ddX: Polarizable Continuum Solvation from Small Molecules to Proteins

Michele Nottoli[1], Michael F. Herbst[2], Aleksandr Mikhalev[3], Abhinav Jha[1], Filippo Lipparini[4], and Benjamin Stamm[1]

[1]Universität Stuttgart, Institute of Applied Analysis and Numerical Simulation, Pfaffenwaldring 57, 70569 Stuttgart, Germany
[2]Mathematics for Materials Modelling, Institute of Mathematics & Institute of Materials, École Polytechnique Fédérale de Lausanne, CH-1015 Lausanne, Switzerland
[3]Center for Artificial Intelligence Technology, Skolkovo Institute of Science and Technology Moscow, Russia
[4]Dipartimento di Chimica e Chimica Industriale, Universitá di Pisa, 56124 Pisa, Italy

Abstract

Polarizable continuum solvation models are popular in both, quantum chemistry and in biophysics, though typically with different requirements for the numerical methods. However, the recent trend of multiscale modeling can be expected to blur field-specific differences. In this regard, numerical methods based on domain decomposition (dd) have been demonstrated to be sufficiently flexible to be applied all across these levels of theory while remaining systematically accurate and efficient. In this contribution, we present `ddX`, an open-source implementation of dd-methods for various solvation models, which features a uniform interface with classical as well as quantum descriptions of the solute, or any hybrid versions thereof. We explain the key concepts of the library design and its API, and demonstrate the use of `ddX` for integrating into standard chemistry packages. Numerical tests illustrate the performance of `ddX` and its interfaces.

1 Introduction

Polarizable continuum solvation models (PCSM) are powerful, simple and effective tools that are used to describe solvation effects on molecules and larger biological systems. Most commonly, such models are used by two large communities: Quantum chemists employ them as a cost-effective model to include environmental effects into the prediction of molecular geometries and spectroscopic observables[1–6]. Biophysicists use them to study the electrostatics of large solvated biomolecules, and hence their binding properties[7–9]. Both communities have developed PCSM independently and coded in different software and with different numerical strategies; the models all share a common ingredient, i.e., they treat the solvent as a continuum, infinite, structureless dielectric that is polarized by the solvent and, possibly, back-polarizes it. The electrostatic response of the dielectric is obtained by solving either the generalized Poisson equation or, for (weakly) ionic solutions, the (linearized) Poisson-Boltzmann equation. Despite this common theoretical background, the numerical realizations of PCSM implemented by the two communities are indeed quite different.

In the context of quantum chemistry, PCSM have been developed to achieve quantum mechanical calculations of molecules in solution at very little additional cost compared to the *in vacuo* setting. The most common numerical strategy is based on a boundary element method approach, where the generalized Poisson equation is rewritten as an integral equation for an *apparent surface charge* supported at the boundary of

the molecular cavity that accommodates the solute. The resulting linear equations are usually solved by dense linear algebra techniques, such as LU or Cholesky decomposition. This approach scales cubically and thus in principle expensive. However, its cost is still negligible with respect to the cost associated with the quantum mechanical treatment. On the other hand, as PCSM are typically used to compute structures and properties, a lot of work has been done over the past decades to achieve smooth, differentiable definitions of the solvation energy. These can be used to compute analytical derivatives of the molecular energy, e.g. to compute response properties or derivatives with respect to the nuclear positions (forces and harmonic frequencies). In quantum chemistry, while more sophisticated models like the linearized Poisson-Boltmann exist[10,11], the two most popular models are based either on a dielectric continuum (Polarizable Continuum Model, PCM)[2,4,12,13] or on a conductor continuum (Conductor-like Screening Model, COSMO, also known as C-PCM)[14–16]. Such models have been implemented in virtually every quantum chemistry package and are available in conjunction with most levels of theory, from semiempirical methods[17–20] to sophisticated post-Hartree-Fock ones[21–25] and also with a linear scaling implementation[26].

In the context of biophysics, on the other hand, modeling ionic solutions is paramount, which makes the (linearized) Poisson-Boltzmann ((L)PB) model[27–29] the *de facto* standard. Many numerical realizations of such a model have been implemented and are available to the community in popular software that solve the (L)PB equation using a variety of methods that include finite differences[30–33], finite elements[34–36], and boundary elements[37–47]. As the modeled systems are typically large biomolecules described with a classical force field (i.e., point charges, or sometimes more sophisticated treatments including distributed multipoles and polarizabilities) and not with quantum mechanics, the treatment of solvation can become a cost-dominating factor in such computations. Therefore, much attention has been given to the numerical efficiency of the implementations, and linear scaling techniques are commonly used to afford the treatment of very large systems. Furthermore, due to the complexity of the systems studied, the use of accurate molecular surfaces, such as the Connolly also known as Solvent Excluded Surface (SES) to define the molecular cavity is common, to be compared with the simpler Van der Waals or Solvent Accessible Surfaces (SAS) used in quantum chemistry applications. On the other hand, the calculation of analytical gradients and the coupling to quantum mechanical codes have not received the same amount of attention as the models used in quantum chemistry.

Despite the differences, there are several regions of overlap, from PCSM being applied to large systems treated with multiscale QM/MM methods[48–57], to the possibility of performing Molecular Dynamics simulations in a polarizable continuum solvent[58,59], that makes the possibility of bridging the gap between the two communities interesting.

In the last decade, a new numerical paradigm for continuum solvation models has been proposed. The new algorithm, based on Schwarz's alternating domain decomposition (dd), has originally been proposed by some of us for the COSMO model. The resulting algorithm, which we have named ddCOSMO[60], exhibits all the properties that make it an ideal candidate to bridge the gap between the two communities. ddCOSMO achieves linear scaling in computational cost and memory requirements, can be used to compute analytical gradients[61] and other response properties with arbitrary accuracy, it can be used to descrive very large systems[62], and can be easily interfaced with quantum mechanical[55,63] and polarizable models[54,57,58]. After the initial work on ddCOSMO, the domain decomposition paradigm has been extended to the PCM[64–66] and, more recently, to the LPB model[67]. For the latter models, a straightforward dd implementation exhibits quadratic scaling computational cost. To overcome this difficulty, a linear scaling implementation based on the Fast Multiple Method has further been achieved[68,69]. All three models have analytical gradients and are numerically robust and accurate.

In this contribution, we present `ddX`, an open-source implementation of ddCOSMO, ddPCM, and ddLPB that comes with clear APIs and is designed to be easily interfaced with both classical and quantum descriptions. We believe that the `ddX` library, besides being the coronation of more than ten years of methodological developments, can provide both quantum chemists and biophysicists with a unified tool to treat solvation effects with a polarizable model. It can be used consistently for any application, ranging from the simulation of electronic spectra to molecular dynamics, from quantum mechanical geometry optimizations to the calculation of the electrostatic solvation energy of large biomolecules. The library is distributed on GitHub

(`https://github.com/ddsolvation/ddX`) under the permissive LGPL v3 license and has been provided with an automatically generated documentation, available at `https://ddsolvation.github.io/ddX/`. So far we have interfaced `ddX` both with codes from quantum chemistry (Psi4[70] and `adcc`[71]) as well as classical MD simulation (Tinker)[72], which will be demonstrated below.

This paper is organized as follows: in Sec. 2 we introduce the models available in `ddX` and give a brief overview on the computation of energy and the analytical derivatives. In Sec. 3, we present the layout of the implementation of the `ddX` library along with some details about the fast multipole method (FMM) and our code design. In Sec. 4, we present our interface to Tinker, Psi4 and `adcc` in detail. Lastly, in Sec. 5, we test the `ddX` library on molecules with atoms up to 10^5 atoms. This paper also has an Appendix A where we give the finer details about the matrix formulation for the three models.

