CPPE: An Open-Source C++ and Python Library for Polarizable Embedding

Maximilian Scheurer,[*,†,‡] Peter Reinholdt,[‡] Erik Rosendahl Kjellgren,[‡] Jógvan Magnus Haugaard Olsen,[¶] Andreas Dreuw,[†] and Jacob Kongsted[*,‡]

†*Interdisciplinary Center for Scientific Computing, Heidelberg University, D-69120 Heidelberg, Germany*

‡*Department of Physics, Chemistry and Pharmacy, University of Southern Denmark, DK-5230 Odense M, Denmark*

¶*Hylleraas Centre for Quantum Molecular Sciences, Department of Chemistry, UiT The Arctic University of Norway, N-9037 Tromsø, Norway*

E-mail: maximilian.scheurer@iwr.uni-heidelberg.de; kongsted@sdu.dk

Abstract

We present a modular open-source library for polarizable embedding (PE) named Cppe. The library is implemented in C++, and it additionally provides a Python interface for rapid prototyping and experimentation in a high-level scripting language. Our library integrates seamlessly with existing quantum chemical program packages through an intuitive and minimal interface. Until now, Cppe has been interfaced to three packages, Q-Chem, Psi4, and PySCF. Furthermore, we show Cppe in action using all three program packages for a computational spectroscopy application. With Cppe, host program interfaces only require minor programming effort, paving the way for new combined methodologies and broader availability of the PE model.

1 Introduction

Accurate computational spectroscopy requires a realistic description of the key interactions of the chromophore with the molecular environment.[1,2] To model such interactions of solute-solvent systems at a reasonable computational cost, embedding methods for continuum solvation and explicit environments exist.[3–5] A quantum mechanical method of choice is combined with one or multiple environment models to simulate molecular properties, such as electronic excitation energies. The procedure of including environmental effects can, to some extent, be uncoupled from the quantum mechanical method. Hence, implementations of a single embedding method can be modularized and interfaced to multiple quantum chemical program packages. Exploiting modular libraries puts the main focus back on the development of quantum mechanical methods: Using a well-tested, production-ready library for the inclusion of environment contributions is much more sustainable and time-efficient than re-implementations for each program package and method. In this paper, we show the implementation of such an open-source modular library for the polarizable embedding (PE) model, named Cppe (C++ and Python library for PE). The PE model[6,7] has, over the last years, emerged as a powerful and user-friendly approach to study molecular properties in complex embedded systems.[8,9] A large variety of combinations with wave function and density functional theory methods has been developed,[10–19] where the most used implementation is publicly available in PElib[20] which is interfaced to the Dalton[21] and DIRAC[22] programs. Furthermore, a closed-source implementation in the Turbomole package exists,[12,23] especially employed for correlated wave function methods.

Our novel Cppe library contains the necessary routines to implement ground state and molecular property calculations with PE. Through its minimalist application programming interface (API), Cppe can be easily coupled to any quantum mechanical host program with little programming effort. The library is inspired by PCMSolver[24] and libefp,[25] which are similar libraries for continuum solvation models and the effective fragment potential (EFP) method, respectively. The continuum and EFP models, as well as the PE model presented

in this paper, all belong to the class of so-called explicitly polarizable effective Hamiltonian models. In fact, due to the similar mathematical structure of these models, modularization may be devised based on similar strategies. Indeed, this is also the case for other polarizable embedding models like the fluctuating charges[26,27] or Drude oscillator[28] approaches which have, to the best of our knowledge, not yet been formulated employing a modular library. Of note, a general linear-scaling implementation for PE models was recently presented.[29]

Cppe provides both a C++ and a Python API, exposing the necessary high-level functionality. The Python API allows for quick manipulation of data and rapid prototyping to try out new variants or combinations of the PE model. On the low level, Cppe is based on the original implementation of the PE model, PElib.[20] Cppe is designed to be as modular as possible, such that it can be interfaced to *any* program without any spill-over of host-program-specific code into Cppe. In the long run, Cppe will make calculations using PE more accessible to a broad user base through several program packages and allow for novel combined methodologies in computational spectroscopy. Of note, we have already interfaced Cppe to three existing quantum chemical program packages Q-Chem,[30] Psi4,[31] and PySCF.[32] The remainder of this paper is structured as follows: First, we briefly review the theoretical background of the PE model. Second, we introduce the design of the library, together with its implementation and capabilities. We also show a step-by-step guide on how to connect Cppe to a new host program. Finally, we investigate the solvatochromism of nile red using existing Cppe interfaces: The absorption spectrum of the chromophore in an aqueous and a protein environment is modeled using the algebraic-diagrammatic construction (ADC) method, equation-of-motion coupled cluster (EOM-CC), and time-dependent density functional theory (TDDFT) in combination with PE.

2 Theoretical Background

The theoretical foundation of the PE model has been presented in previous works.[6,7] Furthermore, there exists an extensive tutorial review[9] on the necessary steps to perform PE calculations for spectroscopic processes in chemical or biological systems. The composite PE energy functional[6,7] can be written as

$$E_{\text{tot}} = E_{\text{QM}} + E_{\text{PE}} + E_{\text{env}} \ , \tag{1}$$

with the energy of the quantum region E_{QM} which includes wave function polarization. The interaction energy of the quantum region and the environment E_{PE} includes polarization of the environment. The internal energy of all fragments in the environment E_{env} is completely independent of the wave function since polarization is already contained in the previous term. Further decomposition of E_{PE} leads to contributions from the electrostatic interaction energy E_{es} between the quantum region and the environment, and the induction energy E_{ind} due to induced charge distributions in the environment. The permanent electrostatic interaction energy consists of a nuclear contribution $E_{\text{es}}^{\text{nuc}}$ and an electronic contribution $E_{\text{es}}^{\text{el}}$, where the nuclear contribution is given by

$$E_{\text{es}}^{\text{nuc}} = \sum_{s=1}^{S} \sum_{|k|=0}^{K_s} \frac{(-1)^{|k|}}{k!} Q_s^{(k)} \sum_{n=1}^{N} T_{sn}^{(k)} Z_n \ . \tag{2}$$

The summation over $|k|$ runs over $(|k|+2)(|k|+1)/2$ multi-indices up to the truncation level K_s of the multipole expansion and the summation over s is running over the S sites in the environment. The k-th component of the $|k|$-th-order Cartesian multipole $Q_s^{(k)}$ is located at the site coordinate $\underline{\mathbf{R}}_s$ in the environment, and Z_n is the nuclear charge of the n-th nucleus located at $\underline{\mathbf{R}}_n$ in the quantum region, comprised of N nuclei in total. Here, the

k-th component of the interaction tensor, $T_{ij}^{(k)}$, between two sites i and j,

$$T_{ij}^{(k)} = \frac{\partial^{|k|}}{\partial x_j^{k_x} \partial y_j^{k_y} \partial z_j^{k_z}} \left(\frac{1}{|\underline{\mathbf{r}}_j - \underline{\mathbf{r}}_i|} \right) \ , \tag{3}$$

was used. Further, the electrostatic interaction energy of the electrons with the environment is given by the expectation value of the electrostatic operator $\hat{\mathcal{V}}_{\text{es}}$ using the electronic density matrix of the quantum region $\mathbf{P}_{\text{el}}$, that is

$$E_{\text{es}}^{\text{el}} = \text{Tr}(\mathbf{P}_{\text{el}} \hat{\mathcal{V}}_{\text{es}}) \ . \tag{4}$$

