Dalton Project: A Python platform for molecular- and electronic-structure simulations of complex systems

Jógvan Magnus Haugaard Olsen, Simen Reine, Olav Vahtras, Erik Kjellgren, Peter Reinholdt, Karen Oda Hjorth Dundas, Xin Li, Janusz Cukras, Magnus Ringholm, Erik D. Hedegård, Roberto Di Remigio, Nanna H. List, Rasmus Faber, Bruno Nunes Cabral Tenorio, Radovan Bast, Thomas Bondo Pedersen, Zilvinas Rinkevicius, Stephan P. A. Sauer, Kurt V. Mikkelsen, Jacob Kongsted, Sonia Coriani, Kenneth Ruud, Trygve Helgaker, Hans Jørgen Aa. Jensen, Patrick Norman, et al.

COLLECTIONS

Paper published as part of the special topic on Electronic Structure Software

View Online   Export Citation   CrossMark

ARTICLES YOU MAY BE INTERESTED IN

Modern quantum chemistry with [Open]Molcas
The Journal of Chemical Physics **152**, 214117 (2020); https://doi.org/10.1063/5.0004835

The ORCA quantum chemistry program package
The Journal of Chemical Physics **152**, 224108 (2020); https://doi.org/10.1063/5.0004608

PSI4 1.4: Open-source software for high-throughput quantum chemistry
The Journal of Chemical Physics **152**, 184108 (2020); https://doi.org/10.1063/5.0006002
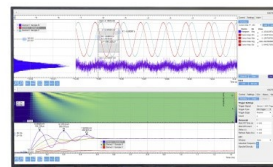
Dalton Project: A Python platform for molecular- and electronic-structure simulations of complex systems

View Online   Export Citation   CrossMark

Jógvan Magnus Haugaard Olsen,[1,a] [ID] Simen Reine,[2,b] [ID] Olav Vahtras,[3] [ID] Erik Kjellgren,[4] [ID]
Peter Reinholdt,[4] [ID] Karen Oda Hjorth Dundas,[1] [ID] Xin Li,[3] [ID] Janusz Cukras,[5] [ID] Magnus Ringholm,[1,3]
Erik D. Hedegård,[6] [ID] Roberto Di Remigio,[1] [ID] Nanna H. List,[7] [ID] Rasmus Faber,[8] [ID]
Bruno Nunes Cabral Tenorio,[8] Radovan Bast,[1] Thomas Bondo Pedersen,[2] [ID] Zilvinas Rinkevicius,[3,9] [ID]
Stephan P. A. Sauer,[10] [ID] Kurt V. Mikkelsen,[10] [ID] Jacob Kongsted,[4] [ID] Sonia Coriani,[8] [ID] Kenneth Ruud,[1] [ID]
Trygve Helgaker,[2] [ID] Hans Jørgen Aa. Jensen,[4] [ID] and Patrick Norman[3,c] [ID]

AFFILIATIONS

[1] Department of Chemistry, Hylleraas Centre for Quantum Molecular Sciences, UiT The Arctic University of Norway,
N-9037 Tromsø, Norway

[2] Department of Chemistry, Hylleraas Centre for Quantum Molecular Sciences, University of Oslo, N-0315 Oslo, Norway

[3] Department of Theoretical Chemistry and Biology, School of Engineering Sciences in Chemistry, Biotechnology and Health,
KTH Royal Institute of Technology, SE-106 91 Stockholm, Sweden

[4] Department of Physics, Chemistry and Pharmacy, University of Southern Denmark, DK-5230 Odense M, Denmark

[5] Department of Chemistry, University of Warsaw, 02-093 Warsaw, Poland

[6] Division of Theoretical Chemistry, Lund University, SE-223 62 Lund, Sweden

[7] Department of Chemistry and the PULSE Institute, Stanford University, Stanford, California 94305, USA
and SLAC National Accelerator Laboratory, Menlo Park, California 94025, USA

[8] DTU Chemistry, Technical University of Denmark, DK-2800 Kongens Lyngby, Denmark

[9] Department of Physics, Faculty of Mathematics and Natural Sciences, Kaunas University of Technology,
LT-51368 Kaunas, Lithuania

[10] Department of Chemistry, University of Copenhagen, DK-2100 Copenhagen Ø, Denmark

Note: This article is part of the JCP Special Topic on Electronic Structure Software.
[a] Electronic mail: jogvan.m.olsen@uit.no
[b] Electronic mail: simen.reine@kjemi.uio.no
[c] Author to whom correspondence should be addressed: panor@kth.se

ABSTRACT

The *Dalton Project* provides a uniform platform access to the underlying full-fledged quantum chemistry codes Dalton and LSDalton as well as the PyFraME package for automatized fragmentation and parameterization of complex molecular environments. The platform is written in Python and defines a means for library communication and interaction. Intermediate data such as integrals are exposed to the platform and made accessible to the user in the form of NumPy arrays, and the resulting data are extracted, analyzed, and visualized. Complex computational protocols that may, for instance, arise due to a need for environment fragmentation and configuration-space sampling of biochemical systems are readily assisted by the platform. The platform is designed to host additional software libraries and will serve as a hub for future modular software development efforts in the distributed Dalton community.

I. INTRODUCTION

More than 20 years have passed since the first version of the Dalton program[1] was released as a result of merging the separate HERMIT, SIRIUS, ABACUS, and RESPONS codes that implemented one- and two-electron integrals, wavefunctions, energy derivatives, and response theory, respectively. Later, the adopted monolithic code development structure turned out to be prohibitively difficult to sustain, and it was interrupted with the release of the atomic-orbital (AO) based linear-scaling initiative as a separate executable named LSDalton.[2] By the time of the Dalton paper in 2014,[3] the two codes represented a powerful general-purpose program system and provided users with access to the most relevant and standard electronic structure theory methods and, moreover, a vast amount of molecular properties. In 2017, all past and present authors of the Dalton and LSDalton codes unanimously voted in favor of open-sourcing the codes under the GNU Lesser General Public License version 2.1 (LGPLv2.1). In the present work, we will briefly recapitulate the functionalities of the codes and detail some of the developments provided in Dalton suite releases from 2015 until today, including the Dalton2020 release. With inspiration from the Molecular Sciences Software Institute (MolSSI) project,[4,5] we also take the opportunity to initiate a transition in the Dalton software engineering practices and we signal this paradigm shift by referring to the Dalton community effort as the *Dalton Project* (DP) initiative.[6] From the developer's perspective, we are taking steps to make it easier to develop, sustain, and maintain a large general-purpose software ecosystem for first-principles quantum molecular modeling of complex systems, and from the user's perspective, we are modifying the design of the user interface to enable new access and interaction patterns.

The general design strategy for the DP platform is that of software modularity[7–9] and based on a hybrid programming language approach, as illustrated in Fig. 1. We introduce an upper layer written in Python with support from specialized libraries,

FIG. 1. Overview of the Dalton Project platform structure.

such as NumPy,[10] SciPy,[11] and MPI4Py.[12] This layer is hardware-aware and capable of managing computer resources, handling user interactivity, steering computation, and performing data processing of results. The lower layer contains libraries written in a language of choice based on the programmer's preference and the task to be addressed, but compute-intensive tasks will typically be performed by libraries written in Fortran, C, or C++. The two layers interact by any one of three means of communication, namely, conventional file input/output (I/O), Python bindings, e.g., through CFFI (C Foreign Function Interface)[13] or pybind11,[14] or pure Python module import. In this scheme, we view the Dalton and LSDalton executables as libraries serving the DP platform, and although further modular library decomposition would be desirable, it is hampered by code legacy and entanglement. More important than offering this new perspective, however, the DP platform encourages future development to be made in the form of modules with clear and specific tasks (or subtasks) that undergo strict unit testing. Modules, or coherent sets of modules, build up libraries that are developed, maintained, and released independently from one another such that the DP ecosystem will see more of a continuous evolution as compared to conventional monolithic program releases. Regarding communication, it is our ambition for the ecosystem to move toward libraries that provide clear application programming interfaces (APIs) and native bindings to Python. The latter allows importing such libraries directly into Python scripts or interactive sessions, enabling fast development, read–eval–print loop (REPL) style, without sacrificing performance. We believe that this software development model will serve us well as we constitute a distributed community of contributors belonging to network nodes with different scientific objectives and timelines.

Within the field of quantum chemistry, the adoption of more modern software engineering strategies with APIs written in Python is in vogue at the moment, and we have been strongly influenced by (i) the Psi4NumPy project that exposes efficient computational kernels from the Psi4 program[15] to enable quick NumPy prototyping of novel science[16] and (ii) the PySCF program that, primarily in Python, implements self-consistent field (SCF) and post-Hartree–Fock (post-HF) electronic structure theory for finite and periodic systems.[17] Moreover, a source of inspiration as well as practical experience for the present work is provided by the VeloxChem project (and program)[18] that, with a hybrid Python/C++ programming model, implements real and complex response theory[19] at the SCF level of theory for execution in high-performance computing (HPC) cluster environments. In VeloxChem, Python is used for a split message passing interface (MPI) communicator management of large-scale distributed hardware resources with an anticipation of heterogeneous cluster nodes to become a future reality. Without noticeable sacrifice in computational efficiency or program execution stability, the higher-level quantum chemical methods and iterative linear response equation solvers are implemented in Python with the use of NumPy and underlying threaded math kernel libraries. With this as background, we have gained sufficient confidence to steer our project into a new direction as far as software engineering practices are concerned.

Our presentation is organized as follows: In Sec. II, we briefly mention some of the key features in Dalton and LSDalton that have already been presented[3] and provide a more detailed description
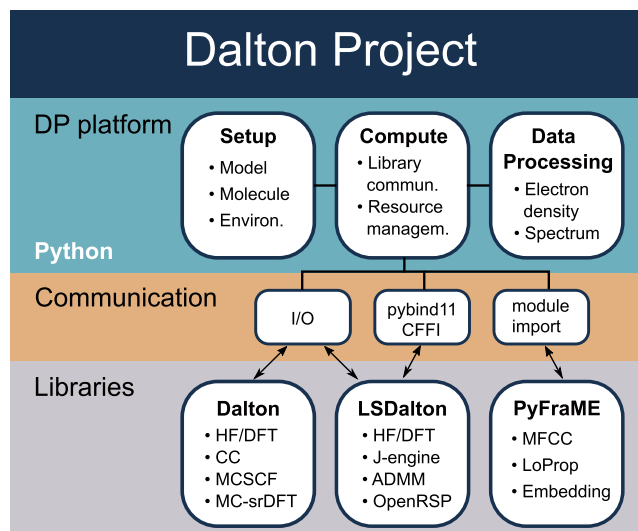
of novel functionalities that have been added thereafter. Moreover, we also present the features of the PyFraME package[20] for the handling of complex chromophore environments. In Sec. III, we give a more comprehensive view of the design of the DP platform as well as provide a concrete illustration of new library-access patterns and program execution practices for the user. In Sec. IV, we present six concrete examples of DP platform runs before closing with an outlook into the future for the *Dalton Project*.

II. CAPABILITIES OF DP PLATFORM LIBRARIES

A. Dalton and LSDalton up until 2014

In 2014, a presentation of the Dalton program system, including the Dalton and LSDalton codes, was published,[3] and functionalities listed in this presentation are, of course, still available and therefore only briefly mentioned here. The two codes are primarily written in Fortran, but parts involving density functional theory (DFT) kernels are mostly written in C. In Dalton, the routines for correlated wavefunction calculations are implemented only for serial execution—but can be linked to standard threaded linear algebra libraries—whereas the self-consistent field (SCF), i.e., HF and DFT, routines are implemented for parallel execution using MPI. LSDalton, on the other hand, comes with a native hybrid OpenMP/MPI parallelization scheme that enables shared memory data handling on central processing unit (CPU) sockets and/or compute nodes. None of the two codes come with support for hardware acceleration, such as general-purpose graphical processing units (GPUs).

The common foundation for the Dalton and LSDalton quantum chemistry programs is that of a nonrelativistic Hamiltonian, basis sets expanded in localized Gaussian AOs, and multi-electron reference states expanded in spin-restricted determinants or configuration-state functions. Relativistic corrections to the zeroth-order one-electron Hamiltonian are available in Dalton in terms of the spin-free second-order Douglas–Kroll–Hess (DKH2) Hamiltonian and effective-core potentials (ECPs). As a perturbative correction to the Hamiltonian, Dalton also offers an implementation of the full Breit–Pauli spin–orbit operator.

LSDalton provides efficient acceleration techniques for SCF-based property calculations and an implementation of the linear-scaling divide–expand–consolidate (DEC) scheme for second-order Møller–Plesset (MP2) and coupled cluster (CC) energy calculations. The code was initially developed to alleviate the restrictions of the Dalton code for calculations on large systems by introducing linear-scaling AO-based SCF and response capabilities based on an exponential *ansatz* of the AO density matrix.

Dalton provides implementations of most of the standard electronic-structure methods, including SCF, MP2, a hierarchy of CC methods [CC2, CCSD, CCSDR(3), CC3, and CCSD(T)], configuration interaction (CI), and multi-configurational SCF (MCSCF) based on the generalized active space (GAS) concept. MCSCF wavefunctions are optimized with a robust trust-region-based second-order approach.