2 Methods

2.1 Model Input: Solute Cavity and Density

The `ddX`-library considers solute cavities that can be written as a union of balls, thus including van der Waals (vdW)- and Solvent Accessible Surface (SAS)-cavities[73,74]. The domain-decomposition approach for Solvent-Excluded Surfaces (SES)-cavities has also been developed[75,76], but this does not (yet) fit the `ddX`-framework. We therefore consider a molecule consisting of N atoms, for each atom we assign a ball $\Omega_j := B_{r_j}(\boldsymbol{x}_j)$, centered at the nucleus position $\boldsymbol{x}_j \in \mathbb{R}^3$ with radius $r_j \in \mathbb{R}$. The region occupied by the solute molecule is then given by

$$\Omega = \bigcup_{j=1}^{N} \Omega_j.$$

The region not occupied by the solute, i.e., $\Omega^{\mathrm{C}} = \mathbb{R}^3 \setminus \Omega$, is considered to be occupied by the bulk solvent. Note that without many changes, the method can also be applied if $\boldsymbol{x}_j$ do not necessarily represent nuclear coordinates or just a subset of those (such as is needed in coarse-grained models). The charge density of the solute is denoted by ρ.

Within the framework of `ddX`, the solute molecule is entirely described by the cavity Ω and the charge distribution ρ. As it is customary in continuum solvation models, it is assumed that the charge ρ is supported in Ω, i.e. $\rho(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \Omega^{\mathrm{C}}$.

As we will see in the following, different models and thus different codes have different representations for the charge distribution ρ. Therefore, ρ can be represented either by a sum of (partial) point-charges or, more generally, a sum of multipoles for classical force-field models, or a sum of point-charges (nuclear charges) in combination with the electronic density ρ_{el} for QM-based models.

In the latter case, ρ_{el} is a distribution that is assumed to be supported in Ω and is represented in terms of basis functions arising from the interfaced QM-code. In practical calculations, however, ρ_{el} is not supported in Ω due to the Gaussian- or Slater-type basis functions having tails that extend beyond the molecular cavity and any electronic charge on the complement of Ω is lost, a problem known as the escaped charge problem[77]. This problem has been extensively studied[77–79] and has been found not to be an issue in practical applications.

2.2 Models for Electrostatic Interaction

The `ddX`-library offers three levels of the theory that model the electrostatic interaction between the solute molecule and the bulk solvent. These models differ by the physical interaction between the solute and the bulk (implicit) solvent. We start the presentation with the physically most complex one of the three and present the other two as its simplification:

1. LPB: Linearized Poisson-Boltzmann model[27–29]. In the LPB model, the (implicit) solvent is assumed to be a homogeneous ionic continuum with relative dielectric permittivity $\varepsilon_s < \infty$ and Debye Hückel

screening constant κ_s. Then, the LPB equations are given by: find the potential V that satisfies

$$-\nabla \cdot (\varepsilon \nabla V) + \kappa^2 V = 4\pi\rho, \qquad \text{in } \mathbb{R}^3, \tag{1}$$

where

$$\varepsilon(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \Omega, \\ \varepsilon_s & \text{otherwise,} \end{cases} \qquad \text{and} \qquad \kappa(\boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \Omega, \\ \sqrt{\varepsilon_s}\kappa_s & \text{otherwise.} \end{cases}$$

In this simple model, we do not account for the Steric effects[80] and hence do not include a Stern layer.

2. PCM: Polarizable Conducting Model[2,4,12,13]. The PCM is the particular case of LPB with $\kappa_s = 0$ and $\varepsilon_s < \infty$. The ddPCM method relies on the following equivalent Integral Equation formulation: Find the charge density σ and the intermediate potential Φ_ε that satisfy

$$R_\varepsilon \Phi_\varepsilon = R_\infty \Phi, \qquad \text{on } \partial\Omega, \tag{2}$$
$$\mathcal{S}_0 \sigma = -\Phi_\varepsilon, \qquad \text{on } \partial\Omega, \tag{3}$$

where

$$R_\varepsilon = 2\pi \frac{\varepsilon_s + 1}{\varepsilon_s - 1} I - \mathcal{D}_0,$$

and $\mathcal{S}_0$ and $\mathcal{D}_0$ denote the single and the double-layer boundary operators, respectively, with respect to the Green's kernel $G_0(r) = \frac{1}{4\pi r}$[12].

3. COSMO: COnductor-like Screening MOdel[5,14–16]. The COSMO is the particular case of LPB with $\varepsilon_s = +\infty$ and $\kappa_s = 0$, i.e., the solvent is assumed to be a perfect conductor. In consequence, $V = 0$ in Ω^{C} and the problem reduces to

$$\begin{aligned} -\Delta V &= 4\pi\rho, && \text{in } \Omega, \\ V &= 0, && \text{on } \partial\Omega, \end{aligned} \tag{4}$$

which, for $W = V - \Phi$, can be reformulated as

$$\begin{aligned} -\Delta W &= 0, && \text{in } \Omega, \\ W &= -\Phi, && \text{on } \partial\Omega. \end{aligned}$$

We are interested in the computation of the electrostatic solvation energy and the electrostatic forces of these models. In all three models, the solvation energy E_s is defined as

$$E_s = f(\varepsilon_s) \frac{1}{2} \int_\Omega \rho(\boldsymbol{x})\, W(\boldsymbol{x})\, d\boldsymbol{x},$$

where $W = V - \Phi$ denotes the reaction potential and

$$f(\varepsilon_s) = \begin{cases} 1 & \text{for PCM and LPB,} \\ \dfrac{\varepsilon_s - 1}{\varepsilon_s + 0.5} & \text{for COSMO,} \end{cases}$$

see [60]. The force-vector F_i acting on the i^{th} atom corresponding to the solvation energy is given by

$$F_i = -\nabla_{\boldsymbol{x}_i} E_s. \tag{5}$$

2.3 Discrete Equations in a General Framework

After discretization of the equations using spherical harmonics and domain decomposition techniques, the resulting linear system for all three methods can be written in the general form

$$\mathbf{M}\mathbf{X} = \mathbf{F}, \tag{6}$$

where $\mathbf{F}$ is a given vector, $\mathbf{X}$ is the vector of unknown degrees of freedom, and $\mathbf{M}$ is the given solution matrix. In all three cases, the entries of $\mathbf{X}$ allows to represent the (approximate) reaction potential W and can be used to compute the (approximate) electrostatic solvation energy that is of the form

$$E_s = \frac{1}{2}\left\langle \mathbf{\Psi}, \mathbf{X} \right\rangle, \tag{7}$$

where $\langle \cdot, \cdot \rangle$ denotes the scalar product of the two enclosed vectors and $\mathbf{\Psi}$ is related to the discretization of ρ.

For brevity, we do not provide the matrix and vector entries of $\mathbf{M}$ and $\mathbf{\Psi}$ for each method here, and refer to Appendix for a brief overview. For a detailed derivation and discussion on this topic, we refer to [61, Section 2], [64, Section III], and [67, Section 6.2.2] for the ddCOSMO, ddPCM, and the ddLPB method, respectively.