Using the second-quantization formalism, we can write the electrostatic operator as

$$\hat{\mathcal{V}}_{\text{es}} = \sum_{s=1}^{S} \sum_{|k|=0}^{K_s} \frac{(-1)^{|k|}}{k!} Q_s^{(k)} \sum_{pq} t_{pq}^{(k)}(\underline{\mathbf{R}}_s) \hat{E}_{pq} \ , \tag{5}$$

with the one-electron orbital excitation operator $\hat{E}_{pq}$ and general molecular orbital indices p and q. The integrals are given by

$$t_{pq}^{(k)}(\underline{\mathbf{R}}_s) = -\int \phi_p^*(\underline{\mathbf{r}}_1) T_{s1}^{(k)} \phi_q(\underline{\mathbf{r}}_1) \, \mathrm{d}\underline{\mathbf{r}}_1 \ , \tag{6}$$

and include again the k-th component of the interaction tensor (eq (3)). The induction energy contribution of a linearly responsive environment amounts to

$$E_{\text{ind}} = -\frac{1}{2} \sum_{s=1}^{S} \underline{\boldsymbol{\mu}}_s^{\text{ind}}(\underline{\mathbf{F}})^{\text{T}} \underline{\mathbf{F}}(\underline{\mathbf{R}}_s) \ , \tag{7}$$

where $\underline{\boldsymbol{\mu}}_s^{\text{ind}}(\underline{\mathbf{F}})$ is the induced dipole moment at site s in the environment, and $\underline{\mathbf{F}}(\underline{\mathbf{R}}_s)$ is the electric field vector acting on site s, comprising the field from nuclei and electrons, as well

as the fields caused by the permanent multipole moments, i.e.,

$$\underline{\mathbf{F}}[\mathbf{P}_{\mathrm{el}}] = \underline{\mathbf{F}}_{\mathrm{nuc}} + \underline{\mathbf{F}}_{\mathrm{el}}[\mathbf{P}_{\mathrm{el}}] + \underline{\mathbf{F}}_{\mathrm{mul}} \ . \tag{8}$$

Note that the electric field from the electrons, and in turn the total field vector, $\underline{\mathbf{F}}$, *explicitly* depend on the electronic density matrix $\mathbf{P}_{\mathrm{el}}$. The electric field created by the electrons is a simple expectation value of the field operator. Further, we define the electric-field operator as

$$\hat{F}^e_a(\underline{\mathbf{R}}_s) = \sum_{pq} t_{a,pq}(\underline{\mathbf{R}}_s)\hat{E}_{pq} \ . \tag{9}$$

The electric-field integrals are defined as

$$t_{a,pq}(\underline{\mathbf{R}}_s) = -\int \phi^*_p(\underline{\mathbf{r}}) \frac{R_{a,s} - r_a}{|\underline{\mathbf{R}}_s - \underline{\mathbf{r}}|^3} \phi_q(\underline{\mathbf{r}}) \, \mathrm{d}\underline{\mathbf{r}} \ , \tag{10}$$

such that the expectation value is simply

$$\underline{\mathbf{F}}_{\mathrm{el}}[\mathbf{P}_{\mathrm{el}}](\underline{\mathbf{R}}_s) = \mathrm{Tr}(\mathbf{P}_{\mathrm{el}}\underline{\hat{\mathbf{F}}}^e(\underline{\mathbf{R}}_s)) \ . \tag{11}$$

The induced moment at a site s depends on the total electric field through

$$\underline{\boldsymbol{\mu}}^{\mathrm{ind}}_s(\underline{\mathbf{F}}) = \boldsymbol{\alpha}_s \left(\underline{\mathbf{F}}_s[\mathbf{P}_{\mathrm{el}}] + \underline{\mathbf{F}}^{\mathrm{ind}}_s \right) \ . \tag{12}$$

Here, the induced fields created by all other sites were added, i.e.,

$$\underline{\mathbf{F}}^{\mathrm{ind}}_s = \sum_{s' \neq s} \mathbf{T}^{(2)}_{ss'} \underline{\boldsymbol{\mu}}^{\mathrm{ind}}_{s'}(\underline{\mathbf{F}}) \ . \tag{13}$$

This leads to a linear system of equations,

$$\mathbf{B}\underline{\boldsymbol{\mu}}^{\mathrm{ind}}(\underline{\mathbf{F}}) = \underline{\mathbf{F}}[\mathbf{P}_{\mathrm{el}}] \ , \tag{14}$$

with the classical response matrix $\mathbf{B}$,[33] given by

$$\mathbf{B} = \begin{pmatrix} \boldsymbol{\alpha}_1^{-1} & -\mathbf{T}_{12}^{(2)} & \dots & -\mathbf{T}_{1S}^{(2)} \\ -\mathbf{T}_{21}^{(2)} & \boldsymbol{\alpha}_2^{-1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\mathbf{T}_{(S-1)S}^{(2)} \\ -\mathbf{T}_{S1}^{(2)} & \dots & -\mathbf{T}_{S(S-1)}^{(2)} & \boldsymbol{\alpha}_S^{-1} \end{pmatrix} \ . \tag{15}$$

The inverse site polarizability tensors $\boldsymbol{\alpha}_s^{-1}$ are on the diagonal and the negative dipole-dipole interaction tensors $\mathbf{T}^{(2)}$ reside on off-diagonal elements. Subsequently, we can include the induced dipole field into the wave function optimization through the induction operator

$$\hat{\mathcal{V}}_{\mathrm{ind}}[\mathbf{P}_{\mathrm{el}}] = -\sum_{s=1}^{S} \sum_{a=x,y,z} \mu_{a,s}^{\mathrm{ind}}(\underline{\mathbf{F}}) \hat{F}_a^e(\underline{\mathbf{R}}_s) \ , \tag{16}$$

using a for the respective Cartesian component x, y, or z. Finally, one solves the self-consistent field (SCF) problem in the presence of the total embedding operator,

$$\hat{\mathcal{V}}_{\mathrm{PE}}[\mathbf{P}_{\mathrm{el}}] = \hat{\mathcal{V}}_{\mathrm{es}} + \hat{\mathcal{V}}_{\mathrm{ind}}[\mathbf{P}_{\mathrm{el}}] \ . \tag{17}$$

Since the embedding operator depends on the wave function itself, namely through the electric fields created by the electronic density, the overall embedding operator is non-linear. The embedding operator is updated in every iteration using the current SCF electronic density matrix. Thus, polarization effects are treated in a self-consistent manner for the electronic ground state.

3 Design and Implementation

Implementation of the Cppe library was, on the low level, guided by the existing Fortran library, PElib.[20] We aimed for high modularity, host program agnosticy[24] and extensibility to design the library as sustainable as possible. Therefore, the Cppe library is implemented in C++ which provides the necessary toolkit for data containers and standard algorithms through the standard template library (STL) together with advantages of object-oriented programming. The latter make the implementation of the necessary data structures for our PE library intuitive and easily extensible. Cppe is built with CMake,[34] widely used in quantum chemical program packages and thus making Cppe easy to integrate in an existing build systems as an optional external dependency. To perform numerical operations on matrices and vectors, e.g., solving the linear equations for induced moments, we employ the header-only Eigen3 library.[35] To bring Cppe to modern Python-based quantum chemistry codes, we aimed at exposing the functionality of the library to the Python layer. This is accomplished through the lightweight header-only pybind11 library,[36] providing interoperability between C++-based codes and Python. The pybind11 interface code inside Cppe is very concise and allows for rapid extension of Python-exposed functionality. Furthermore, Eigen3 matrices and vectors are seamlessly converted to `numpy` arrays[37] and vice versa through pybind11. Python bindings also enabled us to implement a set of unit and functionality tests using pytest.[38] Thus, the suite of test cases can be easily extended on the Python layer. In general, the here presented hybrid C++/Python programming approach has also proven powerful in the recently published Psi4NumPy package.[39]

The most recent version of the Cppe source code can be downloaded from GitHub (`https://github.com/maxscheurer/cppe`). The C++ core library is contained in the `cppe/core` directory in the downloaded folder, whereas the Python bindings reside in `cppe/python_iface`. A table with the source code location of all C++ classes explained in the following can be found in Table S1 in the Supporting Information. Most importantly, the core library is equipped with data containers for embedding potentials, comprised of multipole moments

($\{Q_s^{(k)}\}$, `Multipole` class) and polarizabilities ($\{\boldsymbol{\alpha_s}\}$, `Polarizability` class). The main parameter container for each site in the environment is the `Potential` class, comprised of coordinates $\underline{\mathbf{R}}_s$ for a site s, together with a list of multipoles (`std::vector<Multipole>`), polarizabilities (`std::vector<Polarizability>`), and some helper functions. All together, the full environment parametrization is stored as a `std::vector<Potential>`. The three parameter container classes are depicted in Figure 1. Cppe reads the parameters mentioned