Molecular gradients and Hessians are determined analytically for SCF and MCSCF reference states, and analytic gradients are also available at the levels of MP2, CC2, CCSD, and CCSD(T). In the absence of analytic gradients and Hessians, Dalton can determine

these quantities by numerical differentiation and thereby offers an extensive functionality for exploring potential energy surfaces. The combination of geometric and electric-field perturbations allows for calculations of infrared (IR) and Raman intensities. Analytic linear and nonlinear response functions describing the interactions with external and internal (in general, time-dependent) electromagnetic fields are implemented for the entire selection of electronic-structure methods and enable simulations of a plethora of spectroscopies, too rich to be listed here. At this time, Dalton also included the means for structure-less and atomistic descriptions of chromophore environments through the polarizable continuum model (PCM) and polarizable embedding (PE) model, respectively.

B. Added features in Dalton

The functionalities of Dalton have been expanded in several directions. Here, we provide a summary of selected new features. To bring some structure and order into these developments, we have chosen to divide them into the three categories: (i) electronic-structure theory, (ii) spectroscopy simulations, and (iii) environment modeling. In the first one, we list general quantum-chemical method developments providing new means to describe the electronic structure of ground and excited states. In the second, we describe developments that are more specifically targeting and enabling simulations of certain spectroscopies. Such simulations are connected with certain electronic-structure theory methods and typically also environment models, but the primary objective of the development has been the spectroscopy at hand. In the third, we present approaches aimed at improving the effective description of the chromophore environment. These developments are, of course, made in combination with specific electronic-structure methods, but the environment is at focus.

1. Electronic-structure theory

Based on a range-separated Hamiltonian as proposed by Savin,[21,22] a rigorous combination can be made of wavefunction and density-functional theories for the treatment of the long- and short-range electron–electron Coulomb interactions, respectively. In Dalton, this approach has been implemented at the level of MP2,[23] CI,[24,25] MCSCF,[26–28] and NEVPT2[29] wavefunction theories, and it is now made available in the Dalton2020 release. In conjunction with MCSCF, the main idea is that static (or strong) electron correlation can be effectively accounted for by means of typically quite short determinant expansions of the wavefunction at the same time as dynamic electron correlation can be effectively accounted for by means of DFT with its low computational cost. The resulting method is referred to as multi-configurational short-range DFT (MC-srDFT), and it is available for closed-shell and open-shell systems.[28] Apart from calculations of energies, linear-response properties are available for both singlet and triplet perturbations.[30–32] More details are provided in Sec. IV B where an example is provided in terms of the calculation of the ultraviolet–visible (UV/Vis) absorption spectrum of a retinylidene Schiff base chromophore.

Using Löwdin's inner projection in conjunction with a one- and two-electron excitation operator manifold and an MP2 reference state, the second-order polarization propagator approximation (SOPPA) arises as a means to address the electronic

structure of excited states. Modifications of the original form of this approach have been implemented, including the SOPPA(SCS-MP2) and SOPPA(SOS-MP2) models[33] where spin-component-scaled[34,35] and scaled opposite-spin[36] versions of the Møller–Plesset correlation coefficients are employed. Going instead toward approximations of the SOPPA model, the random phase approximation with second-order non-iterative doubles corrections model [RPA(D)][37] has been extended to triplet excitations,[38] and a similarly derived higher RPA with non-iterative doubles corrections model [HRPA(D)] has been implemented.[38] The RPA(D) and HRPA(D) models are enabled for calculations of not only transition properties but also linear response functions.[39]

Furthermore, with regard to CC approaches, Dalton also offers a new and more efficient implementation of the CC3 model for ground- and excited-state energies,[40] although it does not support full Abelian point-group symmetry, it reduces the computational cost.

2. Spectroscopy simulations

Applying the Liouville equation to pure states in the density-matrix formalism of quantum mechanics has been shown to be equivalent to applying the Ehrenfest theorem to state-transfer operators in Hilbert space, thereby leading to a means to phenomenologically introduce relaxation mechanisms into wavefunction theories.[19,41,42] The resulting complex polarization propagator (CPP) theory defines frequency-dependent response functions for exact and approximate states that are physically sound in all regions of the spectrum, resonant as well as conventional nonresonant, and also x-ray as well as conventional UV/Vis. These response functions fulfill the Kramers–Kronig relations with real and imaginary parts that are associated with separate dispersive and absorptive spectroscopies, such as optical rotatory dispersion (ORD)[43] and electronic circular dichroism (ECD).[44]

Extensions of the CPP theory have been made to allow for the description of nonlinear external-field interactions,[41,45] and the latest release of the Dalton program also offers CPP/DFT simulations of resonant-enhanced hyper-Rayleigh scattering (HRS),[46,47] magnetic circular dichroism (MCD),[48,49] magneto-chiral dichroism (MChD) and birefringence (MChB) dispersion,[50] nuclear spin-induced optical rotation (NSOR) and dichroism (NSCD),[51] and two-photon absorption (TPA) cross sections.[45,52]

Core excitation processes are associated with large valence-electron relaxation and polarization effects that, in a polarization propagator or response theory approach, require multi-electron excited configurations to be properly accounted for.[53] Along this line, Dalton provides a hierarchy of CC methods to model a variety of x-ray spectroscopies including near-edge x-ray absorption fine structure (NEXAFS),[54–57] photo-electron spectroscopy (PES),[56–59] transient x-ray absorption spectroscopy (TRXAS)[60,61] and resonant inelastic x-ray scattering (RIXS).[62] The referred-to hierarchy of CC methods includes the CCS, CC2, and CCSD levels of theory, but core-excitation and core-ionization energies are also available for the CCSDR(3) and CC3 approximations. Both singlet and triplet excited states are encompassed, and the latter are obtained in a spin-adapted formalism.[61] The core–valence separation (CVS) approximation has been made available to decouple core and valence excited states. It can be applied either at the excited-state level only[56,57] or both during the determination of the ground state and excited states in

a frozen-core variant (fc-CVS).[63] An example illustration of a DP platform XAS calculation using the CVS approximation is provided in Sec. IV F.

3. Modeling of chromophore environments

The capabilities in Dalton for including effects from a molecular environment have been extended in several directions. The PCM for efficient modeling of bulk solvent effects can now also be performed at the SOPPA level.[64] In this PCM-SOPPA/RPA model, the static solvent contributions are treated at the SOPPA level, while the dynamic solvent contributions are evaluated at the time-dependent HF level. The PCM model can also be used in combination with the MC-srDFT method.

The PE model[65,66] is a fragment-based (semi-)quantum–classical scheme designed for efficient and accurate inclusion of environment effects in calculations of spectroscopic properties of large and complex molecular systems. The environment is included effectively through an embedding potential whose parameters consist of distributed multipoles and polarizabilities, both of which are derived from quantum-mechanical calculations on the individual fragments that make up the environment. The PyFraME package, which is made available on the DP platform and is described in Sec. II D, can be used to automatize the generation of the embedding-potential parameters. The PE model is implemented in the Polarizable Embedding library (PElib)[67] based on an AO density-matrix-driven formulation, which facilitates a loose-coupling modular implementation in host programs. The PElib was included in the Dalton2013 release, but at that time, it was limited to PE-HF and PE-DFT.[65,66] Since then the implementation has been extended to PE-CC [specifically, PE-CC2, PE-CCSD, and PE-CCSDR(3)],[68] PE-MCSCF,[69] PE-MC-srDFT,[70] and PE-SOPPA.[71] The Dalton2020 release supports linear-, quadratic-, and cubic-response properties for PE-HF/DFT,[66] while PE-CC is limited to linear- and quadratic-response properties, and only linear-response properties are available for PE-MCSCF and PE-MC-srDFT. For PE-HF/DFT, it is also possible to compute properties based on resonant-convergent response theory.[72] London AOs (LAOs) are supported for magnetic linear-response properties that involve a single derivative with respect to a magnetic field.[73] The capabilities have also been extended to enable analytic quantum-mechanical molecular gradients at the PE-HF/DFT level, thus enabling geometry optimization of the core quantum region embedded in a fixed polarizable environment.[74] Local-field effects may also be included in PE-HF/DFT calculations where they are termed effective external field (EEF) effects.[75,76] Electronic energy transfer (EET) couplings can be calculated based on the PE model, including both direct and environment-induced contributions, and using QFITLIB[77] to derive transition-density-fitted multipoles.[78] Bulk solvation effects can be included through the FixSol conductor-like solvation model using the FIXPVA2 cavity tessellation scheme.[79,80] An overview of the developments related to the PE model can be found in Ref. 81, while a tutorial review is available in Ref. 82.

The PE model, and classical models in general, does not include Pauli repulsion between the chromophore and its environment. Such models can therefore suffer from so-called electron spill-out, where the electron density of the chromophore leaks out into the environment, thus causing an over-stabilization of the ground state

and, in particular, the excited states of the embedded chromophore. Negatively charged chromophores or excited states of even partial Rydberg-like character are especially susceptible.[83,84] The polarizable density embedding (PDE) model has been formulated to improve the electrostatic interactions between the chromophore and its environment and to address the electron spill-out issue.[85,86] In this model, the permanent charge distribution of the fragments in the environment is described by their full electronic densities, thus avoiding divergences of the multipole expansions, while still keeping the distributed polarizabilities to efficiently account for polarization effects. In addition, the PDE model contains a Huzinaga–Cantu-like projection operator[87] that models Pauli repulsion effects and thereby effectively prevents electron spill-out. The PDE model has been implemented in PElib using the same AO density-matrix-driven formulation as for the PE model. It can therefore straightforwardly be combined with the same DFT and wavefunction methods as the PE model both in terms of ground-state and response calculations, with the exception of LAOs and analytic molecular gradients.

The latest Dalton release has also received basic frozen density embedding (FDE)[88,89] capability. The FDE implementation enables import of a static embedding potential that has been pre-calculated on a numerical integration grid by another code that implements FDE.[90,91] A matrix representation of the embedding potential is constructed based on the grid and added to the one-electron Fock matrix. The implementation can thus be used with all available DFT and wavefunction methods in Dalton.[92,93]

C. Added features in LSDalton

1. Integral evaluation

Integrals sit at the heart of any quantum chemistry program, both when it comes to computational performance and available methods and properties. The development of an efficient and flexible integral-evaluation code has therefore been essential to the development of LSDalton. Since 2014,[3] four main integral developments have been added: high-order derivative integrals (HODI), integrals and differentiated integrals for embedding techniques involving interaction with point charges and higher-order multipoles, acceleration of the exchange contribution through developments of the auxiliary-density-matrix method (ADMM),[94] and interface with the XCFun library of DFT exchange–correlation (XC) functionals.[95,96] The XCFun library is based on forward-mode automatic differentiation[97] and can therefore generate arbitrary-order derivatives of these functionals.

The one- and two-electron HODI implementation employs the solid-harmonic Hermite scheme of Ref. 98, allowing for a unified scheme for undifferentiated and differentiated integrals by expanding the solid-harmonics in Hermite rather than Cartesian Gaussians; differentiation merely increments one of the quantum numbers of the Hermite Gaussians, whereas differentiation of Cartesian Gaussians gives linear combinations of Cartesian Gaussians. The HODI integrals have been extended to allow interactions with general-order point multipoles (charges, dipoles, and so on) needed for classical embedding techniques.

The exchange contribution is the main computational bottleneck in hybrid DFT calculations. The development of efficient and accurate acceleration techniques for the exchange

contribution will thus greatly improve overall DFT timings and increase the scope and applicability of DFT in general. One such approach is the ADMM,[94,99,100] where the time-critical exchange contribution is instead evaluated in a smaller basis, and corrected with the difference between the local generalized-gradient approximation (GGA) exchange in the full and the small basis. The ADMM has been implemented in LSDalton with different variants for the projection to the smaller basis and GGA correction functional options,[99] and with tailored auxiliary basis sets (admm-n) for the pcseg-n and aug-pcseg-n basis sets.[100]

2. Exploiting the locality of electron correlation

The DEC[101–103] strategy employs highly local orbitals[104,105] to recover the inherent locality of dynamical correlation for large molecules in a linear-scaling fashion. Over the last few years, the DEC framework has been extensively developed and now includes resolution-of-the-identity (RI) accelerated MP2 (DEC-RI-MP2[106–108]), Laplace-transformed RI-MP2 (DEC-LT-RI-MP2[109]), CC theory through DEC-CCSD and DEC-CCSD(T),[110] and through the multilayer DEC framework ML-DEC,[111] which allows for efficient calculations by systematic treatment of the *pair-fragment* at different levels of theory. In addition to energies, densities, and electrostatic potentials, gradients are available at the DEC-MP2 and DEC-RI-MP2 levels,[102,107] and excitation energies are available through the local framework for calculating excitation energies (LoFEx)[112–114] and the correlated natural transition orbital framework for a low-scaling excitation energy (CorNFLEx) approach.[115] Due to the embarrassingly parallel nature of the DEC scheme, excellent scalability to a large number of CPU cores is possible. As an example, a DEC-RI-MP2/cc-pVDZ gradient calculation of the insulin molecule (787 atoms and 7604 basis functions) finished in less than 10 h using 32 000 cores (2000 nodes, each with 16 cores, on the Titan supercomputer).[107]

3. Molecular properties

Several property developments have been undertaken since 2014, including quasi-Newton transition-state optimization, the high-order path-expansion (HOPE)[116] method for improved geometry-optimization steps, automated counterpoise correction, and the same-number-of-optimized-parameters (SNOOP)[117,118] scheme as an improved alternative to the counterpoise correction, and nuclear-selected NMR shielding,[119] to mention a few.

