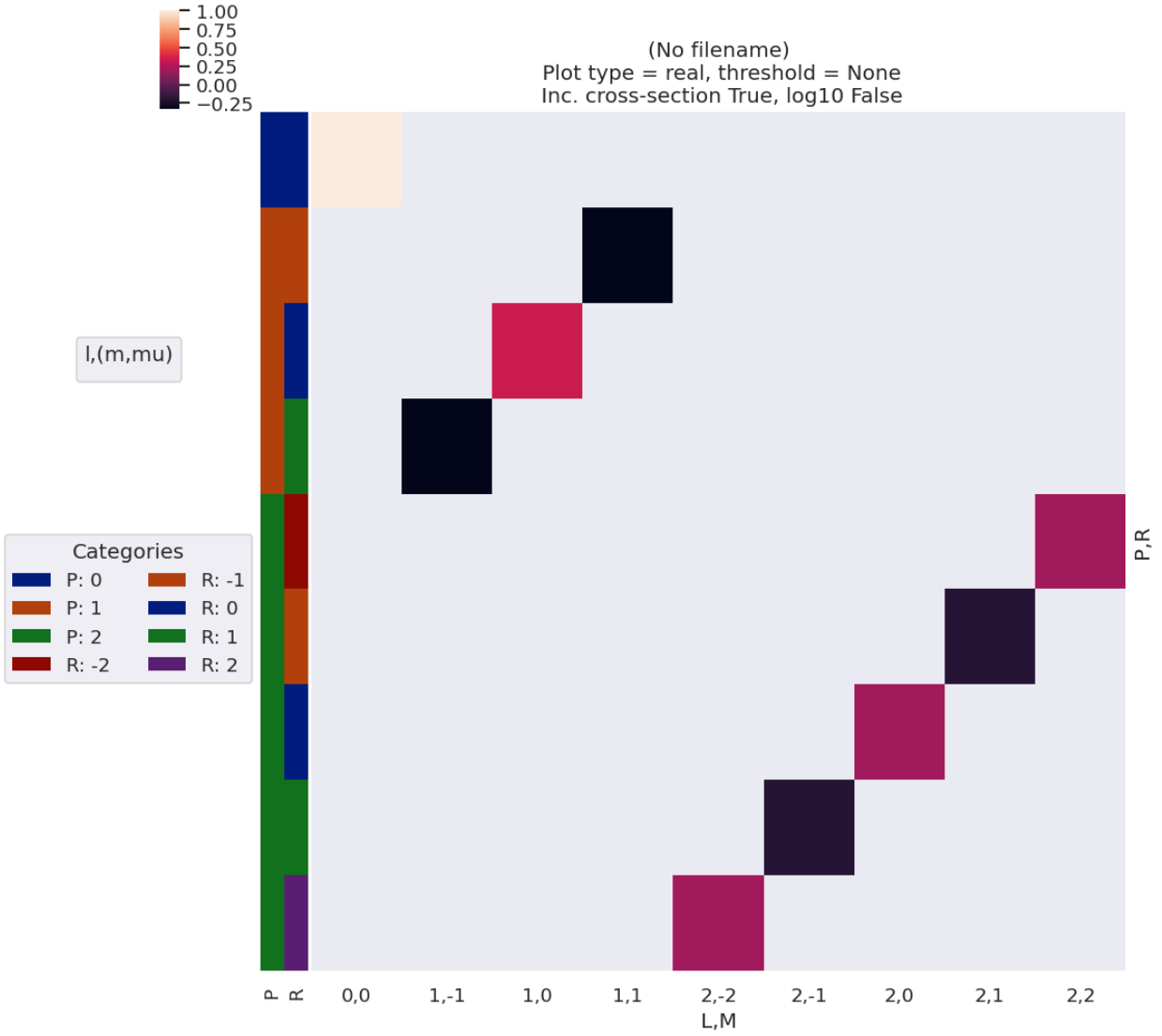


Fig. 4.8: Example $\Delta_{L,M}(0,0,0)$ basis functions (see also Fig. 4.9). For illustration purposes, the plot only shows terms for $R' = 0$. See main text for discussion.

	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. 4.9: Example $\Delta_{L,M}(0,0,0)$ basis functions (see also Fig. 4.8). For illustration purposes, the table only shows terms for $R' = 0$. See main text for discussion.

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 ($Q = S = 0$).

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. (4.27)).

```
#*** Neater version
# Set ADMs for increasing alignment...
tPoints = 10
inputADMs3D = [[0,0,0, *np.ones(tPoints)],     # * np.sqrt(4*np.pi)],  # Optional␣
 ↪multiplier for normalisation
              [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)]]

AKQS = ep.setADMs(ADMs = inputADMs3D)  # TODO WRAP TO CLASS (IF NOT ALREADY!)

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

#*** Plot all
# daPlot, daPlotpd, legendList, gFig = ep.lmPlot(AFterm,     #.sel(R=0).sel(Rp=0),
#                                                xDim = 't', pType = 'r', squeeze =␣
 ↪False, cmap='vlag')  # Note squeeze = False required for 1D case (should add this␣
 ↪to code!)


#*** Plot subsection, L<=2
```

(continues on next page)

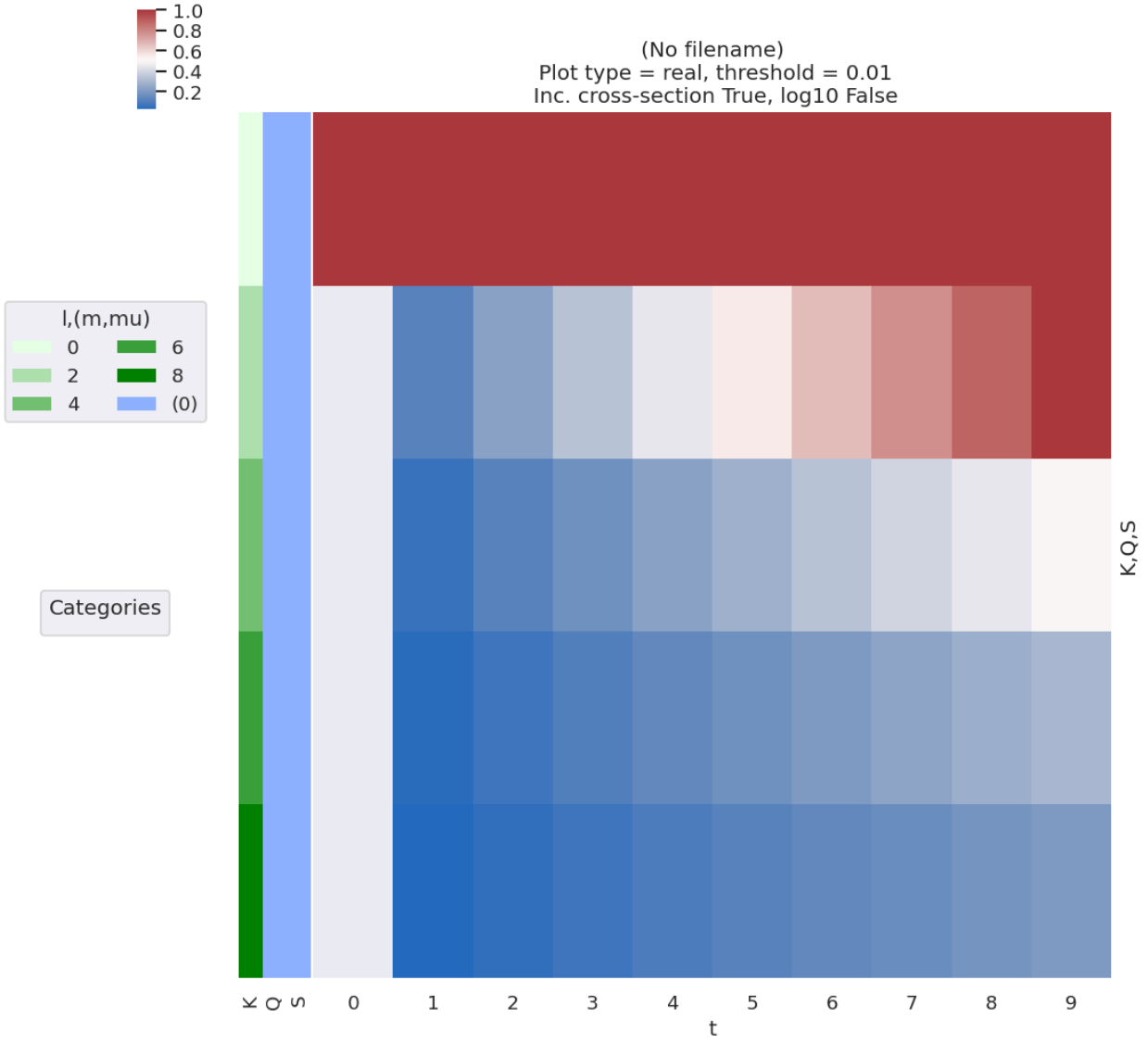


Fig. 4.10: Example ADMs used for *AF* basis function example (see Fig. 4.11). These ADMs essentially show an increasing degree of alignment with the t parameter, with high-order terms increasing at later t.