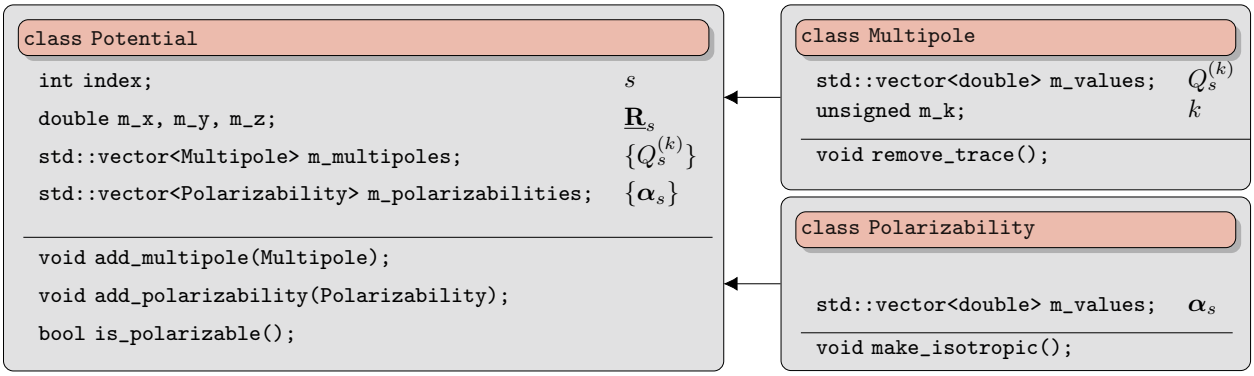


Figure 1: C++ classes containing the embedding potential. The `Multipole` and `Polarizability` classes contain the actual parameters and provide helper functions, e.g., to remove the trace of a multipole moment or to make a polarizability isotropic. A single site in the environment is fully parametrized through an instance of the `Potential` class, consisting of the site index s, the coordinates, and vectors of multipoles and polarizabilities. Helper functions make it easy to add additional parameters or to check if a specific site is polarizable.

above from a so-called potential file. The format is identical to that used in PElib, explained in Ref. 9. Read-in is performed by the `PotfileReader` class. If a special treatment of the border between the quantum and classical region is required, the `PotManipulator` class can, for example, redistribute or remove parameters of the affected sites. In addition to parameters, information about the quantum region has to be stored, for example, to evaluate the nuclear electrostatic interaction energy (eq (2)). This is achieved by the `Molecule` class, which is a slightly decorated `std::vector` with coordinates $\underline{\mathbf{R}}_n$ and charges Z_n of the individual atoms.

The core library also provides classes to compute all classical energies and electric fields: `MultipoleExpansion` computes the nuclei-multipole interaction energy (eq (2)), whereas

`NuclearFields` and `MultipoleFields` implement electric field contributions from nuclei and multipole moments, respectively (eq (8)). Furthermore, the system of linear equations for the induced dipole moments is solved iteratively with a Jacobi algorithm[9] including Anderson mixing (sometimes called DIIS mixing) for accelerated convergence.[40] As can be seen from the theory section, the most important building block of the classical expressions are the T-tensors (eq (3)) which we compute using an open-ended formula.[9,41] To avoid over-polarization, interactions involving T-tensors, i.e., permanent multipole fields ($\underline{\mathbf{F}}_{\text{mul}}$ in eq (8)), or dipole-dipole interaction tensors ($\mathbf{T}^{(2)}$ in eq (15)) can be damped using Thole's exponential scheme.[42,43] Since the implementation of the T-tensors is pivotal, it is tested against auto-generated Python code.[44]

User-provided options, e.g., the path of the potential file, convergence thresholds, or treatment of the border between the quantum and classical region, are defined in the `PeOptions` class. All available options and their default values are listed in Table S2. Of note, the aforementioned low-level building blocks and functions do not need to be assembled from scratch when interfacing Cppe to a new host program, which would be tedious and error-prone. For this reason, Cppe provides a convenient top-level wrapper of all low-level functions exposed through the `CppeState` class (Fig. 2). Using the `CppeState` to manage all necessary PE tasks reduces the programming effort because all implemented functions and data fields of `CppeState` are self-explanatory and correspond to the formulas previously given. The `CppeState` can be constructed from a `Molecule` object and a `PeOptions` object. After instantiation, the potential file is automatically parsed, manipulated, and stored inside the `CppeState`. The `CppeState` is then ready for use, taking care of, e.g., setting up and calling the induced moments solver under the hood.

Having established the components of Cppe in a bottom-up manner, we will now explain how to actually use `CppeState` for the implementation of a host program interface.

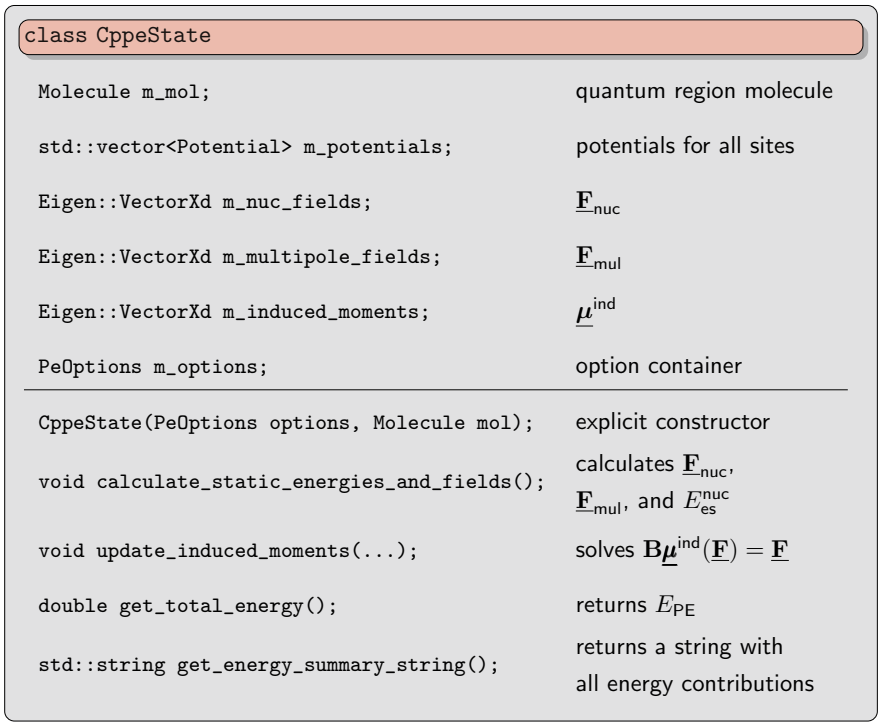


Figure 2: `CppeState` class members. The `CppeState` serves as the top-level interface of the Cppe library. It exposes a variety of functions to carry out all host-program-independent tasks. All the building blocks, e.g., to solve the linear equations for induced dipole moments, are properly assembled in the `CppeState` functions. Further, `CppeState` manages bookkeeping of energy contributions and electric fields.

3.1 Guide to Using Cppe in a Host Program

Cppe is completely agnostic of any host-program-specific code and data. As a result, only an interface on the host program side needs to be implemented, integrating both Cppe and program-specific routines. A schematic overview of the overall interface structure is shown in Figure 3. The host program side of the interface communicates with the input reader, integral library, SCF driver, and post-SCF drivers of the host program. A mock implementation of such a `CppeHostProgramInterface` is presented using Python code snippets. First of all, the constructor of the class (Listing 1) takes a `Molecule` and `PeOptions` object to build the initial `CppeState`. Furthermore, the function constructs a `numpy` array with the coordinates of all polarizable sites for downstream computation of field integrals. Finally, the static contributions to the electrostatic interaction energy (eq (2)) and electric fields are calculated.

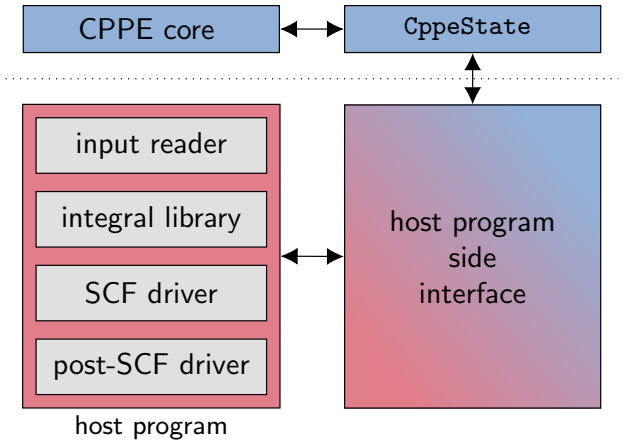


Figure 3: Schematic overview of the interface structure. The host program side of the interface to Cppe requires access to the input reader, integral library, SCF driver, and post-SCF driver of the host program. The host-program-independent tasks are taken care of on the Cppe side, wrapped by a state object. Since the interface to include the Cppe state is minimal, the major programming work is in gathering the required data and integrals from the host program that one is already familiar with.