On a longer-term development line, LSDalton has been interfaced with OpenRSP,[120] to allow, in principle, arbitrary-order molecular properties. OpenRSP is an open-ended response-theory library that manages the generation and solution of the response equations needed for the evaluation of arbitrary-order response properties. The current implementation in LSDalton enables the calculation of a sizable selection of (mixed) electrical and geometrical properties for HF and DFT, for the latter also involving an interface to the XCFun and XCint[121] libraries. This includes properties related to IR, Raman, and hyper-Raman spectral intensities, molecular gradients, Hessians, and cubic force constants. The capabilities of the LSDalton/OpenRSP/XCint/XCFun combination are illustrated for the calculation of IR and Raman spectra of benzene through the DP platform in Sec. IV D.

For modeling of solvent effects, the PCMSolver[122] library for continuum electrostatic solvation has been interfaced to LSDalton. This implementation is available for SCF and electric-dipole response properties up to fourth order.

D. PyFraME: Python framework for Fragment-based Multiscale Embedding

PyFraME[20] is a Python package providing a framework for managing fragment-based multiscale embedding calculations. The basic principle of embedding models in quantum chemistry is the division of a molecular system into two domains: a central core region and its environment. The core region is described at the highest level of theory using DFT or a wavefunction method, while the effects from the environment are included effectively through an embedding potential. To manually set up embedding calculations of large and complex molecular systems can be highly complicated, tedious, and error-prone. This is especially true when considering that configuration-space sampling, e.g., through molecular dynamics (MD) simulations, is usually required, which, in turn, means that the procedure has to be repeated many times.

The highly flexible PyFraME package automatizes workflows, starting from the initial molecular structure to the final embedding potential. It enables the user to easily set up a multilayered description of the environment. Each layer can be described either by a standard embedding potential, i.e., using a predefined set of parameters, or by deriving the embedding-potential parameters based on first-principles calculations. For the latter, a fragmentation method is used to subdivide large molecular structures into smaller computationally manageable fragments. The number of layers, as well as the composition and level of theory used for each layer, can be fully customized.

The basic workflow consists of three main steps. First, a molecular structure is given as an input. Currently, PyFraME supports input files in the PDB format. The input file reader extracts information about the structure and composition of the system, and it also defines the basic units of the system, i.e., fragments. Small molecules would typically constitute a fragment on their own, but larger molecules are usually broken down into small computationally manageable fragments. For example, for proteins, a fragment would usually consist of an amino-acid residue, while for nucleic acids, it could be a nucleotide. The molecular system to be used for the embedding calculation is then built by extracting subsets from the full list of fragments according to specified criteria, such as name, chain ID, distance, or a combination thereof, and placed into separate regions. As mentioned above, any number of regions may be added, and each can be fully customized. Once the system has been built, the final step is the derivation of the embedding potential. Depending on the specifics, it may involve a large number of separate calculations on the individual fragments in order to compute the embedding-potential parameters. For large molecules, where the parameters cannot be computed directly, PyFraME uses a fragmentation method based on the molecular fractionation with conjugate caps (MFCC) approach[123] to derive the parameters. The individual fragment calculations are typically performed by Dalton and the LoProp Python package,[124,125] but this can be customized. The fragmentation of the system, fragment calculations, and subsequent joining of parameters to build the embedding potential are fully automated and can make full use of large-scale HPC resources.

III. DP PLATFORM DESIGN AND FEATURES

The ultimate goal of the DP platform is to establish a flexible, robust, and uniformly accessible environment that can be used for both large-scale applications and to facilitate the development of novel methodology. The challenges for quantum chemistry today are far more complex than earlier, both concerning the complexity of the chemical systems, adaptation to HPC facilities, and the number of tools and approaches needed for applications. As a result, it becomes essential to be able to easily combine the tools and approaches in meaningful ways. The main motivation of the Dalton Project is to provide a platform that can be used to combine the functionality of the tools and methods that are developed by the individual research groups in the Dalton community.

For a long time, we have relied on a monolithic codebase (first Dalton and later also LSDalton) for the development of new computational methodology. These programs have served us well in the past, and we expect this to continue into the foreseeable future. It is clear, however, that the codebase has accumulated substantial technical debt. The tight coupling between the software modules is particularly problematic because it complicates optimization and modernization of even small pieces of code. Moreover, implementation of new methodology often requires unnecessarily high efforts and easily leads to additional technical debt. The risks of relying on a monolithic codebase are especially high when the codebase is maintained by a scientific community such as ours whose primary goal is to perform research. In recognition of this, and the fact that individual groups have different research aims and preferences in terms of software development, we have in later years moved toward a more distributed codebase. This has resulted in the development of a series of software libraries, such as Gen1Int,[126,127] OpenRSP,[120] PCMSolver,[122] PElib,[67] QcMatrix,[128] XCFun,[95,96] and XCint.[121] This has, to some degree, alleviated the problem of the monolithic codebase for some developments, but the main issues remained.

We have now taken the next step and moved completely to a distributed codebase with the Dalton Project, whose main task is to integrate and provide interoperability between the individual software libraries that are developed and maintained by the different nodes in our community. At the same time, however, we acknowledge that there is vast functionality developed in our community during the last few decades that we do not wish to abandon, which is primarily implemented in Dalton and LSDalton. The DP platform thus has to accommodate a wide variety of software from large monolithic programs with a wide range of features to small libraries that provide very specific functionality. The design of the platform has to take this into account in a sustainable manner, so that it can act as a platform for present-day use cases and, importantly, for future developments based on modern software engineering practices. Moreover, the DP platform must be able to exploit current HPC facilities and be prepared for the upcoming exascale supercomputers.

To meet our goals and requirements, we devised a platform structure that is illustrated in Fig. 1. At the top, we have the DP platform itself, written in pure Python (3.6+), that interfaces to external libraries through different communication mechanisms. Python was chosen as the platform language because of its extensibility, emphasis on code readability, and comprehensive standard library, as well

as a large number of specialized libraries. We note here that the term library is used liberally to signify any type of software that can be interfaced to the platform, including libraries in the traditional sense, program executables, and Python modules and packages. The libraries thus require very different means of communication, and to accommodate this, we provide three different mechanisms: file I/O, Python bindings, and pure Python imports. The file I/O communication mechanism is provided to make the vast functionality implemented in Dalton and LSDalton more accessible. In fact, most of the functionality provided on the DP platform currently involves Dalton and LSDalton, but we will gradually move toward using loosely coupled libraries written in pure Python or hybrid Python and Fortran, C, or C++, with the hybrid approach being used for the more compute-intensive numerical tasks. Initially, the DP platform will interface Dalton and LSDalton as well as PyFraME, all of which have been described in Sec. II. In the immediate future, we expect that many of the aforementioned libraries that are currently interfaced to Dalton and LSDalton will be interfaced directly to the DP platform.

The DP platform is a Python package with an API that consists of a set of classes and functions used to set up molecular systems, perform numerical calculations, and process data. Usage of the platform would typically consist of three stages used in succession, namely, *setup*, *compute*, and *data processing*. By separating the compute and data processing stage, the DP platform may be easily employed in large-scale application workflows in which a number of calculations are typically run first, by submitting them to a queuing system on a supercomputer, and subsequently data manipulation, analysis, and visualization are performed.

The setup stage consists of instantiation of one or more of the five base classes: `Molecule`, `Basis`, `QCMethod`, `Property`, and `Environment`, which are used in the compute and data processing stages. The classes have been designed to be library-agnostic so that they can be used and reused for all libraries. However, not all classes are necessarily needed. It depends on the specific type of calculation that is performed. For example, the first four (or all five if an environment model is used) are needed to run, e.g., a TPA calculation employing Dalton or LSDalton, whereas other functionality may only need some of them (e.g., more fine-grained functionality can be obtained from LSDalton, as illustrated in Sec. IV A).

The `Molecule` class contains information about the molecular structure, which can be a single atom, a molecule, a fragment of a molecule, or a set of molecules. It requires, as a minimum, that atomic elements and coordinates are defined. Reasonable defaults are used for other attributes, such as the total charge, spin multiplicity, atomic masses, atomic labels, and, if enabled, information related to point-group symmetry. The atomic elements, coordinates, and, optionally, labels, can be given either as a file, e.g., in XYZ format (optionally with atom labels in the fifth column), or as a string. The DP platform also provides a function that can read the standard molecule file format used by Dalton and LSDalton and return instances of the `Molecule` and `Basis` classes. The atomic labels are used in the `Basis` class as described below, but also to specify, e.g., ghost atoms and point charges.

The `Basis` class holds all basis-set information, which includes the main basis set and, optionally, auxiliary basis sets used for, e.g., RI and ADMM approximations. The basis set can be given either as

a string, in which case the specified basis set is used for all atoms, or as a dictionary using atom labels as the keys and basis sets as the corresponding values. The basis sets are obtained from the basis set exchange (BSE) Python package.[129]

The `QCMethod` class specifies the method, for example, HF, DFT, MCSCF, and CC, together with any associated attributes, such as XC functional or definition of active space. Approximations used to compute Coulomb and exchange contributions are also given here (and also require that the corresponding basis sets are defined in a `Basis` instance). In addition, this class is used to specify additional settings, such as convergence thresholds, maximum number of iterations, and so on.

The `Property` class is used to define which properties to compute. This includes single-point energy, geometry optimization, excitation energies, and so on, as well as additional specifications related to the property, which could be, e.g., the algorithm to use in the geometry optimization or the number of states to include in the calculation of excitation energies. Currently, the DP platform supports only a limited set of properties out of the great number that are available in Dalton and LSDalton, but this will be continuously extended. A selection of some of the current capabilities is demonstrated in the illustrations presented in Sec. IV.

The `Environment` class defines the environment, if present, surrounding a molecule or fragment, which is defined in a `Molecule` instance. It contains information about the type of environment model, e.g., PCM, PE, or PDE, as well as all the parameters and settings belonging to the specific model. For example, for the PE model, this class contains the coordinates of the classical sites and the associated multipoles and polarizabilities.

The libraries are used in the compute and/or data processing stages. For each library, there is an interface provided as a subpackage of the main DP package. The interface API consists of functions that allow the user to interact directly with the libraries. The exact implementation of the interface varies depending on the nature of the library, e.g., what functionality it provides and how the API of the library itself is defined. However, the interface API functions that are exposed to the user must conform to the standards laid out by the DP platform to ensure that there is uniform access to all libraries as well as interoperability between them. This means, for example, that libraries with similar functionality must also provide API functions with identical names and signatures. Moreover, apart from the classes defined by the DP platform, the types and data structures must be either Python built-ins, e.g., integers, floats, lists, and dictionaries, or NumPy arrays.

The default ordering of AO integrals on the DP platform is the Dalton ordering: atoms are ordered according to the user input, and for each atom it is angular-momentum components first and contracted functions second. Other AO orderings may be used on the platform, but the interface API must provide transformation functions to and from the default Dalton ordering. Transformations can then be made on the platform in the cheapest way possible, e.g., on the MO coefficients rather than on the four-center integrals for MP2.

Numerical compute-intensive tasks are performed at the compute stage. This can include anything from the calculation of integral components, all the way to a complete calculation of a molecular property, as well as more complex workflow protocols. In the first development release of the DP platform, users will be able to directly

calculate one- and two-electron integrals using LSDalton through CFFI-based Python bindings and have seamless access to a selection of wavefunctions and properties computed by Dalton and LSDalton using the file I/O communication mechanism. To deal with complex molecular systems that are too large for a full quantum-chemical treatment, we provide an interface to PyFraME that enables workflows involving fragmentation and quantum–classical embedding.

The results obtained at the compute stage can be used directly at the data processing stage, which includes data extraction, manipulation, analysis, and visualization. The feature set that will be available in the first development release includes the ability to perform vibrational analysis to obtain vibrational frequencies and normal modes, natural orbital occupation analysis to assist in the selection of active spaces, and partitioning of properties into atomic and interatomic contributions using the LoProp approach. In addition, the DP platform has capabilities for plotting spectra and visualizing orbitals, electrostatic potentials, and electronic densities.

The DP platform can automatically detect and manage available hardware resources. Manual specification is also possible and allows fine-grained control. In auto-detection mode, the DP platform will first check for resources reserved through a standard HPC job scheduler and, if no scheduler is found, fall back to use resources on the local computer. Resource usage is optimized according to the capabilities of the used libraries, exploiting OpenMP, MPI, or hybrid OpenMP/MPI parallelism when available. Moreover, a task-farming functionality is provided for use cases where many separate calculations are to be performed.

We conclude this section with a brief walk-through example that illustrates how the DP platform can be used to streamline a workflow going from initial molecular structures to TPA spectra, employing both Dalton and LSDalton. In Sec. IV, additional illustrations are presented to demonstrate the fine-grained access to integrals and show some of the new features available on the DP platform.

Assuming that the DP platform has been imported as `import daltonproject as dp` and a list of XYZ filenames is available in `xyz_filenames`, we start by creating a list of `Molecule` instances

```
molecules = []
for filename in xyz_filenames:
    molecule = dp.Molecule(input_file=filename)
    molecules.append(molecule)
```

Then, we create the `Basis`, `QCMethod`, and `Property` instances that will be used for a geometry optimization of the molecules using LSDalton. The order in which the classes are instantiated is unimportant. We here start with `QCMethod`

```
b3lyp = dp.QCMethod(qc_method='DFT',
                    xc_functional='B3LYP',
                    coulomb='DF',
                    exchange='ADMM')
```

specifying that we want to use the B3LYP XC functional together with density-fitting (DF) and ADMM to accelerate the calculation of Coulomb and exchange contributions, respectively. Both acceleration techniques require an auxiliary basis set, which is specified when instantiating the `Basis` class

```
small_basis = dp.Basis(basis='pcseg-1',
                       ri='def2-universal-JKFIT',
                       admm='admm-1')
large_basis = dp.Basis(basis='pcseg-2',
                       ri='def2-universal-JKFIT',
                       admm='admm-2')
```

Finally, we create a `Property` instance

```
geo_opt = dp.Property(geometry_optimization=True)
```

using the default optimization algorithm. With these instances we can proceed to the compute stage.