2.4 Force Computations and Derivatives

For the computation of the electrostatic forces, one needs the computation of gradients as seen by (5). The derivative of the energy with respect to any parameter λ, such as the position of $\boldsymbol{x}_k$ of the k^{th} atom, is given by

$$E_s^\lambda := \frac{\partial E_s}{\partial \lambda} = \frac{1}{2}\left\langle \mathbf{\Psi}^\lambda, \mathbf{X} \right\rangle + \frac{1}{2}\left\langle \mathbf{\Psi}, \mathbf{X}^\lambda \right\rangle = \frac{1}{2}\left\langle \mathbf{\Psi}^\lambda, \mathbf{X} \right\rangle + \frac{1}{2}\left\langle \mathbf{X}_{\text{adj}}, \mathbf{F}^\lambda - \mathbf{M}^\lambda \mathbf{X} \right\rangle \tag{8}$$

where $\mathbf{X}_{\text{adj}}$ is solution to the adjoint linear system

$$\mathbf{M}^* \mathbf{X}_{\text{adj}} = \mathbf{\Psi}, \tag{9}$$

and $\mathbf{M}^*$ is the adjoint of $\mathbf{M}$. Due to the role of $\mathbf{\Psi}$ in eq. 9, we will refer to it also as "adjoint RHS".

For the three models, the RHS $\mathbf{F}$ is written as the product of a term which is specific to the solute and a term which is merely geometric. This allows the factorization (due to the product rule) of its derivative in a term that does not depend on the nature of the solute and a term that has different expressions depending on the nature of ρ. Hence the derivative is written as $\mathbf{F}^\lambda = \mathbf{F}^\lambda_{\text{solvent}} + \mathbf{F}^\lambda_{\text{solute}}$, and we can further write E_s^λ as

$$E_s^\lambda = \frac{1}{2}\left\langle \mathbf{\Psi}^\lambda, \mathbf{X} \right\rangle + \frac{1}{2}\left\langle \mathbf{X}_{\text{adj}}, \mathbf{F}^\lambda_{\text{solute}} \right\rangle + \frac{1}{2}\left\langle \mathbf{X}_{\text{adj}}, \mathbf{F}^\lambda_{\text{solvent}} \right\rangle - \frac{1}{2}\left\langle \mathbf{X}_{\text{adj}}, \mathbf{M}^\lambda \mathbf{X} \right\rangle = E^\lambda_{s,\text{solute}} + E^\lambda_{s,\text{solvent}}. \tag{10}$$

Where $E^\lambda_{s,\text{solute}}$ collects the first two derivative terms and $E^\lambda_{s,\text{solvent}}$ collects the remaining two.

Detailed matrix entries for the derivatives can be found in [61, Section 3], [65, Appendix C], and [69, Section A] for the ddCOSMO, ddPCM, and ddLPB method, respectively.

3 Implementation

With the `ddX` library, we present a uniform implementation of domain decomposition solvation methods. While the main parts of the library are written in `Fortran 2008` making use of the high-performance features of the language for scientific computation, convenient interfaces to `C` as well as `Python` are available as well. Based on this multi-language application program interface (API) a seamless interfacing to a variety

of common quantum chemistry software is possible, which will be demonstrated in Section 4 for both the case of classical and QM codes and employing both our `Fortran` and our `Python` API.

In this section, we will provide an overview of the design of `ddX` and its API, following along the steps of a typical `ddX` calculation. Afterward we provide some notes on the specific fast multipole (FMM) acceleration, which enabled us to achieve linear scaling for all implemented solvation models, as well as some notes on our means of distributing `ddX`.

3.1 Steps of a Calculation

The typical steps of a `ddX` calculation for computing the energy and generic analytical derivatives is as follows:

1. Initialize the library.
2. Compute the primal and adjoint RHSs $\mathbf{F}$ and $\mathbf{\Psi}$.
3. Solve the problem
   (a) Setup the problem by loading the RHSs $\mathbf{F}$ and $\mathbf{\Psi}$.
   (b) Generate a guess for $\mathbf{X}$ and then solve the primal linear system to get $\mathbf{X}$ (eq. 6).
   (c) Compute the energy as $E_s = \frac{1}{2}\langle \mathbf{\Psi}, \mathbf{X} \rangle$ (eq. 7).
   (d) If the calculation of an analytical derivative is requested, generate a guess for $\mathbf{X}_{\text{adj}}$ and then solve the adjoint linear system to get $\mathbf{X}_{\text{adj}}$ (eq. 9).
   (e) Compute the solvation contribution to the derivative using $\mathbf{X}$, $\mathbf{X}_{\text{adj}}$, $\mathbf{F}^{\lambda}_{\text{solvent}}$ and $\mathbf{M}^{\lambda}$ (last two terms eq. 10).
4. Compute the solute contribution to the derivative using $\mathbf{\Psi}^{\lambda}$, $\mathbf{F}^{\lambda}_{\text{solute}}$ and $\mathbf{X}_{\text{adj}}$ (first two terms eq. 10).
5. Finalize the library.

Notably, the steps 3d, 3e, and 4 are specific for the computation of analytical derivatives. Moreover, step 3e is non-zero only if λ is a parameter on which the continuum model explicitly depends, and specifically the most relevant case is when λ are the nuclear positions. Let us illustrate the various steps in three cases:

- In a plain energy calculation for a classical solute, *no* analytical derivative needs to be computed, thus none of the steps 3d, 3e and 4 are required.
- A QM/`ddX` calculation requires the Fock matrix to solve the self-consistent field equations. This is the derivative of the energy with respect to the density matrix elements $P_{\mu\nu}$, thus steps 3d and 4 are required. However, step 3e is not required, as the density matrix elements are not a parameter of the continuum solvation model, and $E^{P_{\mu\nu}}_{s,\text{solvent}} = \frac{\partial E_{s,\text{solvent}}}{\partial P_{\mu\nu}} = 0$.
- Finally, an energy and force calculation for a classical solute, requires all the steps 3d, 3e and 4.

Depending on the nature of the solute density the details of the computation in steps 2 and 4 differs; for example for solutes consisting of point charges, point dipoles or a quantum density each have different expressions to obtain $\mathbf{F}$ or $\mathbf{\Psi}$ in step 2 or the respective derivative contributions from the solute.

In `ddX` we provide implementations for these two steps only in the case of solute densities based on point multipoles. This choice was mainly motivated by three factors: i) point multipoles cover a wide range of use cases, for example classical solutes described by a force field, or the nuclear point charges of QM solutes, ii) point multipoles of arbitrary order can be treated in a common mathematical formalism, iii) in `ddX` we have a tailored implementation of the FMM which can deal with spherical harmonics or multipoles of any order, which can thus be seamlessly employed to accelerate computation in this step. We remark