The key ingredient of the host program interface is to expose a routine to compute the PE

Listing 1: Mock constructor of `CppeHostProgramInterface`

```python
import numpy as np
import integral_library  # host program integral library
from cppe import CppeState

class CppeHostProgramInterface:
    def __init__(self, molecule, options):
        self.cppe_state = CppeState(molecule, options)
        # coordinates of polarizable sites
        self.polarizable_coords = np.array([
            site.position for site in self.cppe_state.potentials
            if site.is_polarizable
        ])
        self.cppe_state.calculate_static_energies_and_fields()
```

operator and energy from an input density matrix. Such a density-driven function can be employed both in the SCF driver and a post-SCF driver. An illustrative implementation of the PE contribution routine is displayed in Listing 2. The PE contribution routine first needs to compute the electrostatics operator (eq (5), step I), making use of the host program

Listing 2: Mock PE contribution routine of the `CppeHostProgramInterface` class

```
def get_pe_contribution(self, density_matrix, elec_only=False):
    # step I: build electrostatics operator
    if not self.V_es and not elec_only:
        self.build_electrostatics_operator()
    e_electrostatic = np.sum(density_matrix * self.V_es)
    self.cppe_state.energies["Electrostatic"]["Electronic"] = e_electrostatic

    # step II: obtain expectation values of elec. field at polarizable sites
    elec_fields = integral_library.electric_field_value(
        self.polarizable_coords, density_matrix
    )
    # step III: solve induced moments
    self.cppe_state.update_induced_moments(elec_fields)
    induced_moments = self.cppe_state.induced_moments

    # step IV: build induction operator
    V_ind = np.zeros_like(self.V_es)
    for coord, ind_mom in zip(self.polarizable_coords, induced_moments):
        field_int = integral_library.electric_field_integral(site=coord)
        V_ind += -1.0 * sum(ind_mom[i] * field_int[i] for i in range(3))
    E_pe = self.cppe_state.energies.total_energy
    V_pe = self.V_es + V_ind
    # only take electronic contributions into account
    if elec_only:
        V_pe = V_ind
        E_pe = self.cppe_state.energies["Polarization"]["Electronic"]
    return E_pe, V_pe
```

integral library together with the multipole moments stored in the `CppeState`. A sketch of this function inside the `CppeHostProgramInterface` class is displayed in Listing 3.

The required integrals (eqs (6) and (10)) must be available in the host program. For practical applications, however, it is often sufficient to have potential derivative integrals through second order, i.e., normal Coulomb integrals (used for the nuclear attraction operator), electric field integrals, and electric field gradient integrals. With these features, it is possible to model electrostatic interactions up to quadrupole moments and to employ self-consistent treatment of dipole polarization.

Once computed, the electrostatics operator can be stored in memory since it is density-

independent. After storing the operator, the electronic contribution to the electrostatic interaction energy (eq (4)) is obtained as the product-trace with the density matrix (l. 6 in Listing 2). Second, the PE routine must obtain the expectation values of the electric

Listing 3: Mock code snippet for construction of the electrostatics operator.

```
def build_electrostatics_operator(self):
    n_bas = integral_library.n_bas  # number of basis functions
    self.V_es = np.zeros((n_bas, n_bas))  # empty numpy array for operator matrix
    for site in self.cppe_state.potentials:
        for multipole in site.multipoles:
            self.V_es += integral_library.potential_derivative(
                position=site.position, order=multipole.k,
                moments=multipole.values
            )
```

field operator from the input density matrix (step II) to obtain the total electric field at all polarizable sites (eq (8)). Third, a simple call to the `CppeState` is made, requesting induced dipole moments from the given electric fields (step III). In the background, the system of linear equations (eq (14)) is solved, the induction energies are updated (eq (7)), and the resulting induced dipole moments are returned as a `numpy` array. In the fourth step, the induction operator (eq (16)) is formed by contracting the electric field integrals with the induced dipole moments (step IV). If the flag `elec_only` is set to `True`, only electronic contributions to the energy and PE operator are taken into account, as it is required for post-SCF procedures. Otherwise, the full operator (eq (17)) is assembled and returned, together with the PE energy contribution (l. 21-27 in Listing 2). Due to the simple structure of the host program interface, only a single routine needs to be called from all places in the program where PE contributions are required, as illustrated in Figure 4. For example, the SCF driver of the host program will call `get_pe_contribution` every iteration providing the current SCF density matrix, and in turn receives the PE contribution to the Fock operator without further ado. This simplistic and clean design makes it easy to implement PE contributions in various places in the host program with a single function call. Furthermore, it makes the interface code easily maintainable.

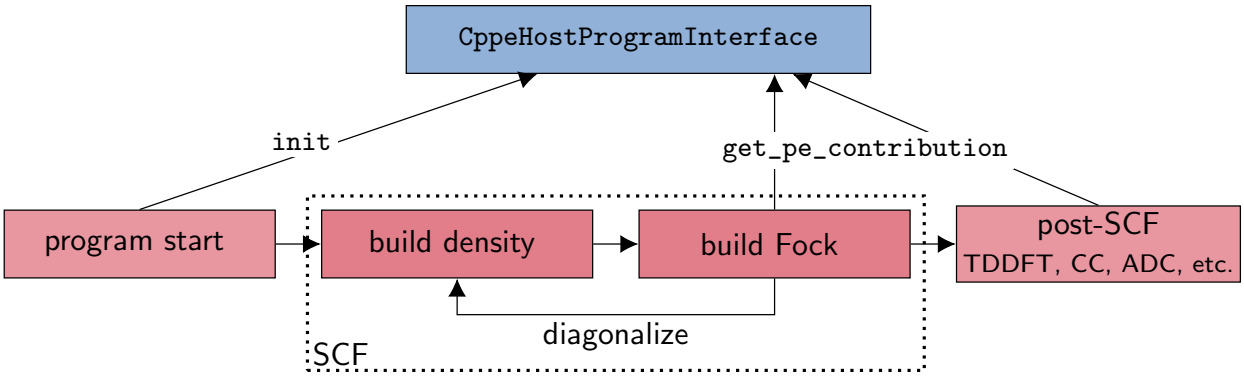


Figure 4: Overall PE program flow employing the `CppeHostProgramInterface`. After the host program (red boxes) starts, the host program interface is instantiated. In downstream routines, e.g., the SCF procedure or post-SCF methods, only the `get_pe_contribution` function needs to be called with an input density matrix to implement PE contributions where needed.

3.2 Existing Interfaces

The Cppe library is already interfaced to three program packages, Q-Chem, Psi4, and PySCF. All three host program interfaces do not exceed 200 lines of code in total and could be implemented with minimal time effort according to the guide above. In Q-Chem (Version 5.2),[30] Cppe is interfaced through C++ and enables ground state PE-SCF calculations together with the simulation of excited states using the PE-ADC method.[19] In this context, Cppe was used in our previous work to model excited states with PE-ADC in a large biomolecular environment.[19] The two open-source packages, Psi4 and PySCF, were connected to Cppe via Python. In Psi4, we also optimized existing integral code to make PE calculations more efficient, showing the benefits of modular and open-source software development.

4 Results

To showcase the features of our new library in all three program packages, Q-Chem, Psi4, and PySCF, we investigated the absorption spectrum of nile red both in water and protein environment using three different methodologies, ADC, EOM-CC, and TDDFT. In the

following, we briefly outline the ease of implementation of these combined methods before presenting the results of our case study.

4.1 Linear Response TDDFT with PySCF and Cppe

Listing 4: PE-TDA contributions in PySCF

```
# compute the PE contribution of the i-th state in AO basis from the current
↪ transition density
e_pe, v_pe_ao = mf._pol_embed.get_pe_contribution(dmov[i], elec_only=True)
# transform the PE contribution to molecular orbital basis
v_pe_ov = lib.einsum('pq,pi,qj->ij', v_pe_ao, orbo.conj(), orbv)
# add PE contribution to the matrix-vector product
v1ov[i] += v_pe_ov
```

Solving the time-dependent Kohn-Sham eigenvalue problem using an iterative Davidson solver requires computation of matrix-vector products of the orbital Hessian matrices with response vectors (also called transition densities).[45] In the case of linear response, one can construct the PE operator using the current iteration's response vector as input density to solve for the induced moments. Subsequently, the induction operator is formed as in eq (16) and added to the matrix-vector product.[46] If the Tamm-Dancoff approximation[47,48] (TDA) is employed, only a single block of the orbital Hessian is taken into account.[49] Having the necessary routines in PySCF for ground state PE-SCF calculations implemented, just three lines of code in the matrix-vector product routine for TDA had to be added. The code fragment is displayed in Listing 4. It involves calling the host program side of the Cppe interface with the current transition density `dmov[i]` and specifying that only electronic contributions should be taken into account. The returned operator is transformed to the molecular orbital basis and added to the matrix-vector product `v1ov[i]`.