For Dalton and LSDalton, when used through the file I/O mechanism, we use the `compute()` function. It returns an `OutputParser` instance that contains methods for transparently fetching relevant results from the output files, as shown further below. The `compute()` function creates a unique filename based on the specific input, which is stored in the `OutputParser` instance. The filename can also be specified through an optional argument in which case it is up to the user to ensure its uniqueness.

We can iterate through the list of molecules, one by one, and perform both the pre-optimization and final optimization in the same loop (updating the molecule coordinates after each step)

```
for molecule in molecules:
    pre = dp.lsdalton.compute(molecule=molecule,
                              basis=small_basis,
                              qc_method=b3lyp,
                              properties=geo_opt)
    molecule.coordinates = pre.final_geometry
    final = dp.lsdalton.compute(molecule=molecule,
                                basis=large_basis,
                                qc_method=b3lyp,
                                properties=geo_opt)
    molecule.coordinates = final.final_geometry
```

LSDalton can exploit the available CPU resources using a hybrid OpenMP/MPI scheme. The optimal use for LSDalton is typically one MPI process per CPU and one OpenMP thread per CPU core, up to a maximum of 20 threads. For example, on a supercomputer with two CPUs per node and 12 cores per CPU, LSDalton will use two MPI processes per node and 12 OpenMP threads per MPI process.

The optimized molecular structures are passed to Dalton for the calculation of TPA cross sections employing the CAM-B3LYP functional

```
camb3lyp = dp.QCMethod(qc_method='DFT',
                       xc_functional='CAM-B3LYP')
```

and a mixed basis set which is defined through a dictionary

```
basis_dict = {'H': 'pcseg-2',
              'C': 'aug-pcseg-2',
              'O': 'aug-pcseg-2'}
mixed_basis = dp.Basis(basis=basis_dict)
```

It is here assumed that the molecules only contain hydrogens, carbons, and oxygens, and that the XYZ files do not contain atomic labels in the fifth column in which case the atomic labels default to

the element symbols. Finally, we create a new `Property` instance for the TPA cross section

```
tpa = dp.Property(two_photon_absorption=True)
tpa.two_photon_absorption(states=4)
```

where we have additionally specified that we want to include four states. We then proceed with the calculation of the TPA cross sections using Dalton, collecting the results for each molecule in the `tpa_results` list

```
tpa_results = []
for molecule in molecules:
    result = dp.dalton.compute(molecule=molecule,
                               basis=mixed_basis,
                               qc_method=camb3lyp,
                               properties=tpa)
    tpa_results.append(result)
```

By default, Dalton will here adopt a purely MPI-parallel execution, corresponding to one MPI process per CPU core. Alternatively, the task farming functionality can be used by supplying the list of molecules to the `compute()` function

```
tpa_results = dp.dalton.compute(molecule=molecules,
                                basis=mixed_basis,
                                qc_method=camb3lyp,
                                properties=tpa)
```

In this scenario, a separate calculation will run for each molecule concurrently, dividing the available CPU resources among them.

Finally, we can plot the TPA spectra using the spectrum module of the DP platform

```
axs = []
for result in tpa_results
    ax = dp.spectrum.plot_two_photon_spectrum(result)
    axs.append(ax)
```

where `ax` is an instance of the Matplotlib[130] `Axes` class. Further customization can then be performed to produce publication-ready figures, which can be plotted as normally done with Matplotlib.

IV. DP PLATFORM ILLUSTRATIONS

In this section, we provide six use cases of the DP platform with the intent to demonstrate novel platform functionalities in terms of data exposure, processing, and visualization as well as to illustrate some of the added features of the platform libraries. The Python scripts used for these DP platform illustrations, along with the corresponding input/output files, are deposited at https://doi.org/10.5281/zenodo.3710462.

A. NumPy-exposure of one- and two-electron integrals

We here demonstrate how to access primitive and contracted integrals on the DP platform. This functionality is made available by LSDalton through CFFI-based Python bindings and enables access to a variety of one- and two-electron integrals including Coulomb and exchange integral matrices. We will soon add exposure of geometrically differentiated integrals (in principle, to arbitrary order), integrals for embedding techniques involving interaction with classical charges and higher-order multipoles, Gaussian-geminal type integrals needed for F12-type theories, attenuated two-electron integrals, multipole-moment integrals, and magnetically differentiated London integrals (to first order for the two-electron integrals).

Integrals and integral components are available on the DP platform in the form of NumPy arrays and currently require instances of `Molecule`, `Basis`, and `QCMethod` classes, as outlined in Sec. III. For example,

```
h2o = dp.Molecule('O  0.000000  0.000000  0.000000;'
                  'H  0.758602  0.000000  0.504284;'
                  'H  0.758602  0.000000 -0.504284')
pcs1 = dp.Basis('pcseg-1')
dft = dp.QCMethod('DFT', 'B3LYP')
```

where `import daltonproject as dp` is assumed. In the following, we also import the LSDalton module: `import daltonproject.lsdalton as lsd`. We are then ready to compute basic DFT integral components, such as one-electron matrices

```
S = lsd.overlap_matrix(h2o, pcs1, dft)
h = lsd.kinetic_matrix(h2o, pcs1, dft) \
    + lsd.nuclear_electron_attraction_matrix(h2o, pcs1, dft)
D = lsd.diagonal_density(h, S, h2o, pcs1, dft)
```

two-electron matrices

```
J = lsd.coulomb_matrix(D, h2o, pcs1, dft)
K = lsd.exchange_matrix(D, h2o, pcs1, dft)
E_xc, V_xc = lsd.exchange_correlation(D, h2o, pcs1, dft)
g = 2.0 * J - K + V_xc
F_ks = h + g
```

Molecular-orbital energies and coefficients can then be obtained, e.g, using SciPy

```
E_mo, C_mo = scipy.linalg.eigh(a=F_ks, b=S)
```

Two-electron four-center integrals can be obtained through

```
eri = lsd.eri4(h2o, pcs1, dft)
```

and, for example, two- and three-center RI integrals, by specifying an auxiliary RI basis, through

```
basis = dp.Basis('pcseg-1', 'def2-universal-JKFIT')
ab_alpha = lsd.ri3(h2o, basis, dft)
alpha_beta = lsd.ri2(h2o, basis, dft)
```

With these integrals, we can easily construct the resolution-of-the-identity (RI) Coulomb matrix
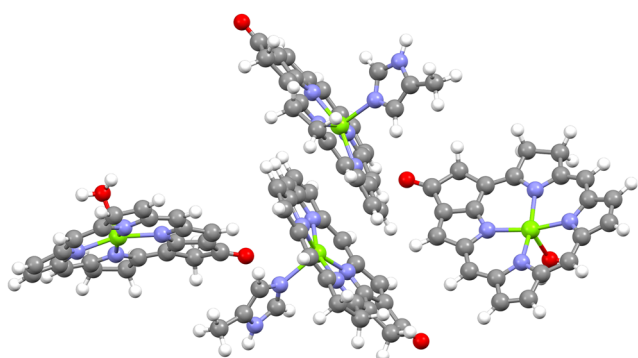
FIG. 2. The structure of the tetrameric model consisting of units of the P700 pigment of photosystem I.

$$\widetilde{J}_{ab} = \sum_{cd} \sum_{\alpha,\beta} (ab|\alpha)(\alpha|\beta)^{-1}(\beta|cd) D_{cd}, \tag{1}$$

through the sequence

$$g_\beta = \sum_{cd} (\beta|cd) D_{cd},$$
$$c_\alpha = \sum_\beta (\alpha|\beta)^{-1} g_\beta, \tag{2}$$
$$\widetilde{J}_{ab} = \sum_\alpha (ab|\alpha) c_\alpha$$

according to

```
g_beta  = np.einsum('cdB,cd', ab_alpha, density)
AB_inv  = np.linalg.inv(alpha_beta)
c_alpha = np.einsum('AB,B', AB_inv, g_beta)
J_ab    = np.einsum('A,abA', c_alpha, ab_alpha)
```

We note that this example is only meant to illustrate the ease with which algorithms can be implemented on the platform and that there are better ways to construct the density-fitted Coulomb

matrix in Eq. (1) (see, for instance, Ref. 131 and references therein).

We end this illustration with a concrete example of a CAM-B3LYP/pcseg-2 single-point energy calculation of a tetrameric model of the P700 pigment of photosystem I,[132] depicted in Fig. 2, to illustrate the computational performance of the LSDalton library. This triple-zeta quality P700 tetramer model consists of 198 atoms and 4744 contracted basis functions. The wall time, with 32 Intel E5-2683v4 dual socket nodes (1024 cores), was 15 min. The calculation used J-engine accelerated density-fitting of the Coulomb contribution and the ADMM2 variant of ADMM for the exchange contribution, and employed one MPI process per CPU and 16 OpenMP threads per process. For more details about the acceleration techniques, consult Ref. 100, where these techniques were studied more extensively.

B. Combined treatment of static and dynamic electron correlation: The multi-configurational short-range density functional theory method

The calculation of UV/Vis spectra with the MC-srDFT method is here illustrated with the retinylidene Schiff base chromophore, as originally addressed in previous works.[31,70] Spectra for this system are shown in Fig. 3 together with the π-orbitals that constitute the active space, denoted by $\pi_1 - \pi_6$. We first note that range-separated calculations introduce a range-separation parameter μ that affects the results as long as one remains short of the full-CI limit. From a practical point of view, the optimal μ-value is that which delivers reliable results with the least amount of computational effort. Benchmark studies have shown that a value around $\mu = 0.4\ a_0^{-1}$ is close to optimal for ground and excited states.[26,133,134] We have adopted this value for the present illustration.

The following workflow was employed in the spectrum calculations: First, the natural orbitals were calculated from the MP2-srPBE ground-state wavefunction and the occupation numbers were used to select a suitable active space. The DP platform provides seamless processing of the MP2-srPBE results, and, based on a user-defined selection criterion, an automatic selection of strongly and weakly occupied orbitals is made. Our adopted CAS(6,6) active space came as a result of including orbitals with occupation numbers
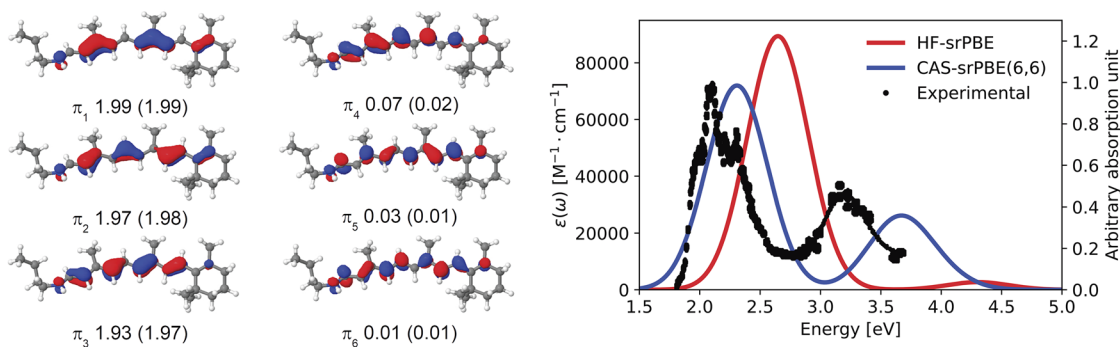

FIG. 3. UV/Vis spectrum of the retinylidene Schiff base chromophore. Results are obtained at the level of CAS(6,6)-srPBE/6-31+G* with $\mu = 0.4\ a_0^{-1}$. Both CAS and MP2 (in parentheses) occupation numbers are given. The experimental spectrum presented in arbitrary units is taken from Ref. 135.

below 1.99 (occupied space) and above 0.01 (virtual space). Second, the CAS(6,6)-srPBE calculation is performed. Third, the DP platform processes data by extracting excitation energies and oscillator strengths and generates the requested absorption spectrum. The following definition of the frequency-dependent molar absorption coefficient $\varepsilon(\omega)$ was used:

$$\varepsilon(\omega) = \frac{e^2 \pi^2 N_A}{\ln(10) 2\pi\epsilon_0 nm_e c} \sum_i \frac{\omega f_i}{\omega_i} g(\omega, \omega_i, \gamma_i), \quad (3)$$

where N_A is Avogadro's constant, e is the elementary charge, m_e is the electron mass, c is the speed of light, ϵ_0 is the vacuum permittivity, n is the refractive index (here set to unity), f_i is the calculated oscillator strength of the ith transition, and ω_i is the corresponding transition angular frequency. Equation (3) also introduces the line-broadening function g with the phenomenological parameter γ_i corresponding to the half-width at half-maximum (HWHM). The DP platform offers spectral broadenings based on normalized Gaussian or Lorentzian line profiles.