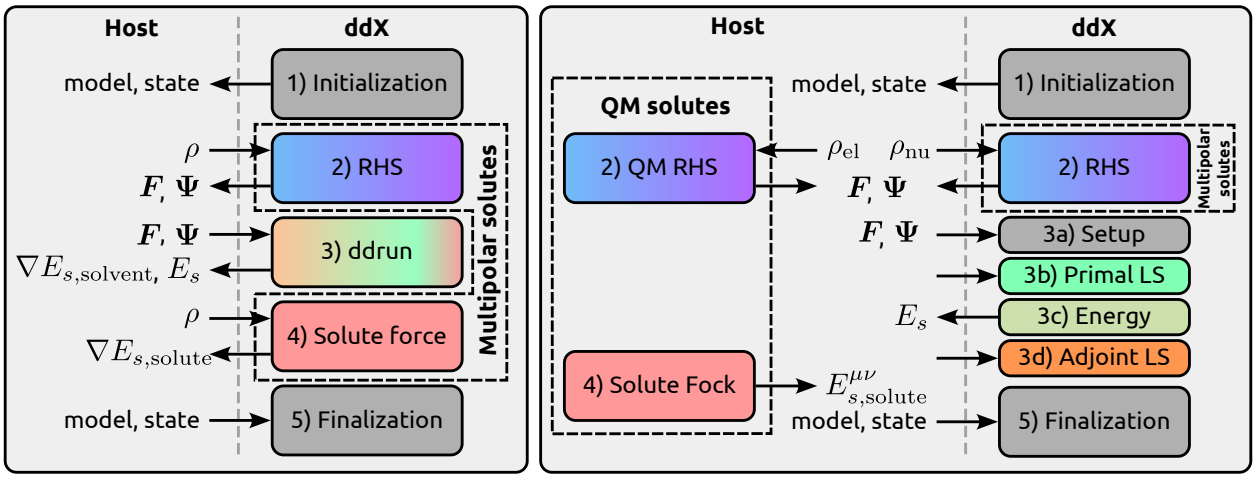

Figure 1: Interactions between a host code and the `ddX` library. Left: case of a point multipolar solute density in the calculation of energy E_s and forces $\nabla E_{s,\text{solute}}$ and $\nabla E_{s,\text{solvent}}$. Right: case of a quantum mechanical density in the calculation of energy E_s and Fock matrix elements $E^{\mu\nu}_{s,\text{solute}}$ Each quantity in the host code represents a variable, and each box in `ddX` represents a call to an API function. Arrows show the arguments passed to the functions and the return values. The variables model and state are passed to all subsequent functions, but this is not represented graphically for clarity. Note also that the solutions of the linear systems and many intermediate quantities are stored in the state variables, so an explicit return is missing.

that providing general implementations for other solute densities is more involved, especially for the case of quantum mechanical solutes, the expressions require the evaluation of electronic integrals and are basis set dependent, see the discussion in Section 4.2. As a result, these are strongly dependent on the host code, and we have opted to not treat this step inside `ddX`.

The following three sections provide details on how the main steps of `ddX` calculations are handled in the implementations. The discussion will refer to the high level API of `ddX`, which are reported in table 1. See Figure 1 for a graphical representation of external codes using `ddX`. Both the case of a classical solute in a calculation of energy and force, and of a QM solute in a calculation of energy and Fock matrix is reported. To make the following discussion more concise, we will only mention the function names in `Fortran`. Table 1 can be used to infer the correspondent function names in `C` and in `Python`.

Purpose	`Fortran`	`C`	`Python`
Housekeeping	`ddinit`	`ddx_allocate_model`	`Model`
	`allocate_state`	`ddx_allocate_state`	`State`
	`deallocate_model`	`ddx_deallocate_model`	—
	`deallocate_state`	`ddx_deallocate_state`	—
Functionality for multipoles	`allocate_electrostatics`	`ddx_allocate_electrostatics`	—
	`multipole_electrostatics`	`ddx_multipole_electrostatics`	`Model.multipole_electrostatics`
	`deallocate_electrostatics`	`ddx_deallocate_electrostatics`	—
	`multipole_psi`	`ddx_multipole_psi`	`Model.multipole_psi`
	`multipole_force_terms`	`ddx_multipole_force_terms`	`State.multipole_force_terms`
Solving the problem	`ddrun`	`ddx_ddrun`	`State.ddrun`

Table 1: API functions in `Fortran`, `C`, and `Python`. Note, that we use "—" to indicate operations, which are automatic in the `Python` interface, such that no explicit function call is needed

3.2 Step 1: Initialization

During the initialization step, the two high-level data structures which are used by `ddX` are generated. `ddX` works with a "model" object and a "state" object:

- The model object (`ddx_type` in `Fortran`) collects all the parameters which control the calculation, like the chosen continuum solvation model, the dielectric permittivity, the information about the discretization, cavity information (the number, centers, and radii of the spheres) and similar. Furthermore, it contains all the precomputed constants which depend on the geometry of the system, and contains the preallocated workspaces, such that most of the memory allocation is done during the initialization phase. This object does not contain variables which depend on the solute's density, like RHSs or solutions.
- The state object (`ddx_state_type` in `Fortran`) contains all the quantities that depend on a given solute density, like RHSs, solutions, and various intermediate quantities for computing analytical derivatives.

Using separate objects for the state and for the model makes it possible to concurrently solve different electrostatic problems characterized by the same geometry. This could be helpful within certain QM calculations of molecular response properties, which require solving a set of response equation for each perturbation, and thus to treat simultaneously multiple perturbed densities. Note, however, that if the geometry is updated, it is necessary to recompute a new model object.

The model object allocation is done by calling the subroutine `ddinit` in `Fortran`, or the corresponding functions in `C` and `Python`. This function performs three main steps: i) a consistency check on the input parameters, ii) the preallocation of all the workspaces, and iii) the precomputation of the constants. The host code needs to provide all the parameters that control the calculation, of which some are mandatory, like the chosen continuum solvation model, cavity information, and the dielectric permittivity, whereas the others are optional and can be used to finely tune the method. In `Fortran` and `Python`, we used keyword arguments for the optional parameters; in this way, it is possible to implement additional functionalities that require new parameters in the future without breaking backwards compatibility with existing external software.

The state object allocation is done by calling the subroutine `allocate_state`, or the corresponding functions in `C` and `Python`. Allocating the state object requires a model object, as the quantities to be allocated and their size depend on the specific nature of the calculation. At this point, the state object is simply an empty container: the information about the RHSs has yet to be loaded, and the solutions have to be computed.

3.3 Step 2: Building the RHSs for Multipolar Solutes

As anticipated, step 2 depends on the nature of the solute, and in `ddX` itself we provide support for densities written as point multipolar distributions. For this reason, we cover here only such a case. For other densities, and in particular for QM ones, some functionalities need to be added to the host code. Further information will be given in the description of the Psi4-interface, Section 4.2.

The primal RHS $\mathbf{F}$ consists of the electrostatic potential and possibly its higher-order derivatives (field and field gradient, depending on the PCSM and on whether the forces are requested). To encompass all possible cases without extensive modifications to the host code, the various electrostatic properties are collected into a high level data type.

In `Fortran` and `C` we provide subroutines for allocating and populating the electrostatics object (`ddx_allocate_electrostatics`, `ddx_multipole_electrostatics`). In `Python`, the same operations are done by the method `Model.multipole_electrostatics`.

The adjoint RHS $\mathbf{\Psi}$ is simpler, as it is the same regardless of the PCSM and kind of computation. In `Fortran` it is computed respectively by calling `multipole_psi`, or in `C` and `Python` with the corresponding functions.

3.4 Step 3: Solving the Problem

Solving the electrostatic problem requires loading the RHSs in the state object, solving the primal and adjoint linear systems, and possibly computing the energy and forces. We provide a high-level procedure that combines the steps together and returns the energy and, if requested, the solvent contribution to the forces. The subroutine for this step is `ddrun` in `Fortran`.

This subroutine takes as input the model and state objects, the RHSs, and the tolerance for the linear systems and gives in output the energy and, if requested, the forces. An optional argument controls if the guesses for the linear systems should be generated from scratch or read from the state object. This might help in the case of repeated calculations for similar RHSs, as it might be the case during the iterative solution to the self-consistent field problem.

Finally, if requested, the subroutine computes the solvation contribution to the nuclear gradients (step 3e).

3.4.1 Finer Control

Some particular applications may require a finer control over the workflow. For this reason, we decided to expose a lower-level API alongside the high-level API described so far. In particular, it is possible to run separately the steps 3a, 3b, 3c, 3d and 3e by calling the associated functions. The same functionality is exposed in `Fortran`, `C`, and `Python` as shown in table 2. The `setup` function takes care of loading the RHSs in the state and setting up the calculation; this is not required in `Python`, as this step is done implicitly by the constructor. The functions `fill_guess` and `fill_guess_adjoint` can be used to generate a guess for the linear system solver; this step can be skipped if a better guess is already present in the state. The `solve` and `solve_adjoint` functions solve the primal (step 3b) and adjoint linear systems (step 3d). The `energy` function computes and returns the energy (step 3c), and, finally, the `solvation_force_terms` functions assemble and return the solvent contributions to the forces (step 3e).