4.2 Correlated Wave Function Methods in Combination with CPPE

In post-SCF methods where full treatment of PE response is more involved, excited states can be computed from the self-consistent PE-SCF ground state only, i.e., keeping the induced moments frozen during the post-SCF procedure. This approximation, however, entails erroneous excitation energies which can be corrected by means of perturbation theory. In the previously mentioned PE-ADC method,[19] the perturbative corrections are based on the transition density (perturbative linear-response-type correction, ptLR), and the difference density (perturbative state-specific correction, ptSS) of the respective excited state. The

Listing 5: Computation of ptSS corrections with PSI4NUMPY and CPPE

```python
def compute_ptss_corrections(ccwfn, nroots):
    ptss = []
    for i in range(1, nroots + 1):
        # obtain the CC density matrix
        ccdmat = ccwfn.variable("CC ROOT {} DA".format(i))
        # obtain the SCF density matrix
        scfdmat = ccwfn.Da()
        # compute the difference density
        ccdmat.subtract(scfdmat)
        ccdmat.scale(2.0)
        # compute the energy correction
        energy, v_pe = ccwfn.pe_state.get_pe_contribution(ccdmat.np,
        ↪  elec_only=True)
        ptss.append(energy)
    return ptss
```

PSI4 program package offers a set of coupled cluster approaches for excited states. Densities of the excited states are readily available. With our CPPE interface to PSI4 in place, we implemented ptSS corrections for EOM-CC excited states using PSI4NUMPY. The corresponding Python code snippet is shown in Listing 5. A converged coupled cluster wave function object `ccwfn`, together with the number of excited states `nroots` is passed to the Python function. Inside the loop, an energy correction is computed for each individual excited state. This implementation is analogous to the one for ADC inside the `adcman` library[50] in Q-Chem.

4.3 Nile Red in Water and Protein Environment

With the combined methodologies at hand, we investigated the absorption spectrum of nile red in different environments. Nile red is known for its strong solvatochromism that can be used as a probe for hydrophobic protein surfaces[51] or lipid droplets.[52] We modeled excited

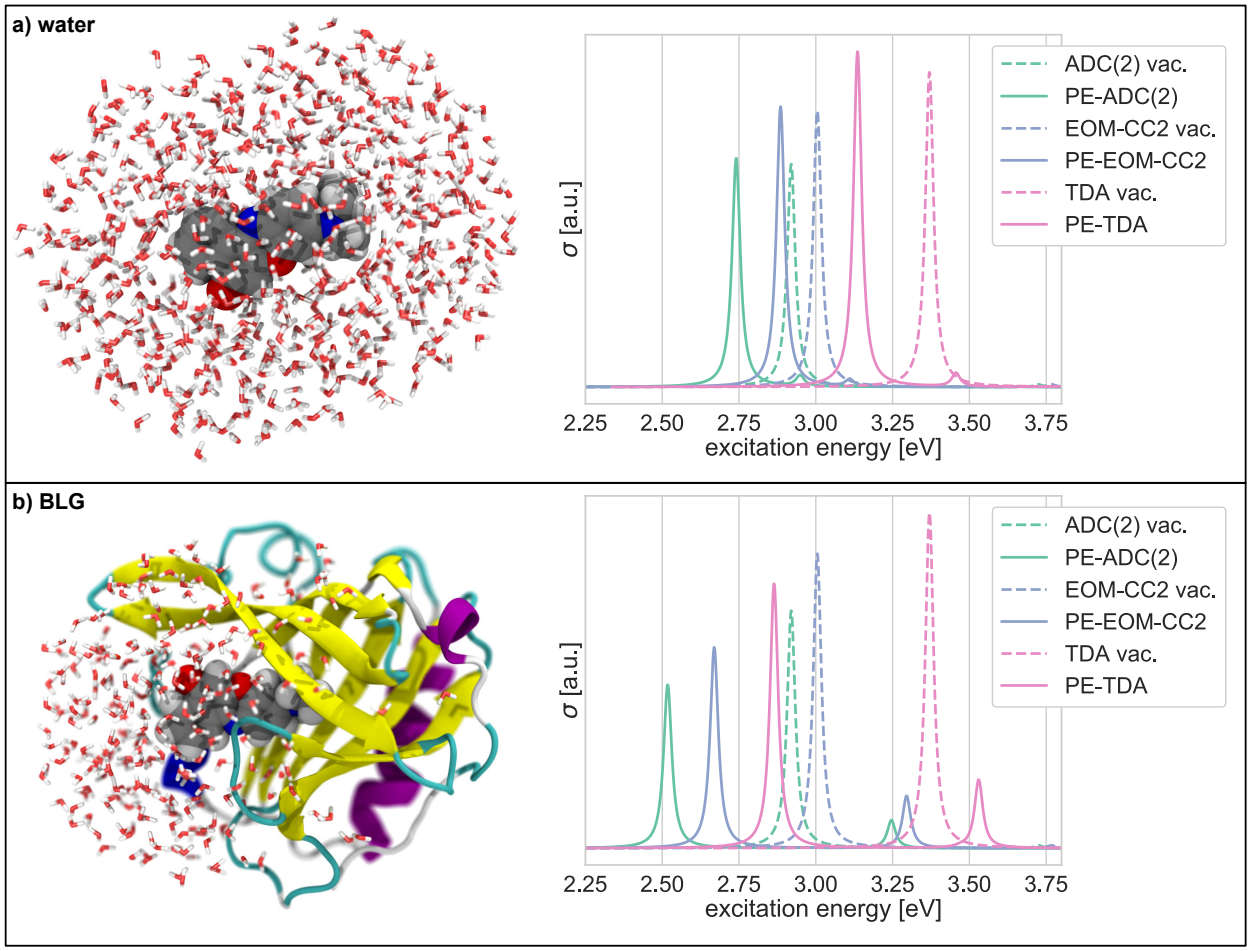


Figure 5: UV/Vis spectrum of nile red employing the three lowest singlet states in **a)** water and **b)** BLG. Dashed lines represent the spectrum in vacuum. The strong solvatochromism both environments is clearly visible.

states of nile red in vacuum, water, and in β-lactoglobulin (BLG). The full computational methodology is given in the Supporting Information. Coordinates of the chromophore, embedding parameters (potential files), Python scripts for Psi4 and PySCF calculations, and input files for Q-Chem calculations can be found in the Supporting Information as a ZIP archive. Here, we consider only a single snapshot in each case, as we seek to demonstrate

the broad applicability of the Cppe library rather than providing statistically converged results. The resulting spectra for vacuum and both environments using ADC(2),[50] EOM-CC2,[53] and TDA/TD-CAM-B3LYP[54] (referred to as TDA in the following) are depicted in Figure 5. Energies, oscillator strengths, and solvent shifts for the energetically lowest, bright singlet state, are summarized in Table 1, whereas results for S_2 and S_3 transitions are given in Tables S3 and S4, respectively, in the Supporting Information. One can clearly see that the lowest peak in the spectrum, corresponding to the $S_0 \rightarrow S_1$ transition, is strongly red-shifted compared to vacuum, where the red-shift is approximately twice as large for BLG as for water.

Table 1: Summary of the first singlet excited state of nile red ($S_0 \rightarrow S_1$) with all employed methods and environments.

Environment	Method	E_{exc} [eV][a]	f[b]	ptLR [eV]	ptSS [eV]	E_{shift} [eV]
vac.	ADC(2)	2.920	0.776	–	–	–
	EOM-CC2	3.005	0.962	–	–	–
	TDA	3.370	1.096	–	–	–
water	ADC(2)	2.741	0.796	−0.061	−0.032	−0.179
	EOM-CC2	2.885	0.976	–	−0.039	−0.12
	TDA	3.136	1.168	–	–	−0.234
BLG	ADC(2)	2.518	0.534	−0.064	−0.037	−0.402
	EOM-CC2	2.669	0.656	–	−0.039	−0.336
	TDA	2.864	0.863	–	–	−0.506

[a] For ADC(2) and EOM-CC2, the perturbative corrections are added to the excitation energy.
[b] Oscillator strength.