With the basis of the results in Fig. 3, we briefly discuss some key features of the MC-srDFT method. We note that the CAS-srPBE spectrum shows two distinct peaks in good agreement with the experimentally observed S_1 and S_2 bands[135,136]—transition energies and oscillator strengths are provided in Table I. It is here essential to employ a multi-configurational wavefunction as can be seen by the comparison to the single-determinant (HF-srPBE) spectrum, which essentially corresponds to the range-separated LC-PBE model. Without account of static correlation, peak positions are strongly blue-shifted and the intensity of the second band is severely underestimated. It is noted that occupation numbers from a regular MP2 calculation suggest significantly larger active spaces.[137] However, the use of CAS(6,6)-srPBE provides the same accuracy as literature CASPT2 results based on a much larger CAS(12,12) active space.[138] This demonstrated opportunity to employ relatively small active spaces with MC-srDFT is not limited to the retinylidene Schiff base chromophore, but has also been shown in other contexts, e.g., to describe the mechanism of [NiFe]-hydrogenase.[139]

The reason behind the failure of single-reference DFT approaches to properly describe the electronic transition underlying

the S_2 band in the retinylidene Schiff base chromophore becomes apparent from an analysis of the MC-srDFT response vectors. For S_1, the dominant element of the response vector refers to the configuration associated with a single-electron excitation from π_3 to π_4 (CI coefficient of 0.78 in Table I). For S_2, on the other hand, there are three almost equally important configurations appearing in the response vector, one of which is associated with a double-electron excitation from π_3 to π_4 (CI coefficient of 0.42 in Table I) that is unaccounted for in standard time-dependent DFT.

C. Modeling complex systems through fragment-based quantum–classical approaches

We here provide an illustration that demonstrates not only the ability of the DP libraries to perform spectrum simulations of *complex systems* but also the ability and potential of the hosting DP platform to manage the workflow of *complex computational protocols* associated with environment fragmentation and configuration-space sampling. It is an indisputable fact that first-principles methods in quantum chemistry come with a computational cost that hampers applications to relevant chemical and biochemical systems such as solutions and protein-embedded chromophores. Methods that allow for low-cost approximate yet accurate modeling of environments are therefore scientifically enabling, and one such approach is the PE model briefly described in Sec. II B 3. In the PE scheme, the environment is represented by atom-centered multipoles (typically up to and including quadrupoles) and atom-centered anisotropic dipole–dipole polarizabilities that allow for a mutual polarization between the quantum part and the classical environment.

As further described in Sec. II B 3, the adopted multipole expansion in the standard PE formulation is in the PDE model replaced with an exact Coulomb interaction as well as a description of non-electrostatic exchange repulsion. The total embedding operator in PDE consists of terms that describe the electrostatic, induction, and exchange-repulsion effects from the environment onto the core quantum region. The electrostatic operator contains the Coulomb interaction with the electrons and nuclei of the environment. For the construction of the electrostatic operator, density-matrix elements of the environment and intermolecular (core–fragment) two-electron integrals are required. The repulsion operator models the effects of exchange repulsion between the quantum core and environment fragment wavefunctions through a projection operator that scales as the square of the intermolecular overlap. The PDE model shows clear advantages over the standard PE model. It effectively avoids artificial stabilization of diffuse excited states and electron spill-out.

Both the PE and PDE models are applicable to large and complex (bio-)molecular systems. Since the parameters describing the environment are derived based on a fully *ab initio* description, the setup of the spectrum calculation involves a large number of preliminary calculations on the separate fragments of the environment. Figure 4 illustrates this situation in terms of TPA spectrum calculations of the Nile red chromophore embedded in the β-lactoglobulin protein. Including the solvent, this system contains 32 582 classical sites. The entire protein and water molecules within 15 Å of the Nile red chromophore were treated with either PE or PDE, while water molecules beyond this distance were

TABLE I. Vertical excitation energies (eV) and oscillator strengths (in parentheses) for the two lowest singlet states in the retinylidene Schiff base chromophore.

Method	S_1	S_2
HF-srPBE	2.62 (1.980)	4.29 (0.060)
CAS(6,6)-srPBE	2.28 (1.592)	3.62 (0.525)
Expt.[135,136]	2.03	3.22
Assignment[a]	CI coeff.	CI coeff.
$\pi_2(\uparrow) \rightarrow \pi_4(\uparrow)$	0.30	−0.53
$\pi_3(\uparrow) \rightarrow \pi_4(\uparrow)$	0.78	0.46
$\pi_3(\uparrow\downarrow) \rightarrow \pi_4(\uparrow\downarrow)$	−0.25	0.42

[a]Largest CI coefficients in the CAS(6,6)-srPBE response vectors. Iso-density orbital plots are shown in Fig. 3.
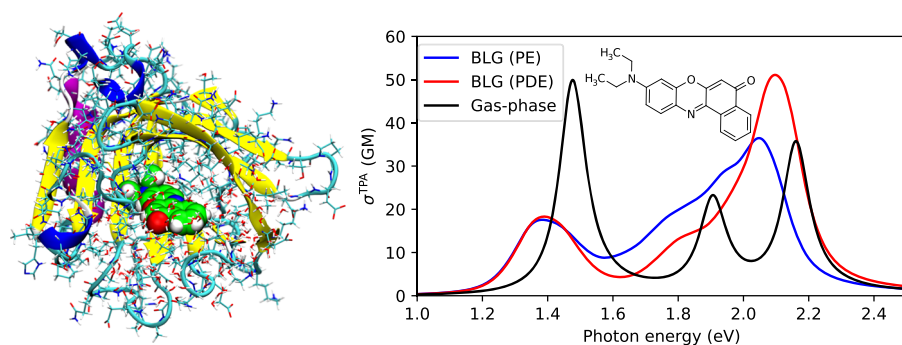
FIG. 4. The left panel shows the Nile red chromophore embedded in the β-lactoglobulin (BLG) protein. The right panel shows the simulated TPA spectra of the Nile red in vacuum and embedded in the protein (described using the PE or PDE models). Results are obtained at the CAM-B3LYP/6-31+G* level of theory.

described using the TIP3P water model (in total 10 000 water molecules).

Configuration sampling of the environment was performed by averaging over 150 snapshots sampled from a quantum mechanics/molecular mechanics (QM/MM) MD simulation.[140] The calculation of embedding-potential parameters for each snapshot took *circa* 1 h and 4 h for PE and PDE, respectively, on 20 nodes (dual socket E5-2680v3, 12 cores). The TPA spectrum calculations (five lowest singlet states) included EEF effects and took *circa* 4.5 h and 5.0 h per snapshot for PE and PDE, respectively, on eight nodes.

Assuming a monochromatic laser source, the TPA cross section (in units of GM) is computed as[141]

$$\sigma^{\text{TPA}}(\omega) = \frac{8\pi^3 \alpha a_0^5}{c} \sum_i \omega^2 \delta_i^{\text{TP}} g(2\omega, \omega_i, \gamma_i), \tag{4}$$

where α is the fine-structure constant, a_0 is the Bohr radius, and g is here chosen to be a Lorentzian with a HWHM of $\gamma = 0.1$ eV [see also Eq. (3)]. The two-photon (TP) transition strength of the ith transition is computed assuming linearly polarized light as

$$\delta_i^{\text{TP}} = \frac{1}{15} \sum_{a,b} \left(2 S_{ab} S_{ab}^* + S_{aa} S_{bb}^* \right), \tag{5}$$

where S_{ab} are the associated TP transition matrix elements.

The gas-phase TPA spectrum of Nile red shows three distinct peaks at 1.5 eV, 1.9 eV, and 2.2 eV. Embedded in the protein, the lowest band becomes red-shifted by 0.1 eV, and it also becomes both broader and lower in intensity. At higher energies, the PDE model preserves the structure from the gas phase partially, with an intense peak at 2.1 eV, and a shoulder at 1.7 eV. In contrast, the PE model predicts a large peak at 2.1 eV with a broad shoulder toward lower energies. This occurs due to the lack of non-electrostatic repulsion, which leads to a high density of states in this region due to artificial stabilization of higher-lying excited states.

D. Open-ended response theory for electric and geometric perturbations: Infrared and Raman spectroscopy with the OpenRSP and SpectroscPy modules

Spectroscopic techniques involving molecular vibrations are useful for characterizing molecular systems, and with the DP platform, an option for the calculation of vibrational spectra is available

through the combined use of LSDalton, OpenRSP,[120] and SpectroscPy.[142] We here illustrate this functionality through a calculation of IR and Raman spectra of benzene, including a brief outline of the key aspects of the LSDalton/OpenRSP/SpectroscPy combination, and its use on the platform.

From the DP platform, energy-derivative tensors generated by LSDalton/OpenRSP are passed on to SpectroscPy that processes them to generate spectroscopic properties. OpenRSP manages the calculation of response properties by an open-ended quasienergy-based formulation of response theory,[143] employing a recursive formalism.[144] OpenRSP has an API through which it can be connected to different host programs. OpenRSP is currently called through LSDalton, which provides the necessary integral derivatives (briefly outlined in Sec. II C 1), XC contributions through modules XCFun[95] and XCint,[121] and response equation solver capability.[145] The use of QcMatrix[128] allows OpenRSP to be agnostic to the details of the matrix operations, i.e., independent of their specific implementation when passing matrices from/to a host program and when carrying out matrix operations inside the OpenRSP core functionality. SpectroscPy is used to produce spectroscopic properties involving molecular vibrations by performing vibrational analysis generating vibrational frequencies and absorption properties. Presently, SpectroscPy offers functionality for IR and Raman spectroscopy in the harmonic approximation and can also combine data from calculations on a series of molecular configurations, which is useful, for example, in applications dealing with flexible molecular systems that require configuration-space sampling. Future extensions will include anharmonic corrections, hyper-Raman spectroscopy, and other spectroscopic processes.

The IR molar decadic absorption coefficient for normal mode i can, in the double harmonic approximation, be expressed as[19]

$$\varepsilon_i(\bar{v}) = \frac{N_A}{12 \ln(10) \varepsilon_0 c^2} \sum_\alpha^{x,y,z} \left(\frac{\partial \mu_\alpha}{\partial Q_i} \right)^2 g(\bar{v}; \bar{v}_i, \gamma_i), \tag{6}$$

where μ is the molecular dipole moment, Q_i is the normal mode coordinate i, and g is here chosen as a Lorentzian [see also Eq. (3)]. The molar absorption coefficient is a function of the wavenumber $\bar{v}$ and depends parametrically on the vibrational wavenumbers $\bar{v}_i$ and the HWHM broadening γ_i of mode i. Derivatives are evaluated at the equilibrium geometry.
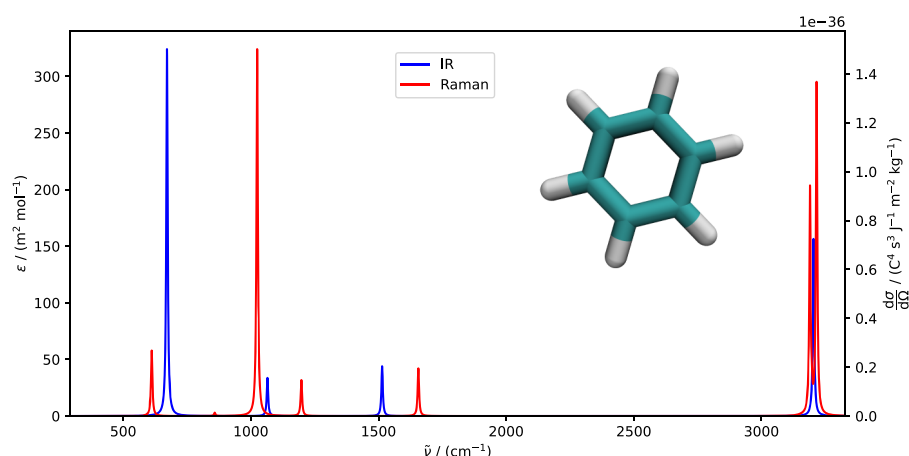
FIG. 5. IR molar decadic absorption coefficient and Raman scattering cross section of benzene using a common HWHM of 3.2 cm^{-1}. The spectra were calculated at the PBE0/pcseg-2 level of theory. The Raman scattering cross sections were calculated for $\bar{\nu}_0 = 2.195 \times 10^4$ cm^{-1} and T = 298 K.

Similarly, the harmonic differential Raman scattering cross section $\sigma_i{}'$ is given by[146]

$$\sigma_i'(\bar{\nu}) = \frac{d\sigma_i(\bar{\nu})}{d\Omega} = \frac{h(\bar{\nu}_0 - \bar{\nu}_i)^4}{16\pi^3 c^2 \bar{\nu}_i \left(1 - \exp\left[-\frac{hc\bar{\nu}_i}{kT}\right]\right)}$$
$$\times \left(45 a_i'^2 + 7 b_i'^2\right) g(\bar{\nu}; \bar{\nu}_i, \gamma_i), \tag{7}$$

where h is Planck's constant, $\bar{\nu}_0$ is the wavenumber of the incident light, k is the Boltzmann constant, and T is the temperature. We have here introduced

$$a_i' = \frac{1}{3}\sum_{\alpha}^{x,y,z} \frac{\partial \alpha_{\alpha\alpha}}{\partial Q_i}; \quad b_i'^2 = \sum_{\alpha}^{x,y,z}\sum_{\beta\neq\alpha}^{x,y,z}\left[\frac{1}{2}\left(\frac{\partial \alpha_{\alpha\alpha}}{\partial Q_i} - \frac{\partial \alpha_{\beta\beta}}{\partial Q_i}\right)^2 + 3\left(\frac{\partial \alpha_{\alpha\beta}}{\partial Q_i}\right)^2\right], \tag{8}$$

where α is the molecular dipole–dipole polarizability.