Purpose	`Fortran`	`C`	`Python`
Solving the problem	`setup`	`ddx_setup`	(`State`)
	`fill_guess`	`ddx_fill_guess`	`State.fill_guess`
	`solve`	`ddx_solve`	`State.solve`
	`fill_guess_adjoint`	`ddx_fill_guess_adjoint`	`State.fill_guess_adjoint`
	`solve_adjoint`	`ddx_solve_adjoint`	`State.solve_adjoint`
	`energy`	`ddx_energy`	`State.energy`
	`solvation_force_terms`	`ddx_solvation_force_terms`	`State.solvation_force_terms`

Table 2: Low-level API functions in `Fortran`, `C` and `Python`.

3.5 Step 4: Solute Contributions to the Derivatives

The expressions that need to be evaluated in this step are quite general, as they depend on both the nature of the solute, and the specific differentiation variable λ. For this reason, in the most general case, the step should be carried out in the host code, e.g. in case of the Fock matrix.

However, `ddX` provides an implementation for this step for multipolar solutes and for λ being the nuclear coordinates. This is an interesting case, as the forces are often needed, they have more cumbersome expressions with respect to other analytical derivatives, and in `ddX` this step benefits from the internal FMM library.

The solvent contribution to the forces is computed by calling `multipole_force_terms` in `Fortran` or the corresponding functions in `C` and `Python`. These functions take as arguments the multipolar distribution, the model object and the state object. The latter contains information about the adjoint solution $\mathbf{X}_{\text{adj}}$ which is required to assemble the scalar products in eq. 10.

3.6 Step 5: Finalization

The finalization of the library is done by deallocating the memory which is occupied by the model object, the state object, and the electrostatics container. These deallocations are done in `Fortran` by respectively calling `deallocate_model`, `deallocate_state` and `deallocate_electrostatics`. In `C` the library provides the corresponding functions, whereas in `Python` the deallocation is handled by the garbage collector.

3.7 FMM Acceleration

Many of the steps reported in 3.1 require the computation of electrostatic interactions and are natively quadratically scaling. Such steps are computing the RHS $\mathbf{F}$, solving the linear systems for ddPCM and ddLPB, computing the solvation contribution to the forces for ddPCM and ddLPB, and finally computing the solute contribution to the forces. All of these steps are accelerated with the FMM[81] and are linear scaling in N, except for the initialization, which scales like $N\log(N)$.

The FMM engine implemented in ddX is tailored to accelerating all the calculations involved in the various domain decomposition algorithms. Here, we do not present our FMM implementation in detail, as it is already covered by several papers, and an in-depth explanation of the `ddX` internal FMM library is given in [68]. However, we provide an explanation of the main differences with respect to traditional FMM schemes.

- **Clusterization:** In `ddX`, we use a non-standard adaptive tree instead of the octree approaches used by standard libraries. The spheres building the cavity are usually arranged according to non-uniform distributions; hence, a standard uniform boxification scheme would produce an unbalanced octree, leading to the suboptimal efficiency of the FMM algorithm. This problem is solved using a different clusterization scheme, the recursive inertial bisection at the cost of having non-standard (and thus non-precomputable) M2M and L2L operators from one tree level to another one. In the inertial bisection step, the distribution of points is split in two by finding the hyperplane that cuts along the moment of inertia. By applying this step recursively, we obtain the recursive inertial bisection algorithm[82,83]. This guarantees a balanced tree and thus optimal number operations. The construction of the tree scales like $N\log(N)$ as the standard octree generation.

- **Spherical harmonics:** In usual libraries, the sources and targets are usually charges or at the lowest order Cartesian multipoles. In `ddX`, we allow the sources and targets to be arbitrary multipolar distributions in real spherical harmonics.

- **Different nature of sources and targets:** Traditional libraries are often meant for symmetric interactions in which the nature of the sources and targets is similar. In `ddX`, the sources are the multipolar distributions at the center of the spheres, while the targets are the integration points on the surface of the spheres. This has two main consequences: i) a particle-to-multipole step (P2M) is missing, whereas a local-to-particle (L2P) step takes care of evaluating the electrostatic properties at the integration points; ii) the whole FMM scheme is not symmetric, as there is an asymmetry in the targets and sources despite the kernel being symmetric.

3.8 Software Distribution and Build System

The source code of `ddX` is freely available on GitHub at `https://github.com/ddsolvation/ddX` under the LGPL v3 license with an automatically generated documentation available at `https://ddsolvation.github.io/ddX/`.

In `ddX` we make use of the CMake build system to detect our dependencies and orchestrate the build of the library. By default, only the `Fortran` and `C` parts of the library are built and linked into a shared object `libddx.so` as well as the program `ddx_driver`. The former can be linked to third-party codes using the headers (`ddx.h` for `C` or compiler-specific `.mod` files for `Fortran`), while the latter provides a standalone

runner for ddX solvation models based on a plaintext input file. In this form the dependencies of the code are minimal, being them limited to BLAS and LAPACK, as well as an OpenMP-compatible compiler.

Alternatively, the library can be installed as a Python package using the pip utility and the Python setuptools module. In this form we provide the code through conda-forge and PiPy, allowing for an even more user-friendly installation — simply by issuing pip install pyddx or conda install conda-forge::pyddx. This also takes care of automatically providing the additional Python-specific dependencies (Pybind11, numpy, scipy) the pyddx Python package requires.

4 Interfacing

In this section, we illustrate how an interface between ddX and an existing code can be achieved by illustrating in detail two examples: one with a QM, namely, Psi4[70], and one with a classical molecular dynamics code, namely, Tinker[72]. The first uses the Python bindings and the second uses the native Fortran API.

4.1 Tinker-interface

Tinker is a collection of different programs and tools that perform various kinds of calculations on systems described using classical molecular mechanics (MM)[72]. By coupling Tinker with ddX we want to allow Tinker to include solvation contributions to energy and forces computed with ddX. This requires four main modifications to Tinker: i) as customary in Tinker, implementing a global data structure that contains all the ddX intermediates and parameters; ii) changing the input parsing to read also ddX parameters; iii) changing the way the energy is computed; iv) changing the way the forces are computed.

Tinker mostly works using global variables, all the variables related to the same problem are collected in a Fortran module, and whenever they need to be used, the corresponding module is imported. In our case, we followed the same pattern and we created a ddx_interface module. This contains pointers to the ddX quantities (data, state, error, and electrostatics) and temporary storage for the parameters.

We will not discuss the input parsing, as it is technical and unrelated to how to use ddX. It is sufficient to know that, when a Tinker input file is read, all the relevant ddX parameters are read and stored in the ddx_interface module. The initialization of ddX is not done at this step, as many important tasks of Tinker — such as molecular dynamics — require updating the geometry, and as the geometry changes the ddX initialization needs to be done from scratch. So it is more convenient to move the ddX initialization to where the energy and forces are computed.

In Tinker, all the modules that deal with certain interaction terms are split into one function that computes only the energy, and one that computes energy and forces. In our case, we slightly deviate from the standard pattern and we added a single function called ddx_run which has a logical argument for doing the forces. In turn, this is called by the Tinker functions for solvation, if a ddX solvation is set.

Algorithm 1 shows the inner workings of the ddx_run function, the functions called reflect the example case reported in fig. 1. All the inputs are read from global variables in Fortran modules, and the outputs (energy and optionally forces) are also written to global variables in modules.