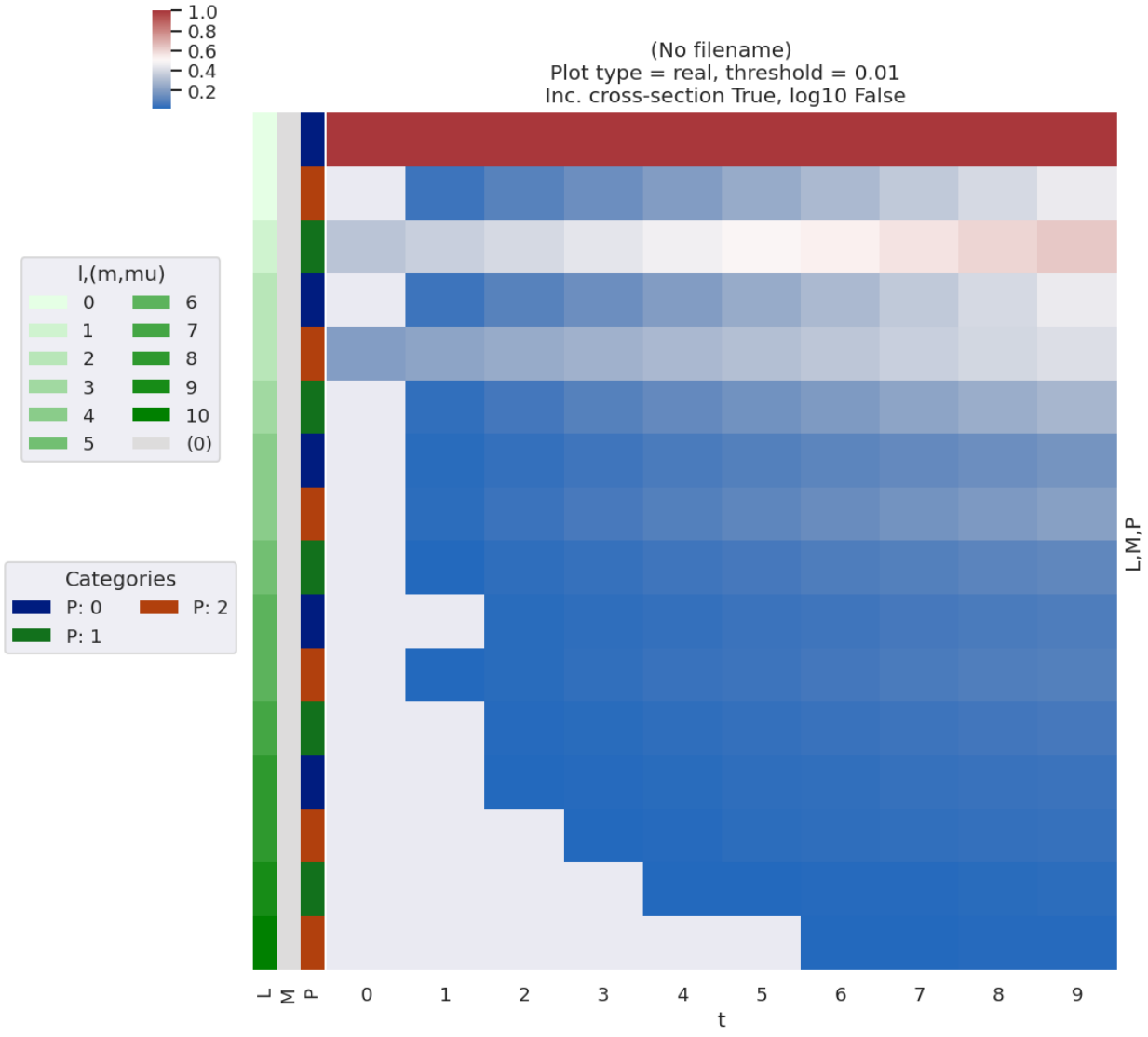


Fig. 4.11: Example of $\tilde{\Delta}_{L,M}(t)$ basis values for various choices of alignment (as per Fig. 4.10). The ADMs essentially show an increasing degree of alignment with the t parameter, with high-order terms increasing at later t, and this is reflected in the $\tilde{\Delta}_{L,M}(t)$ terms with higher-order L appearing at later t.

(continued from previous page)

```
# .sel(R=0).sel(Rp=0) gives M=0 terms only for R=Rp=0
# .sel(R=0) gives M=0 terms only
# .sel(Rp=0) gives M!=0 terms, S-Rp=0 terms only

daPlot, daPlotpd, legendList, gFig = ep.lmPlot(AFterm.where(AFterm.L<=2).sel(Rp=0),
 ↪ # .sel(R=0).sel(Rp=0), #M=0 for R=Rp=0
                                                xDim = 't', pType = 'r', squeeze =
↪False, cmap='vlag')  # Note squeeze = False required for 1D case (should add this
↪to code!)

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

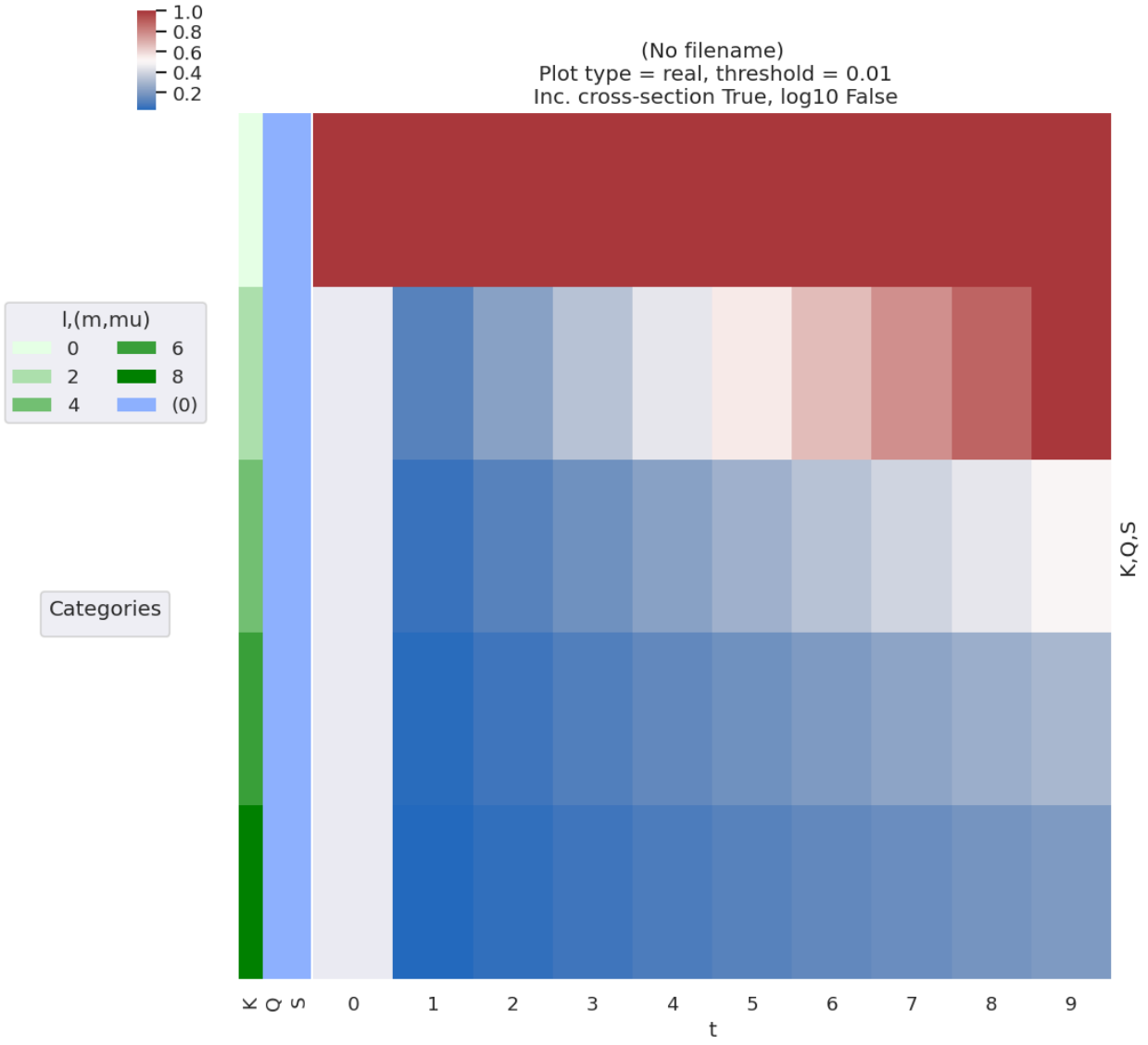


Fig. 4.12: Example $\tilde{\Delta}_{L,M}(t)$ basis values for various choices of "3D" alignment, i.e. including some $K \neq 0$ and $S \neq 0$ terms. Note, in particular, the presence of $M \neq 0$ terms in general, and a complicated dependence of the allowed terms on the alignment (*ADMs*), which may increase, decrease, or even change sign for a given L, M.

For illustration purposes, Fig. 4.12 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 $R' = 0$ terms only.

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. % TODO: give more explict examples here, may want to also reduce and consolidate illustrations further? 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).

4.3.8 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 (4.12)) for a given case. Numerically these are all implemented in the main ePSproc codebase [12, 13, 14], and can be returned by these functions for inspection - the full basis set already defined includes some of these product. Custom tensor product terms are also readily computed with the codebase, with tensor multiplications handled natively by the Xarray data objects.

Polarisation & ADM product term: the main product basis returned, `polProd`, 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. 4.3.7 are illustrated in Fig. 4.13.

The full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ as defined in (4.17) and (4.18) can be determined by the product of this term with the $B_{L,M}$ tensor. This is essentially the complete geometric basis set, % (give or take a phase term or two), 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. 4.3.7 are illustrated in Fig. 4.14.

```
# 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

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

# Convert to Pandas
# pd, _ = ep.multiDimXrToPD(basisPlot, colDims=stackDims)

# Subselect on pol state
daPlot, daPlotpd, legendList, gFig = ep.lmPlot(basisProduct[basisKey].sel(Labels='A'),
 ↪ xDim='t', cmap = cmap, mDimLabel='mu'); # , cmap='vlag');  # Subselect on pol state
# ep.lmPlot(basisProduct[basisKey], xDim='t', cmap = cmap); # , cmap='vlag');  #␣
 ↪Subselect on pol state
```

(continues on next page)

(continued from previous page)

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

```
# Channel functions
daPlot, daPlotpd, legendList, gFig =  ep.lmPlot((basisProduct['BLMtableResort'] *␣
 ↪basisProduct['polProd']).sel(Labels='A').sel({'S-Rp':0}).sel(L=2), xDim='t',␣
 ↪cmap=cmap, mDimLabel='m');  # Basic case
# ep.lmPlot((basis['BLMtableResort'] * basis['polProd']), xDim='t', cmap='vlag');