This is the case for all employed methods. Of course, the solvent shift cannot solely be attributed to differing environment dielectrics, but also to geometric differences of the chromophore in the diffferent environments. Excluding perturbative corrections, the excitation energied obtained from ADC(2) and EOM-CC2 constantly differ by approximately 0.09 eV. Since the ADC(2) and EOM-CC2 methods are rather similar, this close agreement is expected. Note that the ptSS corrections for the S_1 state are almost identical. The overall solvent shifts for EOM-CC2 differ from ADC(2) by approximately the magnitudes of the

ADC(2) ptLR correction terms, which are not included for EOM-CC2. Even though the absolute energies are not in good agreement between CAM-B3LYP and the wave function methods, the solvent shifts from vacuum to water and vacuum to BLG exhibit smaller differences: For PE-TDA, the red-shift of the S_1 state is more pronounced. Due to the bright nature of the involved state, coupling of the environment to the transition density is strong. Since the coupling is included iteratively in our PE-TDA procedure, the solvent shift is larger than for the perturbatively corrected approaches.

The case study of nile red presents Cppe in action employing all three program packages with different methods for computational spectroscopy. The analysis of environment effects conclusively shows that our implementations can reliably model the excited states of nile red at various levels of theory.

5 Conclusion

We presented the design, implementation, and application of our open-source, modular Cppe library for the PE model. Cppe enabled us to easily implement the PE model in three program packages, together with combined approaches for modeling spectroscopic properties. The capabilities of Cppe were exemplified by simulating the UV/Vis spectrum of the nile red chromophore in water and protein environments using all existing interfaces with different quantum-chemical methods. Our presented step-by-step guide makes it possible to interface Cppe to any host program with minor programming effort, especially facilitated by the Python interface of Cppe. Additionally, the outlined implementation procedure together with the combined methods make the PE model more accessible from an educational point of view. The design of our library will enable support of related polarizable embedding models, such as the fluctuating charges (QM/FQ) model,[26,27] or the capacitance molecular mechanics (CMM) approach,[55] in the future. All in all, the presented library makes the PE model accessible to a broad user base through open-source packages and will hopefully trig-

ger further method development for spectroscopic properties employing PE as environment model.

Acknowledgement

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) by means of the research training group "CLiC" (GRK 1986, Complex Light Control). The authors thank M. F. Herbst for help with design of the library. M.S. thanks L. A. Burns, D. G. A. Smith, R. Di Remigio, and T. D. Crawford for help with integration of Cppe in Psi4. J.M.H.O. acknowledges financial support from the Research Council of Norway through its Centres of Excellence scheme (Project ID: 262695). Support from H2020-MSCA-ITN-2017 training network "COSINE – COmputational Spectroscopy In Natural sciences and Engineering" is also acknowledged.

Supporting Information Available

- Supporting Text with computational methodology and Tables with source code directory structure and options.
- Coordinates of nile red, embedding parameters, Python scripts, and input files to run the example calculations

This material is available free of charge via the Internet at `http://pubs.acs.org/`.

References

(1) Morzan, U. N.; de Armiño, D. J. A.; Foglia, N. O.; Ramírez, F.; Lebrero, M. C. G.; Scherlis, D. A.; Estrin, D. A. Spectroscopy in Complex Environments From QM–MM Simulations. *Chem. Rev.* **2018**, *118*, 4071–4113.

(2) Mennucci, B.; Corni, S. Multiscale Modelling of Photoinduced Processes in Composite Systems. *Nat. Rev. Chem.* **2019**, *3*, 315–330.

(3) Tomasi, J.; Mennucci, B.; Cammi, R. Quantum Mechanical Continuum Solvation Models. *Chem. Rev.* **2005**, *105*, 2999–3094.

(4) Brunk, E.; Rothlisberger, U. Mixed Quantum Mechanical/Molecular Mechanical Molecular Dynamics Simulations of Biological Systems in Ground and Electronically Excited States. *Chem. Rev.* **2015**, *115*, 6217–6263.

(5) Wesolowski, T. A.; Shedge, S.; Zhou, X. Frozen-Density Embedding Strategy for Multilevel Simulations of Electronic Structure. *Chem. Rev.* **2015**, *115*, 5891–5928.

(6) Olsen, J. M.; Aidas, K.; Kongsted, J. Excited States in Solution Through Polarizable Embedding. *J. Chem. Theory Comput.* **2010**, *6*, 3721–3734.

(7) Olsen, J. M. H.; Kongsted, J. *Adv. Quantum Chem.*; 2011; Vol. 61; pp 107–143.

(8) List, N. H.; Olsen, J. M. H.; Kongsted, J. Excited States in Large Molecular Systems Through Polarizable Embedding. *Phys. Chem. Chem. Phys.* **2016**, *18*, 20234–20250.

(9) Steinmann, C.; Reinholdt, P.; Nørby, M. S.; Kongsted, J.; Olsen, J. M. H. Response Properties of Embedded Molecules Through the Polarizable Embedding Model. *Int. J. Quantum Chem.* **2019**, *119*, 1–20.

(10) Sneskov, K.; Schwabe, T.; Christiansen, O.; Kongsted, J. Scrutinizing the Effects of Polarization in QM/MM Excited State Calculations. *Phys. Chem. Chem. Phys.* **2011**, *13*, 18551.

(11) Sneskov, K.; Schwabe, T.; Kongsted, J.; Christiansen, O. The Polarizable Embedding Coupled Cluster Method. *J. Chem. Phys.* **2011**, *134*.

(12) Schwabe, T.; Sneskov, K.; Olsen, J. M. H.; Kongsted, J.; Christiansen, O.; Hättig, C. PERI-CC2: A Polarizable Embedded RI-CC2 Method. *J. Chem. Theory Comput.* **2012**, *8*, 3274–3283.

(13) Eriksen, J. J.; Sauer, S. P.; Mikkelsen, K. V.; Jensen, H. J.; Kongsted, J. On the Importance of Excited State Dynamic Response Electron Correlation in Polarizable Embedding Methods. *J. Comput. Chem.* **2012**, *33*, 2012–2022.

(14) Hedegård, E. D.; List, N. H.; Jensen, H. J. A.; Kongsted, J. The Multi-Configuration Self-Consistent Field Method Within a Polarizable Embedded Framework. *J. Chem. Phys.* **2013**, *139*, 044101.

(15) Pedersen, M. N.; Hedegård, E. D.; Olsen, J. M. H.; Kauczor, J.; Norman, P.; Kongsted, J. Damped Response Theory in Combination With Polarizable Environments: The Polarizable Embedding Complex Polarization Propagator Method. *J. Chem. Theory Comput.* **2014**, *10*, 1164–1171.

(16) Hedegård, E. D.; Olsen, J. M. H.; Knecht, S.; Kongsted, J.; Jensen, H. J. A. Polarizable Embedding With a Multiconfiguration Short-Range Density Functional Theory Linear Response Method. *J. Chem. Phys.* **2015**, *142*, 114113.

(17) Steindal, A. H.; Beerepoot, M. T. P.; Ringholm, M.; List, N. H.; Ruud, K.; Kongsted, J.; Olsen, J. M. H. Open-Ended Response Theory With Polarizable Embedding: Multiphoton Absorption in Biomolecular Systems. *Phys. Chem. Chem. Phys.* **2016**, *18*, 28339–28352.

(18) Hedegård, E. D.; Bast, R.; Kongsted, J.; Olsen, J. M. H.; Jensen, H. J. A. Relativistic Polarizable Embedding. *J. Chem. Theory Comput.* **2017**, *13*, 2870–2880.

(19) Scheurer, M.; Herbst, M. F.; Reinholdt, P.; Olsen, J. M. H.; Dreuw, A.; Kongsted, J. Polarizable Embedding Combined With the Algebraic Diagrammatic Construction: Tackling Excited States in Biomolecular Systems. *J. Chem. Theory Comput.* **2018**, *14*, 4870–4883.

(20) Olsen, J. M. H.; List, N. H.; Steinmann, C.; Steindal, A. H.; Nørby, M. S.; Reinholdt, P. PElib: The Polarizable Embedding library. 2018; `https://doi.org/10.5281/zenodo.1209196`.