The spectra shown in Fig. 5 were generated by SpectroscPy using the molecular Hessian and the first-order geometrical derivatives of the molecular dipole moment and polarizability calculated by LSDalton/OpenRSP. SpectroscPy first calculated harmonic vibrational frequencies by carrying out an eigenanalysis of the molecular Hessian[147] and projecting out translational and rotational degrees of freedom.[148] The requested intensity-related quantities were then calculated and plotted as a function of the frequency. The DP platform allows for the whole pipeline to be run in an automated fashion. The libraries are seamlessly connected through the platform, which thus allows for running the calculations, processing the data, and visualizing the results through a simple Python script.

E. Open-shell properties free from spin-contamination: The restricted–unrestricted response theory formalism

One of the unique capabilities of Dalton is the ability to compute various linear and nonlinear response properties of open-shell systems at the spin-restricted open-shell Kohn–Sham (ROKS) level of theory,[149,150] and the DP platform provides the means for seamless and immediate visualization of spin densities to facilitate the interpretation of the results. The spin-restricted formalism ensures that the ground-state electron density is free from spin-contamination, and, as such, it provides a better starting point for molecular property calculations compared to the more widely used unrestricted Kohn–Sham (UKS) approach. The advantages of ROKS over UKS are most apparent in calculations of electron paramagnetic resonance (EPR) spin Hamiltonian parameters, which are explicitly dependent on an effective expectation value of the electronic spin operator, and benchmark studies on organic radicals show that the ROKS approach is able to better predict electronic g-tensors and hyperfine coupling constants.[151–154]

The main strength of the ROKS approach is the ability to produce a spin-contamination free electron density for the high-spin ground state. This is achieved by imposing spin-symmetry restrictions during the SCF optimization.[149] The side effect of these restrictions is that the hereby obtained KS orbitals have a non-vanishing gradient with respect to orbital rotations of triplet spin-symmetry,[152,155] and this side effect manifests itself in the computation of electronic spin-dependent properties, such as hyperfine coupling constants. For example, the expectation value of the one-electron spin-dependent operator $\hat{A}$ in the ROKS approach is given by the direct spin-density and spin-polarization contributions,[152]

$$\langle \hat{A} \rangle = \mathrm{Tr}(\mathbf{A}\,\mathbf{D}_{\mathrm{spin}}) + \mathrm{Tr}(\mathbf{A}\,\mathbf{D}_{\mathrm{pol}}); \quad A_{ij} = \langle \phi_i | \hat{A} | \phi_j \rangle, \tag{9}$$

where the first contribution is computed in the same way as in the UKS approach by contracting the electron spin-density $\mathbf{D}_{\mathrm{spin}}$ with operator matrix $\mathbf{A}$ in the AO basis and the second contribution is computed similarly to the first contribution by replacing $\mathbf{D}_{\mathrm{spin}}$ with the spin-polarization density $\mathbf{D}_{\mathrm{pol}}$.

The spin-polarization density, $\mathbf{D}_{\mathrm{pol}}$, is determined by solving restricted–unrestricted response equations that account for the relaxation of KS orbitals in the presence of the spin-dependent perturbation. The relative importance of the spin-polarization density contribution varies greatly between different molecular properties: from being prominent for isotropic hyperfine coupling constants[152,155] to being negligible for electronic g-tensors.[154]

The use of the spin and spin-polarization densities as obtained from the restricted–unrestricted response formalism is not limited to the calculation of spin-dependent molecular properties, but
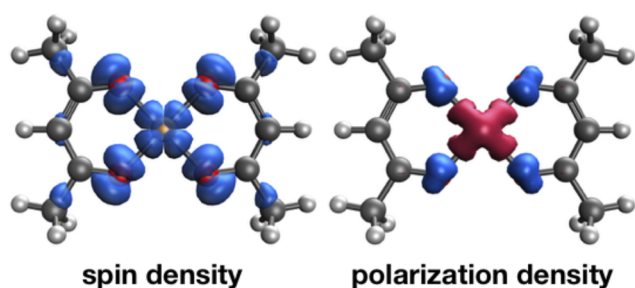
FIG. 6. Spin-density and spin-polarization densities for the $^2B_{1g}$ ground-state of the copper acetylacetonate complex, Cu(AcAc)$_2$. Isodensity surfaces are based on volumetric data files generated on the DP platform by performing restricted–unrestricted response theory calculations of the isotropic hyperfine coupling constant. At the ROKS/B3LYP/Wachtersa+f/Huz-III level of theory, the isotropic hyperfine coupling constant of the copper atom, A(^{63}Cu), amounts to 539 MHz and only the spin-polarization contributes to its value.

can also be employed for topological analyses of open-shell system responses to spin-dependent perturbations. To illustrate this point, we depict the isodensity surfaces of the spin and spin-polarization in Fig. 6 for the $^2B_{1g}$ ground state of the copper acetylacetonate complex, Cu(AcAc)$_2$. The spin-density contribution for Cu(II) complexes is expected to primarily stem from the $3d_{xy}$-orbital of the copper atom,[154] but it is here seen to also acquire significant contributions from the coordinating oxygen atoms. The spin-polarization density is also delocalized across the copper and oxygen atoms, and based on the localization of these two densities, one can predict the behavior of spin-dependent properties and qualitatively estimate the importance of the spin-density and the spin-polarization contributions. We emphasize that the presented disentangled visualization of spin-density and spin-polarization contributions to molecular properties is uniquely accessible in the

restricted–unrestricted response formalism and not available in the UKS approach.

F. Coupled cluster methods for inner-shell spectroscopy

As an illustration of the use of the DP platform for the simulation of x-ray spectroscopies, we present in Fig. 7 the NEXAFS spectrum of acrolein. We here adopt the CCSD level of theory in conjunction with the CVS approximation, as implemented in Dalton (see Sec. II B 2). Based on transition energies and oscillator strengths from CVS-CCSD response theory, the absorption spectrum is determined from Eq. (3) with the use of a Lorentzian line shape function and an HWHM broadening of 0.27 eV for all transitions.

The three lowest excitations at the near carbon edge correspond to the first two bands in the experimental spectrum, These excitations are assigned to $1s \rightarrow \pi^*$ transitions from the three carbon atoms by means of a natural transition orbital (NTO) analysis. The transition associated with the carbonyl group is chemically blue shifted by some 1.5 eV from the other two (see Fig. 7). There is an overall excellent spectrum agreement between theory and experiment, and, although simple, this example illustrates well the value of spectrum simulations for the characterization and interpretation of experiments. The most prominent added value of the DP platform for ground-state x-ray spectroscopy simulations using Dalton comes at this stage of analysis and visualization.

In the example above, the CVS approximation is employed only during the determination of the core-excited states. It is implemented as a projector that, during the iterative solution of the CC eigenvalue problem,[56,57] $\mathbf{A}R^f = \omega_f R^f$, only retains elements R_{aI}^f, $R_{aI,bj}^f$, $R_{ai,bJ}^f$, and $R_{aI,bJ}^f$ (CCSD case), where I and J refer to core orbitals and a and b refer to virtual orbitals. A schematic representation of the CVS approximation is shown in Fig. 8, where only the green matrix
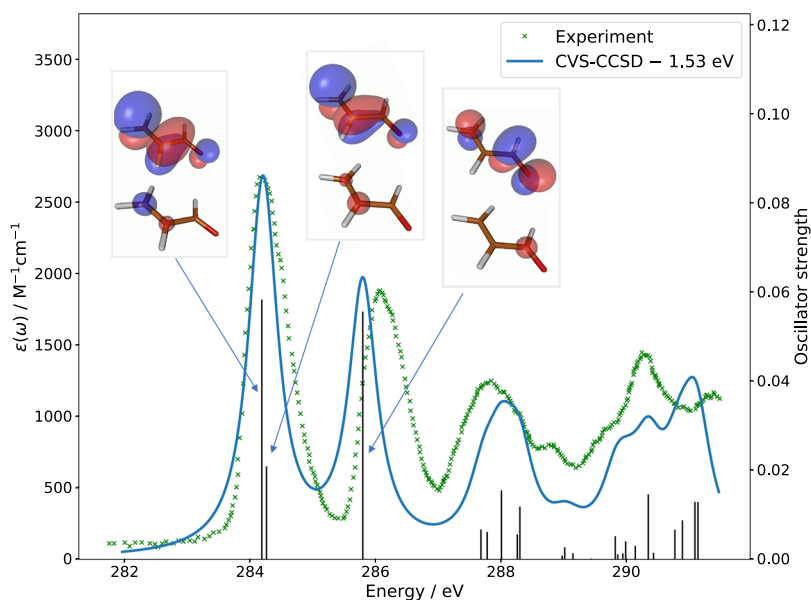

FIG. 7. Near carbon K-edge x-ray absorption spectrum of acrolein, C$_3$H$_4$O, obtained at the CVS-CCSD/6-311++G(d,p) level of theory. The theoretical spectrum is red shifted by 1.53 eV, and the experimental spectrum is taken from Ref. 156 and presented in arbitrary units. Natural transition orbitals for the first three excitations, all of $1s \rightarrow \pi^*$ character, are shown.
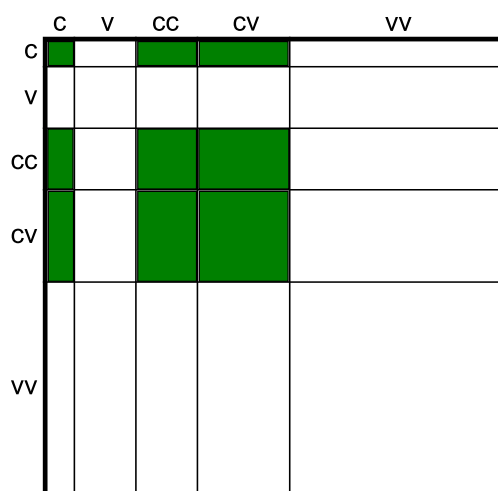
FIG. 8. Schematic representation of the CC Jacobian in the CVS approximation where only the green matrix sub-blocks are considered in solving the eigenvalue equation. The illustration refers to the case where single and double excitations are present. Labels c and v refer to occupied core and valence orbital indices, respectively.

sub-blocks are effectively kept after applying the projector during the iterative procedure of the response solver. Core and valence excitations become decoupled and x-ray spectra are obtained at basically the same computational cost as UV/Vis spectra with the use of identical bottom-up iterative techniques. The ability of a specific CC method to reproduce specific core-excitation spectral signatures depends on the amount of relaxation as well as of single, double, or higher excitation character of the transition. CCSD generally yields core spectra in rather good (semi-quantitative) agreement with experiment, with rigid blue-shifts ranging in between 0.5 eV and 3 eV, depending on the K-edge considered and the basis set adopted.

In addition to facilitating data analysis and visualization, the DP platform greatly simplifies the more complicated setup involved with simulations of transient (excited-state) x-ray absorption and emission spectra. Due to design legacies in Dalton, such simulations without the use of the DP platform will require a substantial amount of manual file handling as the necessary valence and core excitation vectors must be obtained from separate code executions and then later recombined. On the platform, all of these steps are handled in an automated way.

V. OUTLOOK

We have given a presentation of the *Dalton Project* that marks a paradigm shift in the software engineering practices for the Dalton community. At the heart of the Dalton Project, we find a hardware-aware platform written in Python with support from NumPy, SciPy, and MPI4Py, which provides a modern user interface and defines a communication standard for seamless library access and interoperability. At present, the DP platform supports three libraries, namely, Dalton, LSDalton, and PyFraME, but further modular extensions

are underway. Libraries are written in any of the programming languages Python, Fortran, C, or C++, and they communicate with the platform by means of file I/O, Python bindings through CFFI or pybind11, or pure Python module import. Modular programming with enforced unit testing will become the standard for newly started developments, and it is to be anticipated that the existing few mono-lithic codes will gradually be phased out and replaced by a larger number of libraries with more specific tasks. The DP platform is open source and distributed under the GNU General Public License version 3.0 or later (GPLv3+).

The DP platform is developed for execution on personal computers as well as more powerful supercomputers in HPC environments, and, although without guarantees, the user should expect it to run under Windows, MacOS, and Linux operating systems equipped with Python 3.6 (or later) installations (see the DP website https://daltonproject.org for instructions). Installation of the DP libraries is a separate issue, and the platform will gracefully signal to the user when it encounters called for but missing libraries. It is, of course, fully possible to use the DP platform alone on a personal computer to analyze the results available in output files produced on a remote HPC system. As our model to reach a sustainable software ecosystem adopts the notion of separate and quite independent release and distribution policies for the libraries, the main burden of work and the most dependency issues in the installation process are expected to be found in the phase of library installation. Driven by the stimulus of becoming recognized and used, it is anticipated and expected that newly started library developments will care to offer a smooth installation process on a widespread selection of platforms and operating systems.

For developers, the DP platform can lead to rapid prototyping of novel scientific ideas, as illustrated in Sec. IV A with an examination of the RI approximation. However, this example also points out something else of great importance, namely, the educational aspect of the DP platform. Our experience tells us that the process of implementing methods to solve fundamental equations is supremely efficient to reach a deeper understanding of the topic at hand, but only few students are granted this opportunity as core program modules of scientific software were written a long time ago and often made obscure by code optimization and entanglement. What is here illustrated is the access to the needed building blocks to explore quantum chemistry in very much the same manner that we can use the Python NumPy package to explore linear algebra. At the early stage of a Ph.D. education, we believe that this can be of high value and help overcome some of the initial hurdles faced during a career in quantum chemical theory and program development.