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

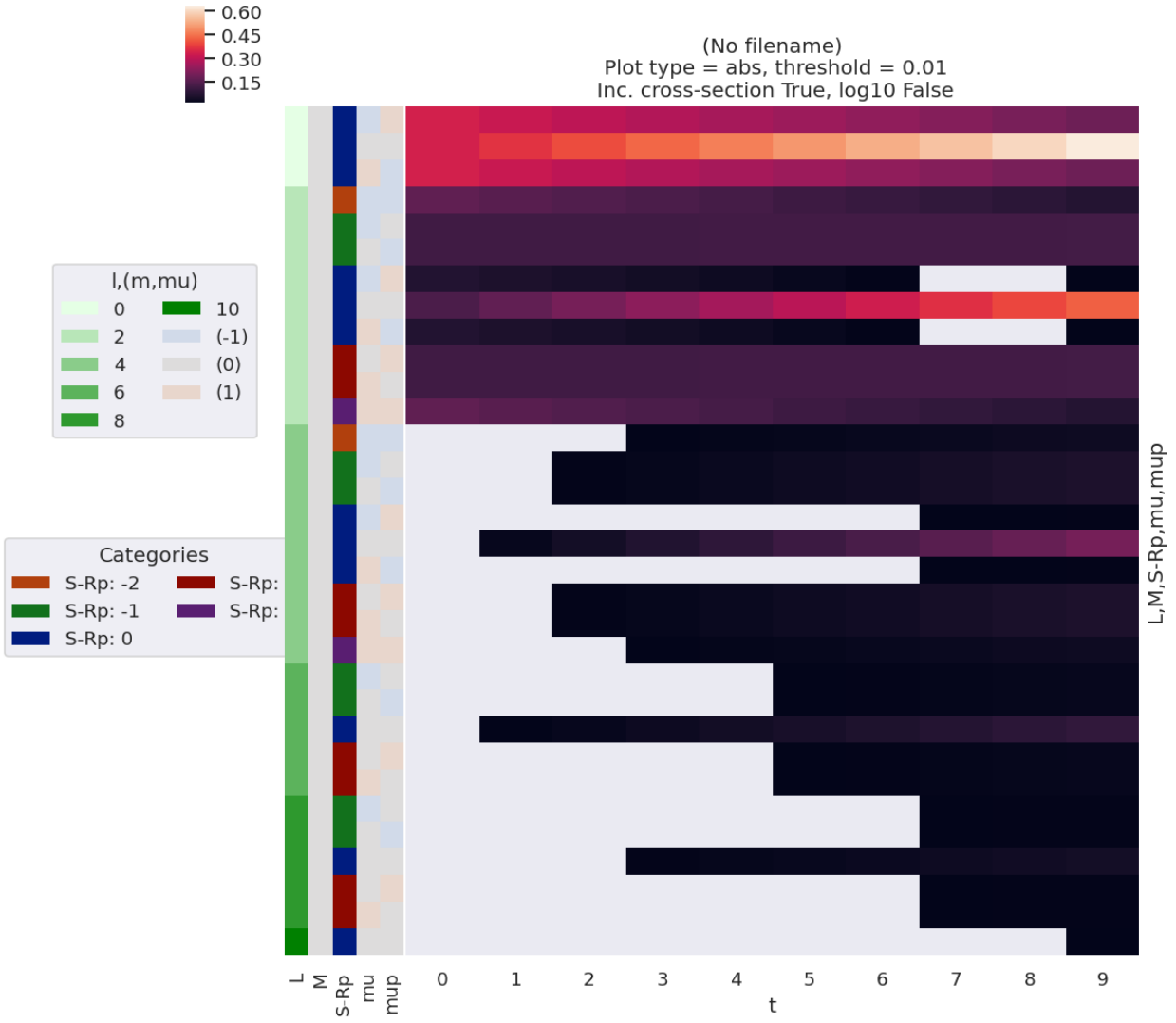


Fig. 4.13: Example product basis function for the polarisation and ADM terms.

```
!date
```

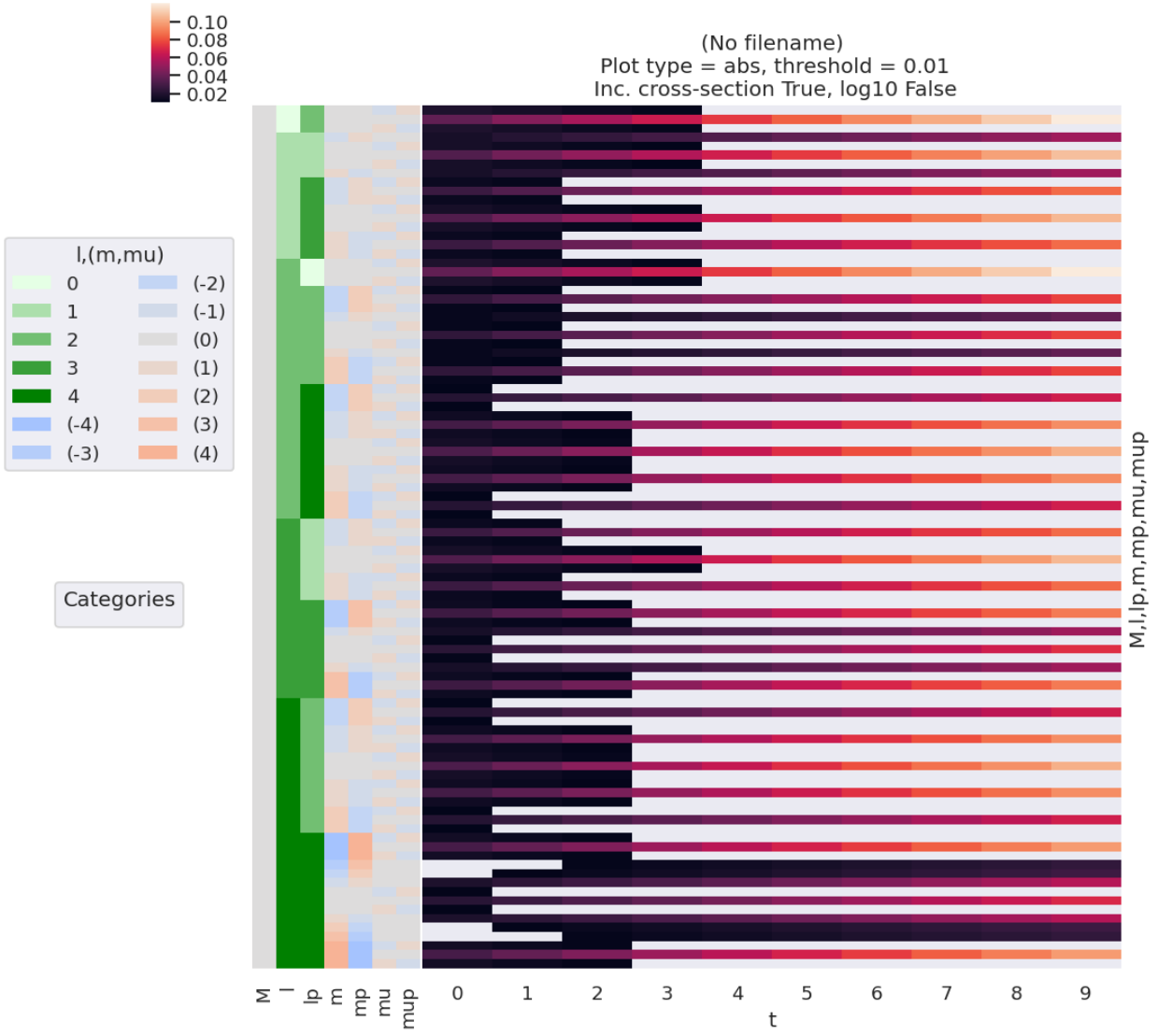


Fig. 4.14: Example of $\bar{\Upsilon}^{u,\zeta\zeta'}_{L,M}$ basis values for various choices of alignment (as per Fig. 4.10), shown for $L = 2$ only. The basis essentially shows the obsevable terms if the ionization matrix elements are neglected, hence the sensitivity of the configuration to each pair 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.

```
Wed 22 Mar 2023 01:30:34 PM EDT
```

4.4 Density matrix representation

4.4.1 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* [52], 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 [52]:

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

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 [52]:

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

$$\hat{\rho} = \sum_n \sum_{mm'} W_n a_{m'}^{(n)} a_m^{(n)*} |\phi_{m'}\rangle\langle\phi_m| \tag{4.28}$$

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

$$\rho_{i,j} = \langle\phi_i|\hat{\rho}|\phi_j\rangle = \sum_n W_n a_i^{(n)} a_j^{(n)*} \tag{4.29}$$

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.

4.4.2 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. (4.3)) 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. 4.3, 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 (u); the density matrix can then be given as:

$$\rho_{L,M}^{u,\zeta\zeta'} = \Upsilon_{L,M}^{u,\zeta\zeta'} \mathbb{I}^{\zeta,\zeta'} \tag{4.30}$$

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. (4.13)):

$$\rho^{\zeta\zeta'} = \mathbb{I}^{\zeta,\zeta'} \tag{4.31}$$

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. (4.30) includes any additional effects incorporated via the channel functions. For reconstruction problems, it is usually the reduced form of Eq. (4.31) 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. (4.30)) 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. (4.12)):

$$\beta_{L,M}^{u} = \sum_{\zeta,\zeta'} \rho_{L,M}^{u,\zeta\zeta'} \tag{4.32}$$

In fact, this form arises naturally since the $\beta_{L,M}^{u}$ terms are the state multipoles (geometric tensors) defining the system, which can be thought of as a coupled basis equivalent of the density matrix representations (see, e.g., Ref. [52], Chpt. 4.).

In a more traditional notation (following Eq. (4.3), see also Ref. [54]), the density operator can be expressed as:

$$\rho(t) = \sum_{LM} \sum_{KQS} A_{QS}^{K}(t) \sum_{\zeta\zeta'} \Upsilon_{L,M}^{u,\zeta\zeta'} |\zeta, \Psi_{+}\rangle\langle\zeta, \Psi_{+}|\mu_q \rho_i \mu_{q'}^{*}|\zeta', \Psi_{+}\rangle\langle\zeta', \Psi_{+}| \tag{4.33}$$

This is, effectively, equivalent to an expansion in the various tensor operators defined in the channel function notation above (Eq. (4.30)), 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 $A_{Q,S}^{K}(t)$ term. (For further discussion of the use of density matrices in other specific cases, see *Quantum Metrology* Vol. 1 [1], 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 [14], as well as in the literature, see, e.g., Ref. [52] for general discussion, Ref. [45] 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 [55]. 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. [52], 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. [56, 57] for recent discussions in this context).

4.4.3 Numerical setup

This follows the setup in Sect. 4.3 *Tensor formulation of photoionization*, using a symmetry-based set of basis functions for demonstration purposes. (Repeated code is hidden in PDF version.)