The first three functions called in lines 2, 3, and 4 are novel additions to Tinker and mostly take care of converting Tinker quantities to ddX quantities, both in terms of format and units of measure. ddx_build_multipoles takes care of reading the multipoles from the Tinker arrays and converting them to real spherical harmonics in atomic units. ddx_build_cavity uses the atom types and positions to create a list of sphere centers and radii in atomic units. ddx_add_sph adds optional user-specified spheres to the list.

Then, ddx_init is a simple wrapper to ddinit which takes also care of allocating $\boldsymbol{\Psi}$. The calls at lines 6 and 7 are direct to the ddX API and compute the primal and adjoint RHSs. The problem is solved by calling ddrun, which is part of the ddX API, and finally, if the forces are requested, the second contribution is computed by calling multipole_force_terms which is again part of the ddX API. The finalization of the library is done by calling ddx_free, this is a wrapper to deallocate_model and deallocate_state which deallocates also $\boldsymbol{\Psi}$ and the arrays containing the multipoles, centers, and radii.

Algorithm 1: Using `ddX` from Tinker. Note that for simplicity, most arguments are omitted from function calls.

```
1  subroutine ddx_run(do_forces):
2      call ddx_build_multipoles() ;
3      call ddx_build_cavity();
4      call ddx_add_sph();
5      call ddx_init();
6      call multipole_electrostatics();
7      call multipole_psi();
8      call ddrun(do_forces);
9      if do_forces then
10         call multipole_force_terms();
11     end
12     call ddx_free();
```

To conclude, coupling Tinker with `ddX` was possible by adding a new module `ddx_interface`, an input parsing function, three helper functions (`ddx_build_multipoles`, `ddx_build_cavity`, and `ddx_add_sph`), a wrapper to `ddinit`, a finalizing function (`ddx_free`) and a main driver (`ddx_run`) of only 166 lines of code.

4.2 Psi4-interface

This section discusses the integration of `ddX` with Psi4[70], a software suite for performing a wide range of first-principle quantum-chemistry calculations. Currently the implicit solvation models of `ddX` can be used both to compute self-consistent ground states (e.g. HF or DFT methods) as well as subsequent linear response properties including excited states computations based on TDDFT. For more details on using `ddX` with Psi4 see the `ddX` section of the Psi4 user manual at `https://psicode.org`.

Following the previous detailed discussion[84] about how to couple domain-decomposition solvation methods to QM host codes, three main modifications were needed inside Psi4: (1) the handling of the additional input parameters for `ddX`, (2) the computation of solvation energy and Fock matrix contributions as well as (3) the addition of integration routines for electrostatic quantities, which are based on a Becke-type quadrature scheme, which Psi4 uses for DFT computations. This quadrature is used for `ddX`, since it is able to properly deal with the overlapping atom-centred balls Ω_j of the `ddX` cavity[84]. (1) and (3) are strongly dependent on the internals of Psi4, thus of minor interest to implement interfaces with other host codes; we will thus focus our discussion on point (2).

Algorithm 2 sketches our main addition to Psi4, the function `get_solvation_contributions`. In a ground state calculation this function is called in each SCF step to add to the Fock or Kohn-Sham matrix the solvation contribution which is computed from the current density matrix. The first block of Algorithm 2 performs the basic setup, which only needs to be done once, e.g. in the first SCF iteration. This step processes the user-defined parameters discussed in Section 3.2 to initialize the `ddX` model (`pyddx.Model`) as well as the Psi4-specific helper classes for the required analytical (`MintsHelper`) and numerical (`NumIntHelper`) integrals. Based on these helper classes, the electronic contributions to $\mathbf{F}$ and $\mathbf{\Psi}$ (Step 3a in Section 3.1) as well as the solute contributions to the derivatives (Step 4) will be computed on the host side. For the detailed expressions, we refer the reader to the aforementioned method-specific literature[61,64,67]. Next, the nuclear contributions are added by calling `model.multipole_electrostatics()` and `model.multipole_psi()`. If this is the first iteration of the self-consistent field loop, the resulting $\mathbf{F}$ and $\mathbf{\Psi}$ are used to initialize the `pyddx.State`. This object in particular stores the RHS of the linear systems (6) and (9) as well as iterated solution vectors $\mathbf{X}$ and $\mathbf{X}_{\text{adj}}$. A following call to `state.fill_guess()` and `state.fill_guess_adjoint()` generates a crude initial guess for $\mathbf{X}$ and $\mathbf{X}_{\text{adj}}$. As discussed in Section 3.4.1 in any of the subsequent SCF

Algorithm 2: Pseudocode illustrating the integration of `ddX` within Psi4. The function takes a density matrix (or transition density matrix) and returns the respective Fock matrix contribution. Psi4 specific code is highlighted.

```
def get_solvation_contributions(density_matrix):
    # Setup
    if first SCF iteration then
        model = pyddx.Model()
        mints = psi4.core.MintsHelper()
        numints = psi4.core.NumIntHelper()
        scaled_ylm = model.scaled_ylm()
    end
    # Electronic contributions to F and Ψ
    numints.dd_density_integral(scaled_ylm, density_matrix)
    mints.electrostatic_potential_value()  # For LPB only
    mints.electric_field_value()
    # Nuclear contributions to F and Ψ
    model.multipole_electrostatics()
    model.multipole_psi()
    # Setup linear systems (6) and (9)
    if first SCF iteration then
        state = pyddx.State(model, F, Ψ)
        state.fill_guess()
        state.fill_guess_adjoint()
    else
        state.update_problem(F, Ψ)
    end
    # Solve (6) and (9)
    state.solve()
    state.solve_adjoint()
    # Use X and X_adj to build Fock terms
    numints.potential_integral(scaled_ylm, X)
    mints.induction_operator(X, X_adj)   # For LPB only
    psi4.core.ExternalPotential().computePotentialMatrix(X_adj)
```

iterations we only update the RHS of the ddX equations (state.update_problem()), and keep the solution from the previous SCF step to be used as a guess for the ddX iterative solver. This reduces the required number of iterations to solve the updated linear systems (6) and (9) in particular in the later SCF steps, where the density matrix only changes little. Next, the forward and adjoint problems defined inside the state object are solved and their solutions $\mathbf{X}$ and $\mathbf{X}_{\text{adj}}$ employed to compute the Fock matrix contributions — using again the host-specific integration routines.

Finally, we consider the computation of solvated excited states in a linear-response TDDFT formalism. Psi4 already offers the method set_external_cpscf_perturbation to add further potential terms to the response equations. We use this function to add a call to get_solvation_contributions when computing vertical excitation energies. For this, we follow the standard approach[85] to only consider the fast (electronic) components in the solvent's response, which in Algorithm 2 reflects in two notable changes. First, we employ the *optical* permittivtiy ε_∞ instead of the static permittivity ε_s when setting up the pyddx.Model and second, we skip over lines 14 and 15, i.e., we do not add the nuclear contribution to $\mathbf{F}$ and $\mathbf{\Psi}$.

4.3 Employing ddX-Solvated Ground States in adcc

ADC-connect (adcc)[71] is a software suite for performing excited states calculations based on the algebraic-diagrammatic construction scheme for the polarization propagator (ADC)[86,87]. A key focus of the code is a modular design, such that multiple host codes can be employed to supply the Hartree-Fock reference. Additional capabilities, such as interfaces to solvation models, can be passed from the host code to adcc as well. In combination with Psi4 this includes the treatment of solvation effects when computing excitation energies. In the past this strategy has already been employed to perform ADC calculations, which consider environmental effects via a polarisable embedding (PE)[71,88] (using cppe[89]) or using the integral-equation formulation of PCM (using PCMSolver[90]). In a similar fashion, we extended adcc in this work to make use of ddX-based solvation models. Both a full linear response treatment[91] as well as an approach based on perturbative linear-response corrections[85] is now available in adcc in combination with ddX solvation models. In both cases Algorithm 2 is called directly from adcc to compute the effect of the fast solvent components on the excitation (i.e. we employ the optical permittivity and no nuclear contributions).