REFERENCES

[1]Dalton, a molecular electronic structure program, Release v2020.0 (2020), see https://daltonprogram.org/.

[2]LSDalton, a linear scaling molecular electronic structure program, Release v2020.0 (2020), see https://daltonprogram.org/.

[3]K. Aidas, C. Angeli, K. L. Bak, V. Bakken, R. Bast, L. Boman, O. Christiansen, R. Cimiraglia, S. Coriani, P. Dahle, E. K. Dalskov, U. Ekström, T. Enevoldsen, J. J. Eriksen, P. Ettenhuber, B. Fernández, L. Ferrighi, H. Fliegl, L. Frediani, K. Hald, A. Halkier, C. Hättig, H. Heiberg, T. Helgaker, A. C. Hennum, H. Hettema, E. Hjertenaes, S. Høst, I.-M. Høyvik, M. F. Iozzi, B. Jansík, H. J. Aa. Jensen, D. Jonsson, P. Jørgensen, J. Kauczor, S. Kirpekar, T. Kjaergaard, W. Klopper, S. Knecht, R. Kobayashi, H. Koch, J. Kongsted, A. Krapp, K. Kristensen, A. Ligabue, O. B. Lutnaes, J. I. Melo, K. V. Mikkelsen, R. H. Myhre, C. Neiss, C. B. Nielsen, P. Norman, J. Olsen, J. M. H. Olsen, A. Osted, M. J. Packer, F. Pawlowski, T. B. Pedersen, P. F. Provasi, S. Reine, Z. Rinkevicius, T. A. Ruden, K. Ruud, V. V. Rybkin, P. Sałek, C. C. Samson, A. S. de Merás, T. Saue, S. P. A. Sauer, B. Schimmelpfennig, K. Sneskov, A. H. Steindal, K. O. Sylvester-Hvid, P. R. Taylor, A. M. Teale, E. I. Tellgren, D. P. Tew, A. J. Thorvaldsen, L. Thøgersen, O. Vahtras, M. A. Watson, D. J. Wilson, M. Ziolkowski, and H. Ågren, Wiley Interdiscip. Rev.: Comput. Mol. Sci. 4, 269 (2014).

[4]The Molecular Sciences Software Institute (MolSSI) (2020), see https://molssi.org/.

[5]A. Krylov, T. L. Windus, T. Barnes, E. Marin-Rimoldi, J. A. Nash, B. Pritchard, D. G. A. Smith, D. Altarawy, P. Saxe, C. Clementi, T. D. Crawford, R. J. Harrison, S. Jha, V. S. Pande, and T. Head-Gordon, J. Chem. Phys. 149, 180901 (2018).

[6]Dalton Project: A Python platform for molecular- and electronic-structure simulations of complex systems (2020), see https://daltonproject.org/.

[7]K. Roberts, Comput. Phys. Commun. 1, 1 (1969).

[8]E. W. Dijkstra, Commun. ACM 11, 341 (1968).

[9]D. L. Parnas, Commun. ACM 15, 1053 (1972).

[10]S. van der Walt, S. C. Colbert, and G. Varoquaux, Comput. Sci. Eng. 13, 22 (2011).

[11]P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt, Nat. Methods 17, 261 (2020).

[12]L. Dalcín, R. Paz, M. Storti, and J. D'Elía, J. Parallel Distrib. Comput. 68, 655 (2008).

[13]A. Rigo and M. Fijalkowski, CFFI: C Foreign Function Interface for Python, 2018, see https://cffi.readthedocs.io/.

[14]W. Jakob, J. Rhinelander, and D. Moldovan, pybind11—Seamless operability between C++11 and Python, 2017, see https://github.com/pybind/pybind11.

[15]R. M. Parrish, L. A. Burns, D. G. A. Smith, A. C. Simmonett, A. E. DePrince, E. G. Hohenstein, U. Bozkaya, A. Y. Sokolov, R. Di Remigio, R. M. Richard, J. F. Gonthier, A. M. James, H. R. McAlexander, A. Kumar, M. Saitow, X. Wang, B. P. Pritchard, P. Verma, H. F. Schaefer, K. Patkowski, R. A. King, E. F. Valeev, F. A. Evangelista, J. M. Turney, T. D. Crawford, and C. D. Sherrill, J. Chem. Theory Comput. 13, 3185 (2017).

[16]D. G. A. Smith, L. A. Burns, D. A. Sirianni, D. R. Nascimento, A. Kumar, A. M. James, J. B. Schriber, T. Zhang, B. Zhang, A. S. Abbott, E. J. Berquist, M. H. Lechner, L. A. Cunha, A. G. Heide, J. M. Waldrop, T. Y. Takeshita, A. Alenaizan, D. Neuhauser, R. A. King, A. C. Simmonett, J. M. Turney, H. F. Schaefer, F. A. Evangelista, A. E. DePrince, T. D. Crawford, K. Patkowski, and C. D. Sherrill, J. Chem. Theory Comput. 14, 3504 (2018).

[17]Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, S. Wouters, and G. K. L. Chan, Wiley Interdiscip. Rev.: Comput. Mol. Sci. 8, e1340 (2018).

[18]Z. Rinkevicius, X. Li, O. Vahtras, K. Ahmadzadeh, M. Brand, M. Ringholm, N. H. List, M. Scheurer, M. Scott, A. Dreuw, and P. Norman, Wiley Interdiscip. Rev.: Comput. Mol. Sci. e1457 (published online 2019).

[19]P. Norman, K. Ruud, and T. Saue, Principles and Practices of Molecular Properties (John Wiley & Sons, Ltd., Chichester, UK, 2018).

[20]J. M. H. Olsen, PyFraME: Python framework for Fragment-based Multiscale Embedding, 2020, see https://gitlab.com/FraME-projects/PyFraME.

[21]A. Savin, in Recent Developments and Applications of Modern Density Functional Theory, edited by J. Seminario (Elsevier, 1996), Vol. 4, pp. 327–357.

[22]A. Savin and H.-J. Flad, Int. J. Quantum Chem. 56, 327 (1995).

[23]E. Fromager and H. J. Aa. Jensen, Phys. Rev. A 78, 022504 (2008).

[24]J. K. Pedersen, "Description of correlation and relativistic effects in calculations of molecular properties," Ph.D. thesis, University of Southern Denmark, 2004.

[25]E. D. Hedegård, S. Knecht, J. S. Kielberg, H. J. Aa. Jensen, and M. Reiher, J. Chem. Phys. 142, 224108 (2015).

[26]E. Fromager, J. Toulouse, and H. J. Aa. Jensen, J. Chem. Phys. 126, 074111 (2007).

[27]E. Fromager, F. Réal, P. Wåhlin, U. Wahlgren, and H. J. Aa. Jensen, J. Chem. Phys. 131, 054107 (2009).

[28]E. D. Hedegård, J. Toulouse, and H. J. Aa. Jensen, J. Chem. Phys. 148, 214103 (2018).

[29]E. Fromager, R. Cimiraglia, and H. J. Aa. Jensen, Phys. Rev. A 81, 024502 (2010).

[30]E. Fromager, S. Knecht, and H. J. Aa. Jensen, J. Chem. Phys. 138, 084101 (2013).

[31]E. D. Hedegård, F. Heiden, S. Knecht, E. Fromager, and H. J. Aa. Jensen, J. Chem. Phys. 139, 184308 (2013).

[32]E. R. Kjellgren, E. D. Hedegård, and H. J. Aa. Jensen, J. Chem. Phys. 151, 124113 (2019).

[33]S. P. A. Sauer, H. F. Pitzner-Frydendahl, M. Buse, H. J. Aa. Jensen, and W. Thiel, Mol. Phys. 113, 2026 (2015).

[34]S. Grimme, J. Chem. Phys. 118, 9095 (2003).

[35]S. Grimme, L. Goerigk, and R. F. Fink, Wiley Interdiscip. Rev.: Comput. Mol. Sci. 2, 886 (2012).

[36]Y. S. Jung, R. C. Lochan, A. D. Dutoi, and M. Head-Gordon, J. Chem. Phys. 121, 9793 (2004).

[37]O. Christiansen, K. L. Bak, H. Koch, and S. P. A. Sauer, Chem. Phys. Lett. 284, 47 (1998).

[38]P. A. B. Haase, R. Faber, P. F. Provasi, and S. P. A. Sauer, J. Comput. Chem. 41, 43 (2020).

[39]A. K. Schnack-Petersen, P. A. B. Haase, R. Faber, P. F. Provasi, and S. P. A. Sauer, J. Comput. Chem. 39, 2647 (2018).

[40]R. H. Myhre and H. Koch, J. Chem. Phys. 145, 044111 (2016).

[41]P. Norman, D. M. Bishop, H. J. Aa. Jensen, and J. Oddershede, J. Chem. Phys. 123, 194103 (2005).

[42]P. Norman, Phys. Chem. Chem. Phys. 13, 20519 (2011).

[43]P. Norman, K. Ruud, and T. Helgaker, J. Chem. Phys. 120, 5027 (2004).

[44]A. Jiemchooroj and P. Norman, J. Chem. Phys. 126, 134102 (2007).

[45]T. Fahleson and P. Norman, J. Chem. Phys. 147, 144109 (2017).

[46]B. F. Milne, P. Norman, F. Nogueira, and C. Cardoso, Phys. Chem. Chem. Phys. 15, 14814 (2013).

[47]B. F. Milne and P. Norman, J. Phys. Chem. A 119, 5368 (2015).

[48]H. Solheim, K. Ruud, S. Coriani, and P. Norman, J. Phys. Chem. A 112, 9615 (2008).

[49]H. Solheim, K. Ruud, S. Coriani, and P. Norman, J. Chem. Phys. 128, 094103 (2008).

[50]J. Cukras, J. Kauczor, P. Norman, A. Rizzo, G. L. J. A. Rikken, and S. Coriani, Phys. Chem. Chem. Phys. 18, 13267 (2016).

[51]J. Vaara, A. Rizzo, J. Kauczor, P. Norman, and S. Coriani, J. Chem. Phys. 140, 134103 (2014).

[52]T. Fahleson, H. Ågren, and P. Norman, J. Phys. Chem. Lett. 7, 1991 (2016).

[53]P. Norman and A. Dreuw, Chem. Rev. 118, 7208 (2018).

[54]S. Coriani, O. Christiansen, T. Fransson, and P. Norman, Phys. Rev. A 85, 022507 (2012).

[55] S. Coriani, T. Fransson, O. Christiansen, and P. Norman, J. Chem. Theory Comput. **8**, 1616 (2012).

[56] S. Coriani and H. Koch, J. Chem. Phys. **143**, 181103 (2015).

[57] S. Coriani and H. Koch, J. Chem. Phys. **145**, 149901 (2016).

[58] J. Cukras, S. Coriani, P. Decleva, O. Christiansen, and P. Norman, J. Chem. Phys. **139**, 094103 (2013).

[59] B. N. C. Tenorio, T. Moitra, M. A. C. Nascimento, A. B. Rocha, and S. Coriani, J. Chem. Phys. **150**, 224104 (2019).

[60] T. Wolf, R. Myhre, J. Cryan, S. Coriani, R. Squibb, A. Battistoni, N. Berrah, C. Bostedt, P. Bucksbaum, G. Coslovich, R. Feifel, K. Gaffney, J. Grilj, T. Martinez, S. Miyabe, S. Moeller, M. Mucke, A. Natan, R. Obaid, T. Osipov, O. Plekan, S. Wang, H. Koch, and M. Gühr, Nat. Commun. **8**, 29 (2017).

[61] R. Faber, E. F. Kjønstad, H. Koch, and S. Coriani, J. Chem. Phys. **151**, 144107 (2019).

[62] R. Faber and S. Coriani, J. Chem. Theory Comput. **15**, 520 (2019).

[63] M. L. Vidal, X. Feng, E. Epifanovsky, A. I. Krylov, and S. Coriani, J. Chem. Theory Comput. **15**, 3117 (2019).

[64] J. J. Eriksen, L. M. Solanko, L. J. Nåbo, D. Wüstner, S. P. A. Sauer, and J. Kongsted, Comput. Theory Chem. **1040-1041**, 54 (2014).

[65] J. M. Olsen, K. Aidas, and J. Kongsted, J. Chem. Theory Comput. **6**, 3721 (2010).

[66] J. M. H. Olsen and J. Kongsted, in Advances in Quantum Chemistry (Elsevier, 2011), Vol. 61, pp. 107–143.

[67] J. M. H. Olsen, N. H. List, C. Steinmann, A. H. Steindal, M. S. Nørby, and P. Reinholdt, PElib: The Polarizable Embedding library, 2020, see https://gitlab.com/pe-software/pelib-public.

[68] K. Sneskov, T. Schwabe, J. Kongsted, and O. Christiansen, J. Chem. Phys. **134**, 104108 (2011).

[69] E. D. Hedegård, N. H. List, H. J. Aa. Jensen, and J. Kongsted, J. Chem. Phys. **139**, 044101 (2013).

[70] E. D. Hedegård, J. M. H. Olsen, S. Knecht, J. Kongsted, and H. J. Aa. Jensen, J. Chem. Phys. **142**, 114113 (2015).