4.4.4 Compute a density matrix

A basic density matrix computation routine is implemented in the ePSproc codebase [12, 13, 14]. 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 Eqns. (4.28), (4.29), 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 [14].

```
# 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.
denDims = 'LM'  # Set dimensions for density matrix. Note stacked dims are OK, in␣
 ↪this case LM = {l,m}
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 https://epsproc.readthedocs.io/en/latest/modules/epsproc.calc.density.html
 ↪#epsproc.calc.density.densityCalc
```

(continues on next page)

(continued from previous page)

```
daOut, *_ = density.densityCalc(matE, denDims = denDims, selDims = selDims, thres =␣
↪thres)

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

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

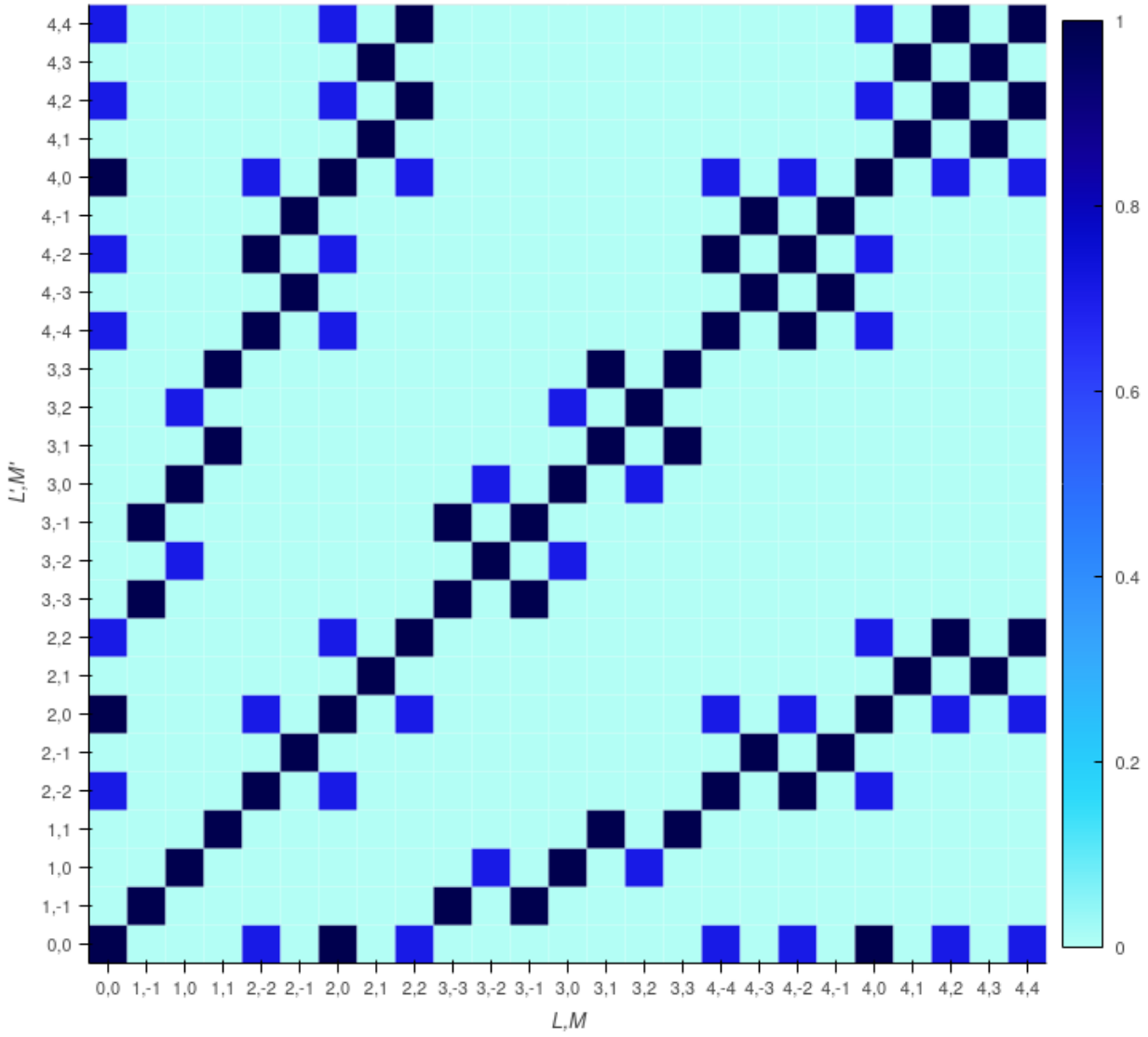


Fig. 4.15: Example density matrix, computed from matrix elements defined purely by symmetry. Note in this case only the real part is non-zero. Axes labels give terms $\{L, M\}$ and $\{L', M'\}$.

4.4.5 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. 4.16; 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. 4.16(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))
matE_noise = matE + matE*((np.random.rand(*list(matE.shape)) - 0.5) * 2*noise)  # Set␣
 ↪range to random values +/-1 * 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)

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'"))
```

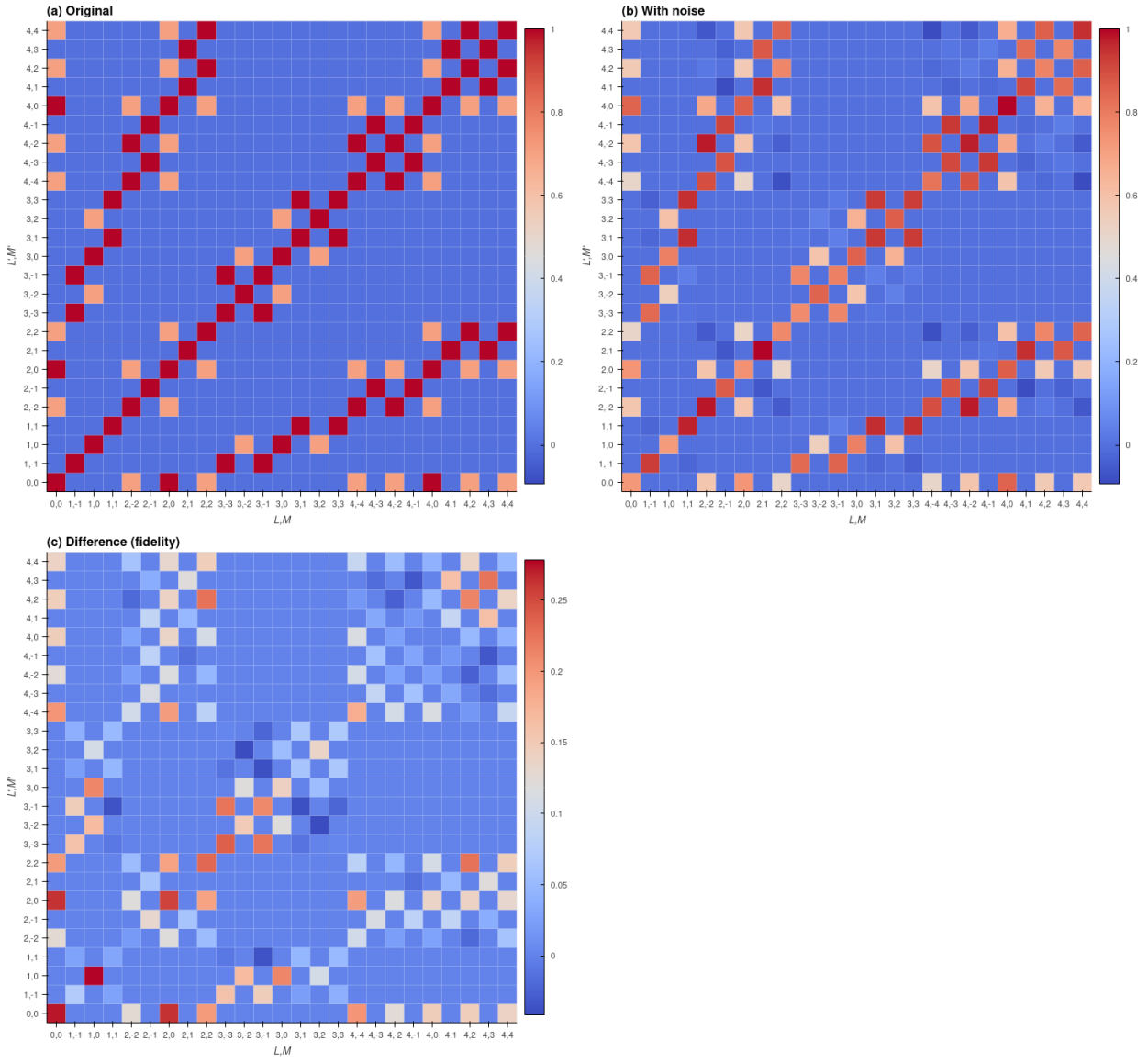


Fig. 4.16: Example density matrices, computed from matrix elements defined purely by symmetry. Here the panels show (a) the original density matrix, (b) density matrix computed with +/- 10% random noise added to the original matrix elements, (c) the difference matrix, which indicates the fidelity of the noisy case relative to the original case. For normalised density matrices the 10% noise case translates to a standard deviation $\sigma \approx 0.2$ on the differences; the maximum error in the test case as illustrated =0.278.

4.4.6 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 [] 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 [] for more possibilities.

Convert numerical arrays to QuTiP objects

```
# Import QuTip
from qutip import *

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

# QuTip objects have data as Numpy arrays, and render as typeset matrices in a␣
 ↪notebook
pa
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
/tmp/ipykernel_19939/1965604571.py in <module>
      1 # Import QuTip
----> 2 from qutip import *
      3
      4 # Wrap density matrices to QuTip objects
      5 # Note sum('Sym') to ensure 2D matrix, and .data to pass Numpy data array␣
 ↪only

ModuleNotFoundError: No module named 'qutip'
```

Fidelity metric

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

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

This is implemented by the `fidelity` function in QuTip. Of note in this case is the 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. 4.16(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()
```

4.5 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. 4.3, 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. 4.3.8, including the full channel (response) functions $\Upsilon_{L,M}^{u,\zeta\zeta'}$ ((4.17) and (4.18)), 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. [58, 59]).

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.

4.5.1 Numerical setup

This follows the setup in Sect. 4.3 *Tensor formulation of photoionization*, using a symmetry-based set of basis functions for demonstration purposes. (Repeated code is hidden in PDF version.)

4.5.2 Experimental information content

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

$$M = \mathrm{n}\{\beta_{L,M}\} \tag{4.34}$$

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:

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

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

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

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. (4.12)), 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 trivial to investigate some of these properties, as shown in the code blocks below.

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

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

# Clean array to remove terms |m|>l, and display
BetaNorm.BLM.where(np.abs(BetaNorm.BLM.m)<=BetaNorm.BLM.l,drop=True)
```

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

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

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

```
# The index can be returned as a Pandas object, and statistical routines applied...

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"Thresholded array (thres={thres}), M={ep.matEleSelector(BetaNorm,␣
 ↪thres=thres).BLM.indexes['BLM'].nunique()}")
```