This demonstrates the seamless integration of ddX within a larger existing Python-based ecosystem for quantum chemistry involving the third-party codes Psi4 and adcc.

4.4 Standalone driver

Alternatively, the library can also be run as a standalone using the previously mentioned ddx_driver. The program ddx_driver is a very simple Fortran code that performs two main functions: it performs the initialization from the file and then performs the steps given in 3.1. In this case, the initialization from the input file is done with a separate Fortran function called ddfromfile, which reads the text-based input file and then calls ddinit with appropriate parameters.

5 Example Applications

ddX was tested through the Tinker/ddX and the Psi4/ddX interfaces. All the calculations have been performed on the JUSTUS2 computer cluster[92], on nodes equipped with 2 Intel Xeon 6252 Gold and up to 384 GB of memory. For what concerns Tinker/ddX we used our fork of Tinker 8[72] at commit d53669b, this is available on GitHub at ddsolvation/tinker-ddX. Tinker was then linked to the ddX shared library obtained by compiling version 0.6.0. Both Tinker and ddX were compiled using the Intel compiler version 19.1.2 and MKL libraries version 2020.2. For what concerns Psi4/ddX we used the version Psi4 at commit 990b0e8. The source code was built in a conda environment containing adcc version 0.15.17 and ddX version 0.6.0 provided through PyPI. Note that in this case, ddX was compiled using the GNU compiler available in the conda environment. Data for reproducibility is available at [93]. The archive contains all the input and output files of the simulations, the conda environment used for Psi4/ddX and Psi4/adcc/ddX calculations, a

copy of the employed versions of `ddX`, Tinker and Psi4, and the `Python` scripts used for generating the tables and plots of this paper.

5.1 Benchmarks

We investigated the scaling of the three models by running calculations on systems of increasing size. To do so, we first converted the protein data bank (PDB) files into Tinker input files using the Tinker tool `pdb2xyz`. We assigned the parameters using the Amber 99 force field[94], which is bundled within Tinker. Further information about the input structures is given in table 3.

PDB code	N atoms	Ref.
2p7r	70	[95]
1etn	143	[96]
1du9	381	[97]
1caa	975	[98]
1d3w	2051	[99]
1hde	9734	[100]
1ju2	20288	[101]
6ftl	39151	[102]
1stm	138408	[103]
1cnl	169	[104]
1ucs	997	[105]

Table 3: PDB codes and number of atoms for the structures used in the benchmark (upper part) and in the MD simulations (bottom part).

Then, for each input structure and for each model among vacuum, ddCOSMO, ddPCM, and ddLPB, we used the Tinker tool `testgrad` to compute the energy and the analytical forces. The execution time for the evaluation of the energy and analytical forces was measured using the `time` command, which also returns the maximum memory usage.

The time required by the various calculation steps is reported in figure 2. The time required to assemble the adjoint RHS is negligible with respect to the rest; the next steps, in terms of cost, are the computation of the RHS and of the forces, which also show a perfectly linear scaling regime. Finally, the most expensive steps are the two linear systems; these also show a slightly super-linear scaling regime due to an increasing number of iterations for the largest systems. For the primal linear system of ddCOSMO, the analysis shows that the number of iterations depends on the globularity of the molecule; see [106]. As shown in a few examples in [107, Section 5], biological molecules like proteins and DNA often have a chain-like structure and are thus not globular, and a constant number of iterations can be expected, but with some natural variation as some molecules are more globular than others that explains this performance.

The overall performance of `ddX` is remarkably good. For the largest system, which consists of more than 138,000 atoms, the total time required to compute the energy ranges between 3 minutes for ddCOSMO, 4 minutes for ddPCM and 56 minutes for ddLPB, the timings increasing to 9, 10 and 204 minutes respectively, if also the forces are computed. We remark that the timings were obtained on a single computer node equipped with two Intel Xeon 6252 Gold CPUs, which is commonly available hardware, and that the results are obtained using conservative discretization parameters, that ensure no compromise on the accuracy of the compute gradients.

Various previous studies can further illustrate the good performance: ddCOSMO was used to solvate viral capsides of up to 7 million atoms in less than 1 hour[57], ddPCM was used to solvate a system of ∼625k atoms in ∼5 hours on a single core[68], and finally ddLPB was tested against the TABI-PB and the APBS finite difference method LPB solvers, finding that it performs similarly for loose accuracies and that outperforms both when a higher accuracy is requested[69].

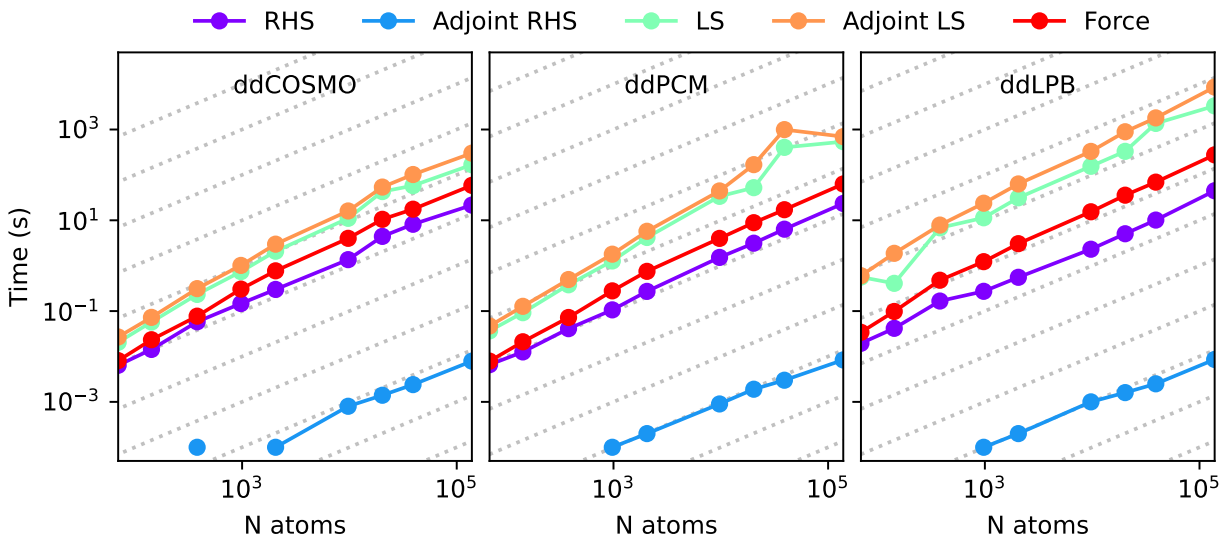

Figure 2: Scaling of the various steps of a `ddX` calculation for systems of increasing size for the three models (ddCOSMO, ddPCM, ddLPB). The various steps are reported using different colors and different models using different panels. The gray dotted lines report an indication of the linear scaling regime.

Figure 3 reports the peak memory usage for the three models, which is only an estimate as the values are computed as the difference between the peak memory usage using `ddX` and the peak memory usage of a vacuum calculation. The three models present a perfectly linear scaling memory consumption, and as expected, the memory consumption increases with the complexity of the model, so it increases going from ddCOSMO to ddPCM and finally to ddLPB.