[71] J. J. Eriksen, S. P. A. Sauer, K. V. Mikkelsen, H. J. Aa. Jensen, and J. Kongsted, J. Comput. Chem. **33**, 2012 (2012).

[72] M. N. Pedersen, E. D. Hedegård, J. M. H. Olsen, J. Kauczor, P. Norman, and J. Kongsted, J. Chem. Theory Comput. **10**, 1164 (2014).

[73] C. Steinmann, J. M. H. Olsen, and J. Kongsted, J. Chem. Theory Comput. **10**, 981 (2014).

[74] N. H. List, M. T. P. Beerepoot, J. M. H. Olsen, B. Gao, K. Ruud, H. J. Aa. Jensen, and J. Kongsted, J. Chem. Phys. **142**, 034119 (2015).

[75] N. H. List, H. J. Aa. Jensen, and J. Kongsted, Phys. Chem. Chem. Phys. **18**, 10070 (2016).

[76] N. H. List, P. Norman, J. Kongsted, and H. J. Aa. Jensen, J. Chem. Phys. **146**, 234101 (2017).

[77] C. Steinmann, QFITLIB: A library to do multipole fitting in quantum chemistry codes, 2020, see https://github.com/cstein/qfitlib.

[78] C. Steinmann and J. Kongsted, J. Chem. Theory Comput. **11**, 4283 (2015).

[79] N. M. Thellamurege and H. Li, J. Chem. Phys. **137**, 246101 (2012).

[80] M. S. Nørby, C. Steinmann, J. M. H. Olsen, H. Li, and J. Kongsted, J. Chem. Theory Comput. **12**, 5050 (2016).

[81] N. H. List, J. M. H. Olsen, and J. Kongsted, Phys. Chem. Chem. Phys. **18**, 20234 (2016).

[82] C. Steinmann, P. Reinholdt, M. S. Nørby, J. Kongsted, and J. M. H. Olsen, Int. J. Quantum Chem. **119**, e25717 (2019).

[83] L. J. Nåbo, J. M. H. Olsen, N. H. List, L. M. Solanko, D. Wüstner, and J. Kongsted, J. Chem. Phys. **145**, 104102 (2016).

[84] D. Hršak, J. M. H. Olsen, and J. Kongsted, J. Chem. Theory Comput. **14**, 1351 (2018).

[85] J. M. H. Olsen, C. Steinmann, K. Ruud, and J. Kongsted, J. Phys. Chem. A **119**, 5344 (2015).

[86] P. Reinholdt, J. Kongsted, and J. M. H. Olsen, J. Phys. Chem. Lett. **8**, 5949 (2017).

[87] S. Huzinaga and A. A. Cantu, J. Chem. Phys. **55**, 5543 (1971).

[88] A. S. P. Gomes and C. R. Jacob, Annu. Rep. Prog. Chem., Sect. C: Phys. Chem. **108**, 222 (2012).

[89] C. R. Jacob and J. Neugebauer, Wiley Interdiscip. Rev.: Comput. Mol. Sci. **4**, 325 (2014).

[90] A. S. P. Gomes, C. R. Jacob, and L. Visscher, Phys. Chem. Chem. Phys. **10**, 5353 (2008).

[91] C. R. Jacob, S. M. Beyhan, R. E. Bulo, A. S. P. Gomes, A. W. Götz, K. Kiewisch, J. Sikkema, and L. Visscher, J. Comput. Chem. **32**, 2328 (2011).

[92] S. Höfener, A. S. P. Gomes, and L. Visscher, J. Chem. Phys. **136**, 044104 (2012).

[93] S. Höfener, A. S. P. Gomes, and L. Visscher, J. Chem. Phys. **139**, 104106 (2013).

[94] M. Guidon, J. Hutter, and J. VandeVondele, J. Chem. Theory Comput. **6**, 2348 (2010).

[95] U. Ekström and contributors, XCFun: A library of exchange-correlation functionals with arbitrary-order derivatives, 2020, see https://xcfun.readthedocs.io/.

[96] U. Ekström, L. Visscher, R. Bast, A. J. Thorvaldsen, and K. Ruud, J. Chem. Theory Comput. **6**, 1971 (2010), available at http://jmlr.org/papers/v18/16-107.html.

[97] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, and D. M. Blei, J. Mach. Learn. Res. **18**, 1 (2017), available at http://jmlr.org/papers/v18/16-107.html.

[98] S. Reine, E. Tellgren, and T. Helgaker, Phys. Chem. Chem. Phys. **9**, 4771 (2007).

[99] P. Merlot, R. Izsák, A. Borgoo, T. Kjærgaard, T. Helgaker, and S. Reine, J. Chem. Phys. **141**, 094104 (2014).

[100] C. Kumar, H. Fliegl, F. J. Jensen, A. M. Teale, S. Reine, and T. Kjaergaard, Int. J. Quantum Chem. **118**, e25639 (2018).

[101] M. Ziółkowski, B. Jansík, T. Kjærgaard, and P. Jørgensen, J. Chem. Phys. **133**, 014107 (2010).

[102] K. Kristensen, P. Jørgensen, B. Jansík, T. Kjærgaard, and S. Reine, J. Chem. Phys. **137**, 114102 (2012).

[103] K. Kristensen, I.-M. Høyvik, B. Jansik, P. Jørgensen, T. Kjærgaard, S. Reine, and J. Jakowski, Phys. Chem. Chem. Phys. **14**, 15706 (2012).

[104] B. Jansík, S. Høst, K. Kristensen, and P. Jørgensen, J. Chem. Phys. **134**, 194104 (2011).

[105] I.-M. Høyvik, B. Jansík, and P. Jørgensen, J. Chem. Theory Comput. **8**, 3137 (2012).

[106] P. Baudin, P. Ettenhuber, S. Reine, K. Kristensen, and T. Kjærgaard, J. Chem. Phys. **144**, 054102 (2016).

[107] D. Bykov, K. Kristensen, and T. Kjærgaard, J. Chem. Phys. **145**, 024106 (2016).

[108] D. Bykov and T. Kjærgaard, J. Comput. Chem. **38**, 228 (2017).

[109] T. Kjærgaard, J. Chem. Phys. **146**, 044103 (2017).

[110] J. J. Eriksen, P. Baudin, P. Ettenhuber, K. Kristensen, T. Kjærgaard, and P. Jørgensen, J. Chem. Theory Comput. **11**, 2984 (2015).

[111] A. L. Barnes, D. Bykov, D. I. Lyakh, and T. P. Straatsma, J. Phys. Chem. A **123**, 8734 (2019).

[112] P. Baudin and K. Kristensen, J. Chem. Phys. **144**, 224106 (2016).

[113] P. Baudin, D. Bykov, D. Liakh, P. Ettenhuber, and K. Kristensen, Mol. Phys. **115**, 2135 (2017).

[114] P. Baudin, T. Kjærgaard, and K. Kristensen, J. Chem. Phys. **146**, 144107 (2017).

[115] P. Baudin and K. Kristensen, J. Chem. Phys. **146**, 214114 (2017).

[116] V. V. Rybkin, U. Ekström, and T. Helgaker, J. Comput. Chem. **34**, 1842 (2013).

[117] K. Kristensen, P. Ettenhuber, J. J. Eriksen, F. Jensen, and P. Jørgensen, J. Chem. Phys. **142**, 114116 (2015).

[118] T. H. Rasmussen, Y. M. Wang, T. Kjærgaard, and K. Kristensen, J. Chem. Phys. **144**, 204102 (2016).

[119] C. Kumar, T. Kjærgaard, T. Helgaker, and H. Fliegl, J. Chem. Phys. **145**, 234108 (2016).

[120] R. Bast, D. H. Friese, B. Gao, D. J. Jonsson, S. S. Reine, M. Ringholm, and K. Ruud, OpenRSP: An open-ended response property library, 2020, see https://openrsp.org/.

[121] R. Bast and contributors, XCint: Exchange–correlation integrator, 2020, see https://xcint.readthedocs.io/.

[122] R. Di Remigio, A. H. Steindal, K. Mozgawa, V. Weijo, H. Cao, and L. Frediani, Int. J. Quantum Chem. **119**, e25685 (2019).

[123] D. W. Zhang and J. Z. H. Zhang, J. Chem. Phys. **119**, 3599 (2003).

[124] O. Vahtras, LoProp for Dalton, 2014, see https://github.com/vahtras/loprop.

[125] L. Gagliardi, R. Lindh, and G. Karlström, J. Chem. Phys. **121**, 4494 (2004).

[126] B. Gao, Gen1Int: An open-ended integral library, 2012, see https://gitlab.com/bingao/gen1int.

[127] B. Gao, A. J. Thorvaldsen, and K. Ruud, Int. J. Quantum Chem. **111**, 858 (2010).

[128] B. Gao, QcMatrix: An abstract matrix library, 2015, see https://gitlab.com/bingao/qcmatrix.

[129] B. P. Pritchard, D. Altarawy, B. Didier, T. D. Gibson, and T. L. Windus, J. Chem. Inf. Model. **59**, 4814 (2019).

[130] J. D. Hunter, Comput. Sci. Eng. **9**, 90 (2007).

[131] S. Reine, A. Krapp, M. F. Iozzi, V. Bakken, T. Helgaker, F. Pawłowski, and P. Sałek, J. Chem. Phys. **133**, 044102 (2010).

[132] C.-M. Suomivuori, N. O. C. Winter, C. Hättig, D. Sundholm, and V. R. I. Kaila, J. Chem. Theory Comput. **12**, 2644 (2016).

[133] M. Hubert, E. D. Hedegård, and H. J. Aa. Jensen, J. Chem. Theory Comput. **12**, 2203 (2016).

[134] M. Hubert, H. J. Aa. Jensen, and E. D. Hedegård, J. Phys. Chem. A **120**, 36 (2016).

[135] I. B. Nielsen, L. Lammich, and L. H. Andersen, Phys. Rev. Lett. **96**, 018304 (2006).

[136] L. H. Andersen, I. B. Nielsen, M. B. Kristensen, M. O. A. El Ghazaly, S. Haacke, M. B. Nielsen, and M. Å. Petersen, J. Am. Chem. Soc. **127**, 12347 (2005).

[137] H. J. Aa. Jensen, P. Jørgensen, H. Ågren, and J. Olsen, J. Chem. Phys. **88**, 3834 (1988).

[138] A. Cembran, R. González-Luque, P. Altoè, M. Merchán, F. Bernardi, M. Olivucci, and M. Garavelli, J. Phys. Chem. A **109**, 6597 (2005).

[139] G. Dong, U. Ryde, H. J. Aa. Jensen, and E. D. Hedegård, Phys. Chem. Chem. Phys. **20**, 794 (2018).

[140] M. Scheurer, P. Reinholdt, E. R. Kjellgren, J. M. H. Olsen, A. Dreuw, and J. Kongsted, J. Chem. Theory Comput. **15**, 6154 (2019).

[141] M. T. P. Beerepoot, D. H. Friese, N. H. List, J. Kongsted, and K. Ruud, Phys. Chem. Chem. Phys. **17**, 19306 (2015).

[142] K. O. H. Dundas, M. Ringholm, Y. Cornaton, and B. Ofstad, SpectroscPy: Python tools for spectroscopy, 2020, see https://gitlab.com/kdu002/SpectroscPy.

[143] A. Thorvaldsen, K. Ruud, K. Kristensen, P. Jørgensen, and S. Coriani, J. Chem. Phys. **129**, 214108 (2008).

[144] M. Ringholm, D. Jonsson, and K. Ruud, J. Comput. Chem. **35**, 622 (2014).

[145] S. Coriani, S. Høst, B. Jansík, L. Thøgersen, J. Olsen, P. Jørgensen, S. Reine, F. Pawłowski, T. Helgaker, and P. Sałek, J. Chem. Phys. **126**, 154108 (2007).

[146] J. Bloino and V. Barone, J. Chem. Phys. **136**, 124108 (2012).

[147] E. B. Wilson, J. C. Decius, and P. C. Cross, *Molecular Vibrations: The Theory of Infrared and Raman Vibrational Spectra* (Courier Corporation, 1980).

[148] T. Helgaker, Acta Chem. Scand. **42a**, 515 (1988).

[149] Z. Rinkevicius, I. Tunell, P. Sałek, O. Vahtras, and H. Ågren, J. Chem. Phys. **119**, 34 (2003).

[150] Z. Rinkevicius, P. C. Jha, C. I. Oprea, O. Vahtras, and H. Ågren, J. Chem. Phys. **127**, 114101 (2007).

[151] Z. Rinkevicius, L. Telyatnyk, P. Sałek, O. Vahtras, and H. Ågren, J. Chem. Phys. **119**, 10489 (2003).

[152] Z. Rinkevicius, L. Telyatnyk, O. Vahtras, and H. Ågren, J. Chem. Phys. **121**, 7614 (2004).

[153] X. Chen, Z. Rinkevicius, Z. Cao, K. Ruud, and H. Ågren, Phys. Chem. Chem. Phys. **13**, 696 (2011).

[154] Z. Rinkevicius, K. J. de Almeida, and O. Vahtras, J. Chem. Phys. **129**, 064109 (2008).

[155] C. I. Oprea, L. Telyatnyk, Z. Rinkevicius, O. Vahtras, and H. Ågren, J. Chem. Phys. **124**, 174103 (2006).

[156] D. Duflot, J.-P. Flament, I. C. Walker, J. Heinesch, and M.-J. Hubin-Franskin, J. Chem. Phys. **118**, 1137 (2003).