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

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

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

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

t	0	1	2	3	4	5	6	7	8	9
count	2.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000	5.000
mean	0.389	0.190	0.225	0.259	0.294	0.328	0.363	0.397	0.432	0.466
std	0.115	0.242	0.265	0.290	0.315	0.341	0.368	0.395	0.423	0.451
min	0.308	0.001	0.002	0.003	0.004	0.006	0.007	0.008	0.009	0.010
25%	0.348	0.009	0.017	0.026	0.035	0.043	0.052	0.060	0.069	0.078
50%	0.389	0.041	0.083	0.124	0.166	0.207	0.249	0.290	0.332	0.373
75%	0.430	0.382	0.457	0.532	0.607	0.681	0.751	0.797	0.844	0.891
max	0.470	0.517	0.564	0.610	0.657	0.704	0.756	0.831	0.906	0.980

```
# 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()
```

l	0	2	4	6	8
m	0	0	0	0	0
count	10.000	10.000	9.000	9.000	9.000
mean	0.680	0.644	0.207	0.043	0.006
std	0.141	0.226	0.114	0.024	0.003
min	0.470	0.308	0.041	0.009	0.001
25%	0.575	0.476	0.124	0.026	0.003
50%	0.680	0.644	0.207	0.043	0.006
75%	0.786	0.812	0.290	0.060	0.008
max	0.891	0.980	0.373	0.078	0.010

BELOW VERY ROUGH - check old dev notebook for better…

```
# Examine unique values at a given level of difference/rounding
# pd.cut(BetaNormLinearADMsPD, 10)

# Use cut - not very useful here, maybe better with agg?
testCut = BetaNormLinearADMsPD.apply(pd.cut, args=[10])
# testCut
# testCut.agg('count')
testCut.agg('unique')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/opt/conda/lib/python3.9/site-packages/pandas/core/apply.py in agg(self)
    699         try:
--> 700             result = super().agg()
    701         except TypeError as err:

/opt/conda/lib/python3.9/site-packages/pandas/core/apply.py in agg(self)
    157         if isinstance(arg, str):
--> 158             return self.apply_str()
    159

/opt/conda/lib/python3.9/site-packages/pandas/core/apply.py in apply_str(self)
    866             return obj._constructor_sliced(value, index=self.agg_axis)
--> 867         return super().apply_str()
    868

/opt/conda/lib/python3.9/site-packages/pandas/core/apply.py in apply_str(self)
    506                 raise ValueError(f"Operation {f} does not support axis=1")
--> 507         return self._try_aggregate_string_function(obj, f, *self.args,␣
↪**self.kwargs)
    508

/opt/conda/lib/python3.9/site-packages/pandas/core/apply.py in _try_aggregate_
↪string_function(self, obj, arg, *args, **kwargs)
```

(continues on next page)

(continued from previous page)

```
    587             # in particular exclude Window
--> 588             return f(obj, *args, **kwargs)
    589

<__array_function__ internals> in unique(*args, **kwargs)

/opt/conda/lib/python3.9/site-packages/numpy/lib/arraysetops.py in unique(ar,␣
↪return_index, return_inverse, return_counts, axis)
    261     if axis is None:
--> 262         ret = _unique1d(ar, return_index, return_inverse, return_counts)
    263         return _unpack_tuple(ret)

/opt/conda/lib/python3.9/site-packages/numpy/lib/arraysetops.py in _unique1d(ar,␣
↪return_index, return_inverse, return_counts)
    322     else:
--> 323         ar.sort()
    324         aux = ar

TypeError: '<' not supported between instances of 'pandas._libs.interval.Interval'␣
↪and 'float'

The above exception was the direct cause of the following exception:

TypeError                                 Traceback (most recent call last)
/tmp/ipykernel_20519/666826212.py in <module>
      6 # testCut
      7 # testCut.agg('count')
----> 8 testCut.agg('unique')

/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py in aggregate(self,␣
↪func, axis, *args, **kwargs)
   8548
   8549         op = frame_apply(self, func=func, axis=axis, args=args,␣
↪kwargs=kwargs)
-> 8550         result = op.agg()
   8551
   8552         if relabeling:

/opt/conda/lib/python3.9/site-packages/pandas/core/apply.py in agg(self)
    704                 f"incompatible data and dtype: {err}"
    705             )
--> 706             raise exc from err
    707         finally:
    708             self.obj = obj

TypeError: DataFrame constructor called with incompatible data and dtype: '<' not␣
↪supported between instances of 'pandas._libs.interval.Interval' and 'float'
```

```
BetaNormLinearADMsPD
```

```
testBasisPD.round(1).T.agg(['min','max','var','count','nunique'])
```

```
testBasisPD.T.nunique()
```

```
# BetaNormX.BLM.indexes['BLM']  #.unqiue()
# pd.unique(BetaNormX.BLM.indexes['BLM'])
# dir(BetaNormX.BLM.indexes['BLM'])   #.agg('count')

BetaNorm = BetaNormX

#
BetaNorm.BLM

# A basic
BetaNorm.BLM.indexes['BLM'].nunique()
```

```
BetaNormX
```

4.5.3 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. [60, 61, 62] for details.)

4.5.4 Information content from density matrices

CHAPTER

FIVE

NUMERICAL IMPLEMENTATION

Further details of the numerical implementations - base packages, conventions and refs - to go here or in appendix?

Part III

Backmatter

CHAPTER

SIX

BIBLIOGRAPHY

CHAPTER

SEVEN

GLOSSARY

MF
: Molecular frame (MF) - coordinate system referenced to the molecule, usually with the z-axis corresponding to the highest symmetry axis.

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.

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. 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.

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. (4.1). 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, 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.

Axis distribution moments
: See *ADMs*.

Part IV

Test pages

CHAPTER

EIGHT

BUILD VERSIONS AND CONFIG TESTS

8.1 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 #271: omp_set_nested routine deprecated, please use omp_set_max_active_
 ↪levels instead.
```

```
--------------------------------------------------------------------------------
  Date: Wed Mar 22 13:31:05 2023 EDT

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

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

             pemtk : 0.0.1
           epsproc : 1.3.2-dev
            xarray : 2022.3.0
            pandas : 1.3.4
             scipy : 1.7.1
        matplotlib : 3.4.3
        jupyterlab : 3.2.1
            plotly : 5.13.1
         holoviews : 1.15.4
             numpy : 1.20.3
```

(continues on next page)

(continued from previous page)

```
            IPython : 7.28.0
             scooby : 0.7.1
--------------------------------------------------------------------------------
```

```
!jupyter-book --version
```

```
Jupyter Book      : 0.15.1
External ToC      : 0.3.1
MyST-Parser       : 0.18.1
MyST-NB           : 0.17.1
Sphinx Book Theme : 1.0.0
Jupyter-Cache     : 0.5.0
NbClient          : 0.5.4
```

8.2 Docker build env

To do

8.3 Book versions

```
QMpath = '/home/jovyan/QM3'
!git -C {QMpath} branch
!git -C {QMpath} log --format="%H" -n 1
```

```
  gh-pages
* main
dfa0ef69d8bd5a16ee2fef94983d97b398467a8f
```

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

```
4263720ddb7693f9a31bfb2d2d9f93f38d78c940        refs/heads/gh-pages
c1abd028f0bc20c5871bae7dc880467b14a33b56        refs/heads/main
```

8.4 Github pkg versions

Note - can't get versions for local pip installs from repo(?).

```
from pathlib import Path
import epsproc as ep
ep.__file__
```

```
'/opt/conda/lib/python3.9/site-packages/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
```

```
fatal: not a git repository (or any of the parent directories): .git
fatal: not a git repository (or any of the parent directories): .git
```

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

```
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
92819e6072d949f4ea3c71020a48046c117ce6c2        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
```

```
7b3bcb92ce5262cfc46562366c2ce024dd4509d5        refs/heads/master
3f4686dffdbb310f15692f978ba36d6a3d15e8d3        refs/heads/mfFittingDev
```

8.5 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                       1_llvm    conda-forge
accessible-pygments       0.0.3                     pypi_0    pypi
alabaster                 0.7.13                    pypi_0    pypi
alembic                   1.7.4               pyhd8ed1ab_0    conda-forge
alsa-lib                  1.2.3.2               h166bdaf_0    conda-forge
altair                    4.1.0                       py_1    conda-forge
ansi2html                 1.8.0             py39hf3d152e_1    conda-forge
anyio                     3.3.4             py39hf3d152e_0    conda-forge
appdirs                   1.4.4               pyh9f0ad1d_0    conda-forge
argon2-cffi               21.1.0            py39h3811e60_0    conda-forge
asteval                   0.9.29              pyhd8ed1ab_0    conda-forge
astropy                   5.0.4             py39hd257fcd_0    conda-forge
async_generator           1.10                        py_0    conda-forge
atk-1.0                   2.36.0                h3371d22_4    conda-forge
attrs                     21.2.0              pyhd8ed1ab_0    conda-forge
babel                     2.9.1               pyh44b312d_0    conda-forge
backcall                  0.2.0               pyh9f0ad1d_0    conda-forge
backports                 1.0                         py_2    conda-forge
backports.functools_lru_cache 1.6.4              pyhd8ed1ab_0    conda-forge
beautifulsoup4            4.10.0              pyha770c72_0    conda-forge
blas                      2.112                    openblas    conda-forge
blas-devel                3.9.0           12_linux64_openblas    conda-forge
bleach                    4.1.0               pyhd8ed1ab_0    conda-forge
blinker                   1.4                         py_1    conda-forge
blosc                     1.21.0                h9c3ff4c_0    conda-forge
bokeh                     2.4.1             py39hf3d152e_1    conda-forge
boost-cpp                 1.74.0                h312852a_4    conda-forge
bottleneck                1.3.2             py39hce5d2b2_4    conda-forge
brotli                    1.0.9                 h7f98852_5    conda-forge
brotli-bin                1.0.9                 h7f98852_5    conda-forge
brotli-python             1.0.9             py39h5a03fae_7    conda-forge
brotlipy                  0.7.0           py39h3811e60_1001    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                  2.0.4                 h5f21a17_1    conda-forge
ca-certificates           2022.12.7             ha878542_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              h6cf1ce9_1008    conda-forge
cartopy                   0.20.1            py39he7aa91e_2    conda-forge
certifi                   2022.12.7           pyhd8ed1ab_0    conda-forge
certipy                   0.1.3                       py_0    conda-forge
cffi                      1.14.6            py39h4bc2ebd_1    conda-forge
cfitsio                   4.0.0                 h9a35b8e_0    conda-forge
cftime                    1.6.0             py39hd257fcd_1    conda-forge
chardet                   4.0.0             py39hf3d152e_1    conda-forge
charls                    2.2.0                 h9c3ff4c_0    conda-forge
```

(continues on next page)