(21) Aidas, K.; Angeli, C.; Bak, K. L.; Bakken, V.; Bast, R.; Boman, L.; Christiansen, O.; Cimiraglia, R.; Coriani, S.; Dahle, P.; Dalskov, E. K.; Ekström, U.; Enevoldsen, T.; Eriksen, J. J.; Ettenhuber, P.; Fernández, B.; Ferrighi, L.; Fliegl, H.; Frediani, L.; Hald, K.; Halkier, A.; Hättig, C.; Heiberg, H.; Helgaker, T.; Hennum, A. C.; Hettema, H.; Hjertenæs, E.; Høst, S.; Høyvik, I.-M.; Iozzi, M. F.; Jansík, B.; Jensen, H. J. Aa.; Jonsson, D.; Jørgensen, P.; Kauczor, J.; Kirpekar, S.; Kjærgaard, T.; Klopper, W.; Knecht, S.; Kobayashi, R.; Koch, H.; Kongsted, J.; Krapp, A.; Kristensen, K.; Ligabue, A.; Lutnæs, O. B.; Melo, J. I.; Mikkelsen, K. V.; Myhre, R. H.; Neiss, C.; Nielsen, C. B.; Norman, P.; Olsen, J.; Olsen, J. M. H.; Osted, A.; Packer, M. J.; Pawlowski, F.; Pedersen, T. B.; Provasi, P. F.; Reine, S.; Rinkevicius, Z.; Ruden, T. A.; Ruud, K.; Rybkin, V. V.; Sałek, P.; Samson, C. C. M.; de Merás, A. S.; Saue, T.; Sauer, S. P. A.; Schimmelpfennig, B.; Sneskov, K.; Steindal, A. H.; Sylvester-Hvid, K. O.; Taylor, P. R.; Teale, A. M.; Tellgren, E. I.; Tew, D. P.; Thorvaldsen, A. J.; Thøgersen, L.; Vahtras, O.; Watson, M. A.; Wilson, D. J. D.; Ziolkowski, M.; Ågren, H. The Dalton Quantum Chemistry Program System. *WIREs Comput. Mol. Sci.* **2014**, *4*, 269–284.

(22) Saue, T.; Visscher, L.; Jensen, H. J. A.; Bast, R.; Bakken, V.; Dyall, K. G.; Dubillard, S.; Ekström, U.; Eliav, E.; Enevoldsen, T.; Faßhauer, E.; Fleig, T.; Fossgaard, O.; Gomes, A. S. P.; Hedegård, E. D.; Helgaker, T.; Henriksson, J.; Iliaš, M.; Jacob, C. R.;

Knecht, S.; Komorovský, S.; Kullie, O.; Lærdahl, J. K.; Larsen, C. V.; Lee, Y. S.; Nataraj, H. S.; Nayak, M. K.; Norman, P.; Olejniczak, G.; Olsen, J.; Olsen, J. M. H.; Park, Y. C.; Pedersen, J. K.; Pernpointner, M.; Di Remigio, R.; Ruud, K.; Sałek, P.; Schimmelpfennig, B.; Shee, A.; Sikkema, J.; Thorvaldsen, A. J.; Thyssen, J.; van Stralen, J.; Villaume, S.; Visser, O.; Winther, T.; Yamamoto, S. DIRAC18. **2018**,

(23) Marefat Khah, A.; Karbalaei Khani, S.; Hättig, C. Analytic Excited State Gradients for the QM/MM Polarizable Embedded Second-Order Algebraic Diagrammatic Construction for the Polarization Propagator PE-ADC(2). *J. Chem. Theory Comput.* **2018**, *14*, 4640–4650.

(24) Di Remigio, R.; Steindal, A. H.; Mozgawa, K.; Weijo, V.; Cao, H.; Frediani, L. PCM-Solver: An Open-Source Library for Solvation Modeling. *Int. J. Quantum Chem.* **2019**, *119*, 1–28.

(25) Kaliman, I. A.; Slipchenko, L. V. LIBEFP: A New Parallel Implementation of the Effective Fragment Potential Method as a Portable Software Library. *J. Comput. Chem.* **2013**, *34*, 2284–2292.

(26) Bryce, R. A.; Buesnel, R.; Hillier, I. H.; Burton, N. A. A Solvation Model Using a Hybrid Quantum Mechanical/Molecular Mechanical Potential With Fluctuating Solvent Charges. *Chem. Phys. Lett.* **1997**, *279*, 367–371.

(27) Lipparini, F.; Barone, V. Polarizable Force Fields and Polarizable Continuum Model: A Fluctuating Charges/Pcm Approach. 1. Theory and Implementation. *J. Chem. Theory Comput.* **2011**, *7*, 3711–3724.

(28) Boulanger, E.; Thiel, W. Solvent Boundary Potentials for Hybrid QM/MM Computations Using Classical Drude Oscillators: A Fully Polarizable Model. *J. Chem. Theory Comput.* **2012**, *8*, 4527–4538.

(29) Lipparini, F. General Linear Scaling Implementation of Polarizable Embedding Schemes. *J. Chem. Theory Comput.* **2019**, *15*, 4312–4317.

(30) Shao, Y.; Gan, Z.; Epifanovsky, E.; Gilbert, A. T.; Wormit, M.; Kussmann, J.; Lange, A. W.; Behn, A.; Deng, J.; Feng, X.; Ghosh, D.; Goldey, M.; Horn, P. R.; Jacobson, L. D.; Kaliman, I.; Khaliullin, R. Z.; Kuś, T.; Landau, A.; Liu, J.; Proynov, E. I.; Rhee, Y. M.; Richard, R. M.; Rohrdanz, M. A.; Steele, R. P.; Sundstrom, E. J.; Woodcock, H. L.; Zimmerman, P. M.; Zuev, D.; Albrecht, B.; Alguire, E.; Austin, B.; Beran, G. J. O.; Bernard, Y. A.; Berquist, E.; Brandhorst, K.; Bravaya, K. B.; Brown, S. T.; Casanova, D.; Chang, C.-M.; Chen, Y.; Chien, S. H.; Closser, K. D.; Crittenden, D. L.; Diedenhofen, M.; DiStasio, R. A.; Do, H.; Dutoi, A. D.; Edgar, R. G.; Fatehi, S.; Fusti-Molnar, L.; Ghysels, A.; Golubeva-Zadorozhnaya, A.; Gomes, J.; Hanson-Heine, M. W.; Harbach, P. H.; Hauser, A. W.; Hohenstein, E. G.; Holden, Z. C.; Jagau, T.-C.; Ji, H.; Kaduk, B.; Khistyaev, K.; Kim, J.; Kim, J.; King, R. A.; Klunzinger, P.; Kosenkov, D.; Kowalczyk, T.; Krauter, C. M.; Lao, K. U.; Laurent, A. D.; Lawler, K. V.; Levchenko, S. V.; Lin, C. Y.; Liu, F.; Livshits, E.; Lochan, R. C.; Luenser, A.; Manohar, P.; Manzer, S. F.; Mao, S.-P.; Mardirossian, N.; Marenich, A. V.; Maurer, S. A.; Mayhall, N. J.; Neuscamman, E.; Oana, C. M.; Olivares-Amaya, R.; O'Neill, D. P.; Parkhill, J. A.; Perrine, T. M.; Peverati, R.; Prociuk, A.; Rehn, D. R.; Rosta, E.; Russ, N. J.; Sharada, S. M.; Sharma, S.; Small, D. W.; Sodt, A.; Stein, T.; Stück, D.; Su, Y.-C.; Thom, A. J.; Tsuchimochi, T.; Vanovschi, V.; Vogt, L.; Vydrov, O.; Wang, T.; Watson, M. A.; Wenzel, J.; White, A.; Williams, C. F.; Yang, J.; Yeganeh, S.; Yost, S. R.; You, Z.-Q.; Zhang, I. Y.; Zhang, X.; Zhao, Y.; Brooks, B. R.; Chan, G. K.; Chipman, D. M.; Cramer, C. J.; Goddard, W. A.; Gordon, M. S.; Hehre, W. J.; Klamt, A.; Schaefer, H. F.; Schmidt, M. W.; Sherrill, C. D.; Truhlar, D. G.; Warshel, A.; Xu, X.; Aspuru-Guzik, A.; Baer, R.; Bell, A. T.; Besley, N. A.; Chai, J.-D.; Dreuw, A.; Dunietz, B. D.; Furlani, T. R.; Gwaltney, S. R.; Hsu, C.-P.; Jung, Y.; Kong, J.; Lambrecht, D. S.; Liang, W.; Ochsenfeld, C.; Rassolov, V. A.;

Slipchenko, L. V.; Subotnik, J. E.; Van Voorhis, T.; Herbert, J. M.; Krylov, A. I.; Gill, P. M.; Head-Gordon, M. Advances in Molecular Quantum Chemistry Contained in the Q-Chem 4 Program Package. *Mol. Phys.* **2015**, *113*, 184–215.