(continued from previous page)

```
charset-normalizer        2.0.0              pyhd8ed1ab_0    conda-forge
chrpath                   0.16              h7f98852_1002    conda-forge
click                     8.0.3            py39hf3d152e_0    conda-forge
cloudpickle               2.0.0              pyhd8ed1ab_0    conda-forge
colorama                  0.4.4              pyh9f0ad1d_0    conda-forge
colorcet                  3.0.1              pyhd8ed1ab_0    conda-forge
conda                     4.10.3           py39hf3d152e_2    conda-forge
conda-package-handling    1.7.3            py39h3811e60_0    conda-forge
configurable-http-proxy   4.5.0           node15_he6ea98c_0    conda-forge
cryptography              35.0.0           py39h95dcef6_1    conda-forge
curl                      7.79.1               h2574ce0_1    conda-forge
cycler                    0.10.0                     py_2    conda-forge
cython                    0.29.24          py39he80948d_0    conda-forge
cytoolz                   0.11.0           py39h3811e60_3    conda-forge
dash                      2.9.1              pyhd8ed1ab_0    conda-forge
dask                      2021.10.0          pyhd8ed1ab_0    conda-forge
dask-core                 2021.10.0          pyhd8ed1ab_0    conda-forge
dbus                      1.13.6               h5008d03_3    conda-forge
dcw-gmt                   2.1.1                ha770c72_0    conda-forge
debugpy                   1.4.1            py39he80948d_0    conda-forge
decorator                 5.1.0              pyhd8ed1ab_0    conda-forge
defusedxml                0.7.1              pyhd8ed1ab_0    conda-forge
dill                      0.3.4              pyhd8ed1ab_0    conda-forge
distributed               2021.10.0        py39hf3d152e_0    conda-forge
docutils                  0.18.1                   pypi_0    pypi
ducc0                     0.23.0           py39h5a03fae_0    conda-forge
entrypoints               0.3             py39hde42818_1002    conda-forge
epsproc                   1.3.2.dev0               pypi_0    pypi
exceptiongroup            1.1.1              pyhd8ed1ab_0    conda-forge
expat                     2.4.8                h27087fc_0    conda-forge
ffmpeg                    4.4.0                h6987444_4    conda-forge
fftw                      3.3.10          nompi_h77c792f_102    conda-forge
firefox                   100.0                h27087fc_0    conda-forge
flask                     2.1.3              pyhd8ed1ab_0    conda-forge
flask-compress            1.13               pyhd8ed1ab_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                2.14.0               h8e229c2_0    conda-forge
fonts-conda-ecosystem     1                             0    conda-forge
fonts-conda-forge         1                             0    conda-forge
freetype                  2.10.4               h0708190_1    conda-forge
freexl                    1.0.6                h7f98852_0    conda-forge
fribidi                   1.0.10               h36c2ea0_0    conda-forge
fsspec                    2021.10.1          pyhd8ed1ab_0    conda-forge
future                    0.18.3             pyhd8ed1ab_0    conda-forge
gdal                      3.3.3            py39h0494519_2    conda-forge
gdk-pixbuf                2.42.6               h04a7f16_0    conda-forge
geckodriver               0.30.0               h3146498_0    conda-forge
geos                      3.10.0               h9c3ff4c_0    conda-forge
geotiff                   1.7.0                hcfb7246_3    conda-forge
gettext                   0.19.8.1            h73d1719_1008    conda-forge
ghostscript               9.54.0               h27087fc_2    conda-forge
ghp-import                2.1.0                    pypi_0    pypi
giflib                    5.2.1                h36c2ea0_2    conda-forge
```

(continues on next page)

(continued from previous page)

```
glib                        2.70.2              h780b84a_4    conda-forge
glib-tools                  2.70.2              h780b84a_4    conda-forge
gmp                         6.2.1               h58526e2_0    conda-forge
gmpy2                       2.1.0b5          py39h78fa15d_0   conda-forge
gmt                         6.2.0               h7e416ca_3    conda-forge
gnuplot                     5.4.3               hedd7fda_0    conda-forge
gnutls                      3.6.13              h85f3911_1    conda-forge
graphicsmagick              1.3.37              hdc87540_0    conda-forge
graphite2                   1.3.13            h58526e2_1001   conda-forge
greenlet                    1.1.2            py39he80948d_0   conda-forge
gshhg-gmt                   2.3.7             ha770c72_1003   conda-forge
gst-plugins-base            1.18.5              hf529b03_3    conda-forge
gstreamer                   1.18.5              h9f60fe5_3    conda-forge
gtk2                        2.24.33             h539f30e_1    conda-forge
h11                         0.14.0             pyhd8ed1ab_0   conda-forge
h5netcdf                    1.1.0              pyhd8ed1ab_1   conda-forge
h5py                        3.4.0          nompi_py39h7e08c79_101    conda-forge
harfbuzz                    3.1.1               h83ec7ef_0    conda-forge
hdf4                        4.2.15              h10796ff_3    conda-forge
hdf5                        1.12.1         nompi_h2750804_101    conda-forge
heapdict                    1.0.1                     py_0    conda-forge
holoviews                   1.15.4             pyhd8ed1ab_0   conda-forge
hvplot                      0.8.3                     py_0    pyviz
icu                         68.2                h9c3ff4c_0    conda-forge
idna                        3.1                pyhd3deb0d_0   conda-forge
imagecodecs                 2021.8.26        py39h571908b_2   conda-forge
imageio                     2.9.0                     py_0    conda-forge
imagesize                   1.4.1                   pypi_0    pypi
importlib-metadata          4.8.1            py39hf3d152e_0   conda-forge
importlib_metadata          4.8.1                hd8ed1ab_1   conda-forge
importlib_resources         5.3.0              pyhd8ed1ab_0   conda-forge
ipykernel                   6.4.2            py39hef51801_0   conda-forge
ipympl                      0.8.2              pyhd8ed1ab_0   conda-forge
ipython                     7.28.0           py39hef51801_0   conda-forge
ipython_genutils            0.2.0                     py_1    conda-forge
ipywidgets                  7.6.5              pyhd8ed1ab_0   conda-forge
itsdangerous                2.1.2              pyhd8ed1ab_0   conda-forge
jbig                        2.1               h7f98852_2003   conda-forge
jedi                        0.18.0           py39hf3d152e_2   conda-forge
jinja2                      3.0.2              pyhd8ed1ab_0   conda-forge
joblib                      1.1.0              pyhd8ed1ab_0   conda-forge
jpeg                        9d                   h36c2ea0_0   conda-forge
json-c                      0.15                 h98cffda_0   conda-forge
json5                       0.9.5              pyh9f0ad1d_0   conda-forge
jsonschema                  4.1.2              pyhd8ed1ab_0   conda-forge
jupyter-book                0.15.1                  pypi_0    pypi
jupyter-cache               0.5.0                   pypi_0    pypi
jupyter-dash                0.4.2              pyhd8ed1ab_1   conda-forge
jupyter_client              7.0.6              pyhd8ed1ab_0   conda-forge
jupyter_core                4.9.1            py39hf3d152e_0   conda-forge
jupyter_server              1.11.1             pyhd8ed1ab_0   conda-forge
jupyter_telemetry           0.1.0              pyhd8ed1ab_1   conda-forge
jupyterhub                  1.4.2            py39hf3d152e_0   conda-forge
jupyterhub-base             1.4.2            py39hf3d152e_0   conda-forge
jupyterlab                  3.2.1              pyhd8ed1ab_0   conda-forge
jupyterlab-spellchecker     0.7.3                   pypi_0    pypi
```

(continues on next page)

(continued from previous page)

```
jupyterlab_pygments       0.1.2              pyh9f0ad1d_0    conda-forge
jupyterlab_server         2.8.2              pyhd8ed1ab_0    conda-forge
jupyterlab_widgets        1.0.2              pyhd8ed1ab_0    conda-forge
jupytext                  1.14.5             pyhcff175f_0    conda-forge
jxrlib                    1.1                  h7f98852_2    conda-forge
kaleido-core              0.2.1                h3644ca4_0    conda-forge
kealib                    1.4.14               h87e4c3c_3    conda-forge
kiwisolver                1.3.2             py39h1a9c180_0    conda-forge
krb5                      1.19.2               hcc1bbae_2    conda-forge
lame                      3.100              h7f98852_1001    conda-forge
latexcodec                2.0.1                    pypi_0    pypi
lcms2                     2.12                 hddcbb42_0    conda-forge
ld_impl_linux-64          2.36.1               hea4e1c9_2    conda-forge
lerc                      3.0                  h9c3ff4c_0    conda-forge
libaec                    1.0.6                h9c3ff4c_0    conda-forge
libarchive                3.5.2                hccf745f_1    conda-forge
libblas                   3.9.0           12_linux64_openblas    conda-forge
libbrotlicommon           1.0.9                h7f98852_5    conda-forge
libbrotlidec              1.0.9                h7f98852_5    conda-forge
libbrotlienc              1.0.9                h7f98852_5    conda-forge
libcblas                  3.9.0           12_linux64_openblas    conda-forge
libclang                  11.1.0          default_ha53f305_1    conda-forge
libcurl                   7.79.1               h2574ce0_1    conda-forge
libdap4                   3.20.6               hd7c4107_2    conda-forge
libdeflate                1.8                  h7f98852_0    conda-forge
libedit                   3.1.20191231         he28a2e2_2    conda-forge
libev                     4.33                 h516909a_1    conda-forge
libevent                  2.1.10               h9b69904_4    conda-forge
libffi                    3.4.2                h9c3ff4c_4    conda-forge
libgcc-ng                 11.2.0              h1d223b6_11    conda-forge
libgd                     2.3.3                h6ad9fb6_0    conda-forge
libgdal                   3.3.3                h18e3bf0_2    conda-forge
libgfortran-ng            11.2.0              h69a702a_11    conda-forge
libgfortran5              11.2.0              h5c6108e_11    conda-forge
libglib                   2.70.2               h174f98d_4    conda-forge
libgomp                   11.2.0              h1d223b6_11    conda-forge
libiconv                  1.16                 h516909a_0    conda-forge
libkml                    1.3.0              h238a007_1014    conda-forge
liblapack                 3.9.0           12_linux64_openblas    conda-forge
liblapacke                3.9.0           12_linux64_openblas    conda-forge
libllvm11                 11.1.0               hf817b99_2    conda-forge
libmsym                   0.2.4                    pypi_0    pypi
libnetcdf                 4.8.1           nompi_hb3fd0d9_101    conda-forge
libnghttp2                1.43.0               h812cca2_1    conda-forge
libnsl                    2.0.0                h7f98852_0    conda-forge
libogg                    1.3.4                h7f98852_1    conda-forge
libopenblas               0.3.18          pthreads_h8fe5266_0    conda-forge
libopus                   1.3.1                h7f98852_1    conda-forge
libpng                    1.6.37               h21135ba_2    conda-forge
libpq                     13.5                 hd57d9b9_1    conda-forge
libprotobuf               3.18.1               h780b84a_0    conda-forge
librttopo                 1.1.0                h0ad649c_7    conda-forge
libsodium                 1.0.18               h36c2ea0_1    conda-forge
libsolv                   0.7.19               h780b84a_5    conda-forge
libspatialite             5.0.1               h1d9e4f1_10    conda-forge
libssh2                   1.10.0               ha56f1ee_2    conda-forge
```

(continues on next page)

(continued from previous page)

```
libstdcxx-ng              11.2.0               he4da1e4_11    conda-forge
libtiff                   4.3.0                 h6f004c6_2    conda-forge
libuuid                   2.32.1              h7f98852_1000    conda-forge
libuv                     1.41.1                h7f98852_0    conda-forge
libvorbis                 1.3.7                 h9c3ff4c_0    conda-forge
libvpx                    1.11.0                h9c3ff4c_3    conda-forge
libwebp                   1.2.1                 h3452ae3_0    conda-forge
libwebp-base              1.2.1                 h7f98852_0    conda-forge
libxcb                    1.13                h7f98852_1004    conda-forge
libxkbcommon              1.0.3                 he3ba5ed_0    conda-forge
libxml2                   2.9.12                h72842e0_0    conda-forge
libzip                    1.8.0                 h4de3113_1    conda-forge
libzlib                   1.2.11              h36c2ea0_1013    conda-forge
libzopfli                 1.0.3                 h9c3ff4c_0    conda-forge
linkify-it-py             2.0.0                     pypi_0    pypi
llvm-openmp               12.0.1                h4bd325d_1    conda-forge
llvmlite                  0.37.0             py39h1bbdace_0    conda-forge
lmfit                     1.1.0                pyhd8ed1ab_0    conda-forge
locket                    0.2.0                        py_2    conda-forge
lz4-c                     1.9.3                 h9c3ff4c_1    conda-forge
lzo                       2.10                h516909a_1000    conda-forge
mako                      1.1.5                pyhd8ed1ab_0    conda-forge
mamba                     0.17.0             py39h951de11_0    conda-forge
markdown                  3.4.1                pyhd8ed1ab_0    conda-forge
markdown-it-py            2.2.0                pyhd8ed1ab_0    conda-forge
markupsafe                2.0.1              py39h3811e60_0    conda-forge
mathjax                   2.7.7                 ha770c72_3    conda-forge
matplotlib-base           3.4.3              py39h2fa2bec_1    conda-forge
matplotlib-inline         0.1.3                pyhd8ed1ab_0    conda-forge
mdit-py-plugins           0.3.5                pyhd8ed1ab_0    conda-forge
mdurl                     0.1.0                pyhd8ed1ab_0    conda-forge
mistune                   0.8.4             py39h3811e60_1004    conda-forge
mock                      4.0.3              py39hf3d152e_1    conda-forge
mpc                       1.2.1                 h9f54685_0    conda-forge
mpfr                      4.1.0                 h9202a9a_1    conda-forge
mpmath                    1.2.1                pyhd8ed1ab_0    conda-forge
msgpack-python            1.0.2              py39h1a9c180_1    conda-forge
mysql-common              8.0.27                ha770c72_3    conda-forge
mysql-libs                8.0.27                hfa10184_3    conda-forge
myst-nb                   0.17.1                    pypi_0    pypi
myst-parser               0.18.1                    pypi_0    pypi
nbclassic                 0.3.3                pyhd8ed1ab_0    conda-forge
nbclient                  0.5.4                pyhd8ed1ab_0    conda-forge
nbconvert                 6.2.0              py39hf3d152e_0    conda-forge
nbformat                  5.1.3                pyhd8ed1ab_0    conda-forge
ncurses                   6.2                   h58526e2_4    conda-forge
nest-asyncio              1.5.1                pyhd8ed1ab_0    conda-forge
netcdf4                   1.5.8           nompi_py39h64b754b_101    conda-forge
nettle                    3.6                   he412f7d_0    conda-forge
networkx                  2.6.3                pyhd8ed1ab_1    conda-forge
nodejs                    15.14.0               h92b4a50_0    conda-forge
notebook                  6.4.5                pyha770c72_0    conda-forge
nspr                      4.32                  h9c3ff4c_1    conda-forge
nss                       3.72                  hb5efdd6_0    conda-forge
numba                     0.54.1             py39h56b8d98_0    conda-forge
numexpr                   2.7.3              py39hde0f152_0    conda-forge
```

(continues on next page)

(continued from previous page)

```
numpy                     1.20.3           py39hdbf815f_1    conda-forge
oauthlib                  3.1.1              pyhd8ed1ab_0    conda-forge
olefile                   0.46               pyh9f0ad1d_1    conda-forge
openblas                  0.3.18          pthreads_h4748800_0    conda-forge
openh264                  2.1.1                h780b84a_0    conda-forge
openjpeg                  2.4.0                hb52868f_1    conda-forge
openssl                   1.1.1o               h166bdaf_0    conda-forge
outcome                   1.2.0              pyhd8ed1ab_0    conda-forge
packaging                 21.0               pyhd8ed1ab_0    conda-forge
pamela                    1.0.0                       py_0    conda-forge
pandas                    1.3.4             py39hde0f152_0    conda-forge
pandoc                    2.15                 h7f98852_0    conda-forge
pandocfilters             1.5.0              pyhd8ed1ab_0    conda-forge
panel                     0.14.4             pyhd8ed1ab_0    conda-forge
pango                     1.48.10              h54213e6_2    conda-forge
param                     1.13.0             pyh1a96a4e_0    conda-forge
parso                     0.8.2              pyhd8ed1ab_0    conda-forge
partd                     1.2.0              pyhd8ed1ab_0    conda-forge
patsy                     0.5.2              pyhd8ed1ab_0    conda-forge
pcre                      8.45                 h9c3ff4c_0    conda-forge
pemtk                     0.0.1                    pypi_0    pypi
pexpect                   4.8.0              pyh9f0ad1d_2    conda-forge
pickleshare               0.7.5           py39hde42818_1002    conda-forge
pillow                    8.3.2             py39ha612740_0    conda-forge
pip                       21.3.1             pyhd8ed1ab_0    conda-forge
pixman                    0.40.0               h36c2ea0_0    conda-forge
plotly                    5.13.1             pyhd8ed1ab_0    conda-forge
pooch                     1.5.2              pyhd8ed1ab_0    conda-forge
poppler                   21.09.0              ha39eefc_3    conda-forge
poppler-data              0.4.12               hd8ed1ab_0    conda-forge
postgresql                13.5                 h2510834_1    conda-forge
proj                      8.1.1                h277dcde_2    conda-forge
prometheus_client         0.11.0             pyhd8ed1ab_0    conda-forge
prompt-toolkit            3.0.21             pyha770c72_0    conda-forge
protobuf                  3.18.1            py39he80948d_0    conda-forge
psutil                    5.8.0             py39h3811e60_1    conda-forge
pthread-stubs             0.4                 h36c2ea0_1001    conda-forge
ptyprocess                0.7.0              pyhd3deb0d_0    conda-forge
pybtex                    0.24.0                   pypi_0    pypi
pybtex-docutils           1.0.2                    pypi_0    pypi
pycosat                   0.6.3           py39h3811e60_1006    conda-forge
pycparser                 2.20               pyh9f0ad1d_2    conda-forge
pyct                      0.4.6                       py_0    conda-forge
pyct-core                 0.4.6                       py_0    conda-forge
pycurl                    7.44.1            py39h72e3413_0    conda-forge
pydata-sphinx-theme       0.13.1                   pypi_0    pypi
pyerfa                    2.0.0.1           py39hd257fcd_2    conda-forge
pygments                  2.14.0                   pypi_0    pypi
pygmt                     0.5.0              pyhd8ed1ab_1    conda-forge
pyjwt                     2.3.0              pyhd8ed1ab_0    conda-forge
pyopenssl                 21.0.0             pyhd8ed1ab_0    conda-forge
pyparsing                 3.0.3              pyhd8ed1ab_0    conda-forge
pyproj                    3.2.1             py39ha81a305_2    conda-forge
pyrsistent                0.17.3            py39h3811e60_2    conda-forge
pyshp                     2.3.1              pyhd8ed1ab_0    conda-forge
pyshtools                 4.10              py39hd777142_0    conda-forge
```

(continues on next page)

(continued from previous page)

```
pysocks                   1.7.1           py39hf3d152e_3    conda-forge
pytables                  3.6.1           py39h2669a42_4    conda-forge
python                    3.9.7         hb7a2778_3_cpython    conda-forge
python-dateutil           2.8.2             pyhd8ed1ab_0    conda-forge
python-json-logger        2.0.1             pyh9f0ad1d_0    conda-forge
python-kaleido            0.2.1             pyhd8ed1ab_0    conda-forge
python_abi                3.9                     2_cp39    conda-forge
pytz                      2021.3            pyhd8ed1ab_0    conda-forge
pyviz_comms               2.2.1             pyhd8ed1ab_1    conda-forge
pywavelets                1.1.1           py39hce5d2b2_3    conda-forge
pyyaml                    6.0             py39h3811e60_0    conda-forge
pyzmq                     22.3.0          py39h37b5a0c_0    conda-forge
qt                        5.12.9              hda022c4_4    conda-forge
quaternion                2022.4.2        py39hd257fcd_0    conda-forge
readline                  8.1                 h46c0cb4_0    conda-forge
reproc                    14.2.3              h7f98852_0    conda-forge
reproc-cpp                14.2.3              h9c3ff4c_0    conda-forge
requests                  2.26.0            pyhd8ed1ab_0    conda-forge
requests-unixsocket       0.2.0                     py_0    conda-forge
retrying                  1.3.3                     py_2    conda-forge
ruamel.yaml               0.17.16         py39h3811e60_0    conda-forge
ruamel.yaml.clib          0.2.2           py39h3811e60_2    conda-forge
ruamel_yaml               0.15.80       py39h3811e60_1004    conda-forge
scikit-image              0.18.3          py39hde0f152_0    conda-forge
scikit-learn              1.0.1           py39h7c5d8c9_1    conda-forge
scipy                     1.7.1           py39hee8e79c_0    conda-forge
scooby                    0.7.1             pyhd8ed1ab_0    conda-forge
seaborn                   0.9.0                     py_2    conda-forge
seaborn-base              0.11.2            pyhd8ed1ab_0    conda-forge
selenium                  4.8.2             pyhd8ed1ab_0    conda-forge
send2trash                1.8.0             pyhd8ed1ab_0    conda-forge
setuptools                58.2.0          py39hf3d152e_0    conda-forge
setuptools-scm            6.4.2             pyhd8ed1ab_0    conda-forge
shapely                   1.8.0           py39hc7dd4e9_2    conda-forge
six                       1.16.0            pyh6c4a22f_0    conda-forge
snappy                    1.1.8               he1b5a44_3    conda-forge
sniffio                   1.2.0           py39hf3d152e_1    conda-forge
snowballstemmer           2.2.0                   pypi_0    pypi
sortedcontainers          2.4.0             pyhd8ed1ab_0    conda-forge
soupsieve                 2.0.1                     py_1    conda-forge
spherical_functions       2022.4.1          pyhd8ed1ab_1    conda-forge
sphinx                    5.0.2                   pypi_0    pypi
sphinx-book-theme         1.0.0                   pypi_0    pypi
sphinx-comments           0.0.3                   pypi_0    pypi
sphinx-copybutton         0.5.1                   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
sphinxcontrib-htmlhelp    2.0.1                   pypi_0    pypi
sphinxcontrib-jsmath      1.0.1                   pypi_0    pypi
```