(31) M. Parrish, R.; A. Burns, L.; G. A. Smith, D.; C. Simmonett, A.; Eugene DePrince, A.; G. Hohenstein, E.; Bozkaya, U.; Yu. Sokolov, A.; Di Remigio, R.; M. Richard, R.; F. Gonthier, J.; M. James, A.; R. McAlexander, H.; Kumar, A.; Saitow, M.; Wang, X.; P. Pritchard, B.; Verma, P.; F. Schaefer, H.; Patkowski, K.; A. King, R.; F. Valeev, E.; A. Evangelista, F.; M. Turney, J.; Daniel Crawford, T.; David Sherrill, C. Psi4 1.1: An Open-Source Electronic Structure Program Emphasizing Automation, Advanced Libraries, and Interoperability. *J. Chem. Theory Comput.* **2017**, *13*, 3185–3197.

(32) Sun, Q.; Berkelbach, T. C.; Blunt, N. S.; Booth, G. H.; Guo, S.; Li, Z.; Liu, J.; McClain, J. D.; Sayfutyarova, E. R.; Sharma, S.; Wouters, S.; Chan, G. K. L. PySCF: The Python-Based Simulations of Chemistry Framework. *Wiley Interdiscip. Rev. Comput. Mol. Sci.* **2018**, *8*, e1340.

(33) Applequist, J.; Carl, J. R.; Fung, K.-K. Atom Dipole Interaction Model for Molecular Polarizability. Application to Polyatomic Molecules and Determination of Atom Polarizabilities. *J. Am. Chem. Soc.* **1972**, *94*, 2952–2960.

(34) CMake. `https://cmake.org/`.

(35) Guennebaud, G.; Jacob, B.; other, Eigen v3. http://eigen.tuxfamily.org, 2010.

(36) Jakob, W.; Rhinelander, J.; Moldovan, D. pybind11 – Seamless operability between C++11 and Python. 2017; https://github.com/pybind/pybind11.

(37) Van Der Walt, S.; Colbert, S. C.; Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.* **2011**, *13*, 22–30.

(38) Krekel, H.; Oliveira, B.; Pfannschmidt, R.; Bruynooghe, F.; Laugher, B.; Bruhin, F. pytest. 2004; `https://github.com/pytest-dev/pytest`.

(39) Smith, D. G.; Burns, L. A.; Sirianni, D. A.; Nascimento, D. R.; Kumar, A.; James, A. M.; Schriber, J. B.; Zhang, T.; Zhang, B.; Abbott, A. S.; Berquist, E. J.; Lechner, M. H.; Cunha, L. A.; Heide, A. G.; Waldrop, J. M.; Takeshita, T. Y.; Alenaizan, A.; Neuhauser, D.; King, R. A.; Simmonett, A. C.; Turney, J. M.; Schaefer, H. F.; Evangelista, F. A.; Deprince, A. E.; Crawford, T. D.; Patkowski, K.; Sherrill, C. D. Psi4NumPy: An Interactive Quantum Chemistry Programming Environment for Reference Implementations and Rapid Development. *J. Chem. Theory Comput.* **2018**, *14*, 3504–3511.

(40) Walker, H.; Ni, P. Anderson Acceleration for Fixed-Point Iterations. *SINUM* **2011**, *49*, 1715–1735.

(41) Dykstra, C. E. Efficient Calculation of Electrically Based Intermolecular Potentials of Weakly Bonded Clusters. *J. Comput. Chem.* **1988**, *9*, 476–487.

(42) Thole, B. T. Molecular Polarizabilities Calculated With a Modified Dipole Interaction. *Chem. Phys.* **1981**, *59*, 341–350.

(43) Van Duijnen, P. T.; Swart, M. Molecular and Atomic Polarizabilities: Thole's Model Revisited. *J. Phys. Chem. A* **1998**, *102*, 2399–2407.

(44) Meurer, A.; Smith, C. P.; Paprocki, M.; Čertík, O.; Kirpichev, S. B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J. K.; Singh, S.; Rathnayake, T.; Vig, S.; Granger, B. E.; Muller, R. P.; Bonazzi, F.; Gupta, H.; Vats, S.; Johansson, F.; Pedregosa, F.; Curry, M. J.; Terrel, A. R.; Roučka, v.; Saboo, A.; Fernando, I.; Kulal, S.; Cimrman, R.; Scopatz, A. SymPy: Symbolic Computing in Python. *PeerJ Comput. Sci.* **2017**, *3*, e103.

(45) Kauczor, J.; Jørgensen, P.; Norman, P. On the Efficiency of Algorithms for Solving

Hartree-Fock and Kohn-Sham Response Equations. *J. Chem. Theory Comput.* **2011**, *7*, 1610–1630.

(46) Schröder, H.; Schwabe, T. Corrected Polarizable Embedding: Improving the Induction Contribution to Perichromism for Linear Response Theory. *J. Chem. Theory Comput.* **2018**, *14*, 833–842.

(47) Tamm, I. Relativistic Interaction of Elementary Particles. *J. Phys.(USSR)* **1945**, *9*, 449.

(48) Dancoff, S. M. Non-Adiabatic Meson Theory of Nuclear Forces. *Phys. Rev.* **1950**, *78*, 382–385.

(49) Hirata, S.; Head-gordon, M. Time-Dependent Density Functional Theory Within the TammDancoff Approximation. *Chem. Phys. Lett.* **1999**, *314*, 291–299.

(50) Wormit, M.; Rehn, D. R.; Harbach, P. H.; Wenzel, J.; Krauter, C. M.; Epifanovsky, E.; Dreuw, A. Investigating Excited Electronic States Using the Algebraic Diagrammatic Construction (ADC) Approach of the Polarisation Propagator. *Mol. Phys.* **2014**, *112*, 774–784.

(51) Sackett, D. L.; Wolff, J. Nile Red as a Polarity-Sensitive Fluorescent Probe of Hydrophobic Protein Surfaces. *Anal. Biochem.* **1987**, *167*, 228–234.

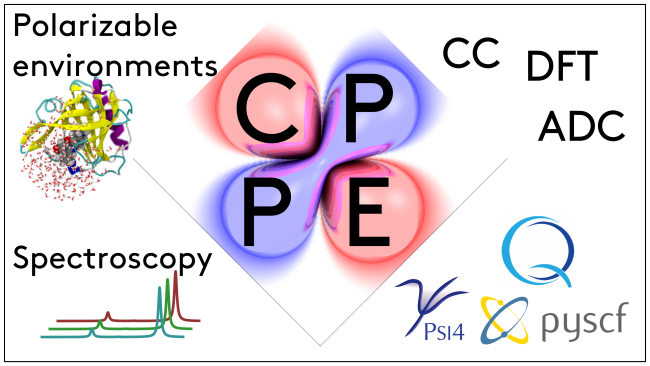
(52) Greenspan, P.; Mayer, E. P.; Fowler, S. D. Nile Red: A Selective Fluorescent Stain for Intracellular Lipid Droplets. *J. Cell Biol.* **1985**, *100*, 965–973.

(53) Christiansen, O.; Koch, H.; Jørgensen, P. The Second-Order Approximate Coupled Cluster Singles and Doubles Model CC2. *Chem. Phys. Lett.* **1995**, *243*, 409–418.

(54) Yanai, T.; Tew, D. P.; Handy, N. C. A New Hybrid Exchange–Correlation Functional Using the Coulomb-Attenuating Method (CAM-B3LYP). *Chem. Phys. Lett.* **2004**, *393*, 51–57.

(55) Rinkevicius, Z.; Li, X.; Sandberg, J. A.; Mikkelsen, K. V.; Ågren, H. A Hybrid Density Functional Theory/Molecular Mechanics Approach for Linear Response Properties in Heterogeneous Environments. *J. Chem. Theory Comput.* **2014**, *10*, 989–1003.

Graphical TOC Entry



For Table of Contents Only.