(continues on next page)

(continued from previous page)

```
sphinxcontrib-qthelp      1.0.3                    pypi_0    pypi
sphinxcontrib-serializinghtml 1.1.5                    pypi_0    pypi
spinsfast                 2022.4.1        py39hc8e5db4_2    conda-forge
sqlalchemy                1.4.26          py39h3811e60_0    conda-forge
sqlite                    3.36.0              h9cd32fc_2    conda-forge
statsmodels               0.13.0          py39hce5d2b2_0    conda-forge
sympy                     1.9             py39hf3d152e_0    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                 0.12.1          py39hf3d152e_0    conda-forge
testpath                  0.5.0             pyhd8ed1ab_0    conda-forge
threadpoolctl             3.0.0             pyh8a188c0_0    conda-forge
tifffile                  2021.10.12        pyhd8ed1ab_0    conda-forge
tiledb                    2.3.4               he87e0bf_0    conda-forge
tk                        8.6.11              h27826a3_1    conda-forge
toml                      0.10.2            pyhd8ed1ab_0    conda-forge
tomli                     2.0.1             pyhd8ed1ab_0    conda-forge
toolz                     0.11.1                    py_0    conda-forge
tornado                   6.1             py39h3811e60_1    conda-forge
tqdm                      4.62.3            pyhd8ed1ab_0    conda-forge
traitlets                 5.1.1             pyhd8ed1ab_0    conda-forge
trio                      0.22.0          py39hf3d152e_1    conda-forge
trio-websocket            0.10.2            pyhd8ed1ab_0    conda-forge
typing_extensions         3.10.0.2          pyha770c72_0    conda-forge
tzcode                    2022a               h166bdaf_0    conda-forge
tzdata                    2021e               he74cb21_0    conda-forge
uc-micro-py               1.0.1                    pypi_0    pypi
uncertainties             3.1.7             pyhd8ed1ab_0    conda-forge
urllib3                   1.26.7            pyhd8ed1ab_0    conda-forge
wcwidth                   0.2.5             pyh9f0ad1d_2    conda-forge
webencodings              0.5.1                     py_1    conda-forge
websocket-client          0.57.0          py39hf3d152e_4    conda-forge
werkzeug                  2.1.2             pyhd8ed1ab_1    conda-forge
wget                      3.2                      pypi_0    pypi
wheel                     0.37.0            pyhd8ed1ab_1    conda-forge
widgetsnbextension        3.5.1           py39hf3d152e_4    conda-forge
wsproto                   1.2.0             pyhd8ed1ab_0    conda-forge
x264                      1!161.3030          h7f98852_1    conda-forge
x265                      3.5                 h924138e_3    conda-forge
xarray                    2022.3.0          pyhd8ed1ab_0    conda-forge
xerces-c                  3.2.3               h9d8b166_3    conda-forge
xlrd                      2.0.1             pyhd8ed1ab_3    conda-forge
xorg-kbproto              1.0.7            h7f98852_1002    conda-forge
xorg-libice               1.0.10              h7f98852_0    conda-forge
xorg-libsm                1.2.3            hd9c2040_1000    conda-forge
xorg-libx11               1.7.2               h7f98852_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               h7f98852_1    conda-forge
xorg-libxrender           0.9.10           h7f98852_1003    conda-forge
xorg-libxt                1.2.1               h7f98852_2    conda-forge
xorg-renderproto          0.11.1           h7f98852_1002    conda-forge
xorg-xextproto            7.3.0            h7f98852_1002    conda-forge
xorg-xproto               7.0.31           h7f98852_1007    conda-forge
xz                        5.2.5               h516909a_1    conda-forge
```

(continues on next page)

(continued from previous page)

```
yaml                      0.2.5              h516909a_0    conda-forge
zeromq                    4.3.4              h9c3ff4c_1    conda-forge
zfp                       0.5.5              h9c3ff4c_7    conda-forge
zict                      2.0.0                    py_0    conda-forge
zipp                      3.6.0            pyhd8ed1ab_0    conda-forge
zlib                      1.2.11           h36c2ea0_1013    conda-forge
zstd                      1.5.0              ha95c52a_0    conda-forge
```

BIBLIOGRAPHY

[1] Paul Hockett. *Quantum Metrology with Photoelectrons, Volume 1: Foundations*. IOP Publishing, 2018. ISBN 978-1-68174-684-5. doi:10.1088/978-1-6817-4684-5.

[2] Paul Hockett. *Quantum Metrology with Photoelectrons, Volume 2: Applications and Advances*. IOP Publishing, 2018. ISBN 978-1-68174-688-3. doi:10.1088/978-1-6817-4688-3.

[3] Claude Marceau, Varun Makhija, Dominique Platzer, A. 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. doi:10.1103/PhysRevLett.119.083401.

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

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

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

[7] 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. doi:10.3233/978-1-61499-649-1-87.

[8] Brian E. Granger and Fernando Pérez. Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering*, 23(2):7–14, March 2021. doi:10.1109/MCSE.2021.3059263.

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

[10] Executable Books Community. Jupyter Book. Zenodo, February 2020. doi:10.5281/zenodo.4539666.

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

[12] Paul Hockett. ePSproc: Post-processing suite for ePolyScat electron-molecule scattering calculations. *Authorea*, 2016. doi:10.6084/m9.figshare.3545639.

[13] Paul Hockett. ePSproc: Post-processing for ePolyScat (Github repository). Github, 2016. doi:10.6084/m9.figshare.3545639.

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

[15] 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. doi:10.1016/0370-1573(86)90147-X.

[16] 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. doi:10.1063/1.467237.

[17] 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. doi:10.1063/1.479794.

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

[19] 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. arXiv:1905.06156, doi:10.1016/j.cpc.2019.107062.

[20] 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.

[21] 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. doi:10.1002/jcc.540141112.

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

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

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

[25] Stephan Hoyer and Joe Hamman. Xarray: N-D labeled Arrays and Datasets in Python. *Journal of Open Research Software*, 5(1):10, April 2017. doi:10.5334/jors.148.

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

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

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

[29] Mark A. Wieczorek and Matthias Meschede. SHTools: Tools for Working with Spherical Harmonics. *Geochemistry, Geophysics, Geosystems*, 19(8):2574–2592, August 2018. doi:10.1029/2018GC007529.

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

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

[32] Matthew Newville, Till Stensitzki, Daniel B. Allen, and Antonino Ingargiola. LMFIT: Non-Linear Least-Square Minimization and Curve-Fitting for Python. Zenodo, September 2014. doi:10.5281/zenodo.11813.

[33] Marcus Johansson and Valera Veryazov. Automatic procedure for generating symmetry adapted wavefunctions. *Journal of Cheminformatics*, 9(1):8, February 2017. doi:10.1186/s13321-017-0193-3.

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

[35] C. Yang. On the Angular Distribution in Nuclear Reactions and Coincidence Measurements. *Physical Review*, 74(7):764–772, October 1948. doi:10.1103/PhysRev.74.764.

[36] D Dill. Fixed-molecule photoelectron angular distributions. *The Journal of Chemical Physics*, 65(3):1130–1133, 1976. doi:10.1063/1.433187.

[37] 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. doi:10.1098/rsta.1963.0002.

[38] SL Altmann and 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).

[39] N Chandra. Photoelectron spectroscopic studies of polyatomic molecules. I. Theory. *Journal of Physics B: Atomic and Molecular Physics*, 20(14):3405–3415, July 1987. doi:10.1088/0022-3700/20/14/013.

[40] 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. doi:10.1063/1.466638.

[41] Tamar Seideman. Time-resolved photoelectron angular distributions: concepts, applications, and directions. *Annual review of physical chemistry*, 53:41–65, January 2002. doi:10.1146/annurev.physchem.53.082101.130051.

[42] Tamar Seideman. Time-resolved photoelectron angular distributions as a probe of coupled polyatomic dynamics. *Physical Review A*, 64(4):042504, September 2001. doi:10.1103/PhysRevA.64.042504.

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

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

[45] 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. doi:10.1063/1.461023.

[46] 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. doi:10.1039/c1cp22031d.

[47] 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. doi:DOI: 10.1016/S0368-2048(00)00148-1.

[48] 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. doi:10.1080/01442350110045046.

[49] 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. doi:10.1002/9780470259498.ch6.

[50] Katharine L. Reid and Jonathan G. Underwood. Extracting molecular axis alignment from photoelectron angular distributions. *The Journal of Chemical Physics*, 112(8):3643, 2000. doi:10.1063/1.480517.

[51] 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. doi:10.1063/1.481918.

[52] Karl Blum. *Density Matrix Theory and Applications*. Volume 64. Springer Berlin Heidelberg, Berlin, Heidelberg, 3rd editio edition, 2012. ISBN 978-3-642-20560-6. doi:10.1007/978-3-642-20561-3.

[53] 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. arXiv:2012.04561, doi:10.1088/1361-6455/ac135f.

[54] 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. doi:10.1063/5.0109607.

[55] 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. doi:10.1016/S1076-5670(03)80065-4.

[56] 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. doi:10.1088/0953-4075/44/19/192001.

[57] 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. doi:10.1088/978-0-750-31062-8.

[58] 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. doi:10.1088/0953-4075/33/13/306.

[59] 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. doi:10.1088/0953-4075/45/19/194012.

[60] 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. doi:10.1103/PhysRevLett.112.223001.

[61] 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. doi:10.1103/PhysRevA.92.013411.

[62] 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. doi:10.1088/0953-4075/48/21/214004.

INDEX