5.2 Molecular Dynamics

The interface with Tinker and the rigorous analytical gradients implemented in `ddX` allow one to perform MD simulations. We tested the MD simulations on two systems (reported in the bottom of table 3). For each PDB structure, the input files were prepared using the `pdbxyz` tool from Tinker and the Amber 99 sb force field[108], which is bundled in Tinker. The input files were used to run the first 2 ps of NVT MD simulation at a temperature of 300 K, and then the restart files were used to run 2 ps long NVE simulations. The simulations were repeated using two different threshold for the linear solvers of `ddX`, 10^{-4} as a loose threshold and 10^{-8} as a tight threshold. These two simple-minded examples show that the forces computed by ddX are indeed accurate, making thus rigorously energy conserving NVE simulations with ddX possible.

Figure 4 reports an analysis of the energy conservation during the NVE simulations for the two systems and for the four models (including vacuum). The energy conservation is overall good, as it remains of the same order of magnitude as the one observed for the vacuum trajectory. In terms of relative energy conservation, the energy fluctuations in the antifreeze protein MD remain in the -2–4‰ range, whereas the ones of the peptide remain in the ±2‰ range. The usage of a tighter threshold mostly affects the short-time fluctuations, with a more pronounced effect on ddLPB due to the inner-outer linear system solver.

5.3 Psi4 Calculations

The `ddX` interface with Psi4 was tested by comparing between using `ddX` and using PCMSolver[90], which we use as a reference. As a test molecule, we choose the dye nile red. This molecule presents a large

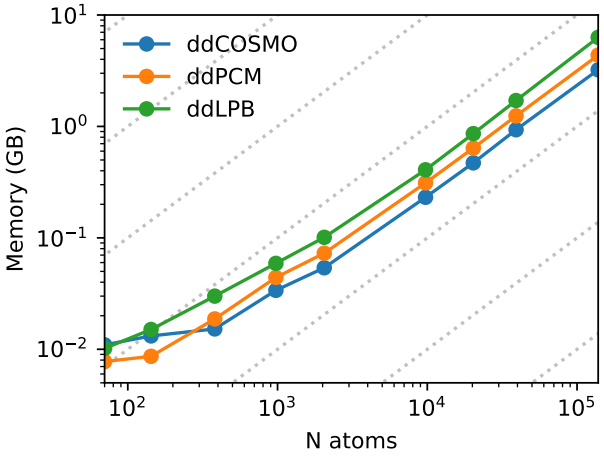

Figure 3: Memory consumption of `ddX` in a energy and forces calculation done with the Tinker/`ddX` implementation. The three models (ddCOSMO, ddPCM and ddLPB) are reported with different colors. The gray dotted lines report an indication of the linear scaling regime.

solvatochromic red-shift and is of medium size (with 42 atoms) which makes it an interesting test case for continuum solvation models.

First, we optimized the ground state geometry of nile red using the MP2 method with a cc-pVDZ basis set. The optimized ground state geometry was then used to run both MP2 calculations and TDDFT calculations. In this case, continuum solvation was enabled and the calculations were repeated in three different solvents, water (wat, $\varepsilon_s = 80.1$, $\varepsilon_\infty = 1.777$), acetonitrile (acn, $\varepsilon_s = 37.5$, $\varepsilon_\infty = 1.807$), and cyclohexane (cyhex, $\varepsilon_s = 2.02$, $\varepsilon_\infty = 2.034$).

Solvent	**Model**	**Energy (Hartree)**
wat	ddPCM	-1026.5343
wat	PCM	-1026.5343
wat	ddCOSMO	-1026.5343
wat	COSMO	-1026.5344
acn	ddPCM	-1026.5340
acn	PCM	-1026.5341
acn	ddCOSMO	-1026.5342
acn	COSMO	-1026.5342
cyhex	ddPCM	-1026.5279
cyhex	PCM	—
cyhex	ddCOSMO	-1026.5290
cyhex	COSMO	-1026.5290
vac	—	-1026.5241

Table 4: Results of MP2 calculations on nile red, done in three solvents (water, acetonitrile and cyclohexane), with methods from PCMSolver (PCM, COSMO) and from ddX (ddPCM, ddCOSMO). The results of calculation done in vacuum are added as a reference.

The MP2 results are reported in table 4, whereas the TDDFT results are reported in table 5. The `ddX` results agree perfectly with the PCMSolver results, confirming the validity of the implementation. It was not possible to perform the PCMSolver calculations in cyclohexane due to numerical errors, and in this case, `ddX` showed its better stability. We did not report the timings, as all the investigated models performed similarly,

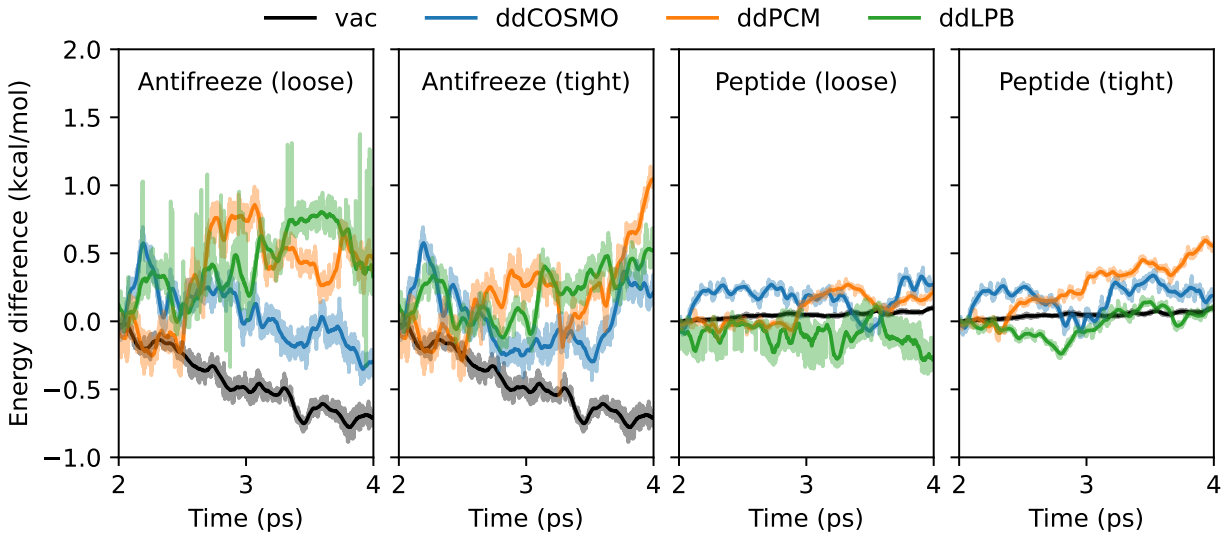

Figure 4: Energy conservation during the NVE trajectories, the values reported are the total energy minus the starting energy of the NVE simulation. The different models (vacuum, ddCOSMO, ddPCM and ddLPB) are reported using different colors, the combination of the two systems, and the two convergence thresholds (loose 10^{-4}, tight 10^{-8}) using different panels. The solid curves report a moving average of the energies over 100 steps to improve the readability, and the partially transparent curves report the energies and show the short-term fluctuations.

suggesting that the bottleneck of the calculations is not solving the linear systems but rather computing the electronic integrals at the grid points, an operation which is similar for both `ddX` and PCMSolver.

However, it should be noted that since `ddX` is entirely linear scaling, we can expect a molecule size at which the PCMSolver calculation becomes more expensive as the quadratic contribution for solving the linear system becomes dominant.

To conclude, the `ddX` implementation in Psi4 shows similar performance and improved stability with respect to traditional implementations while generating the same results. Further, for this size of the solute, there is no significant difference in the timings between ddCOSMO and ddPCM as opposed to PCMsolver. Overall, the Psi4-`ddX` interface is ready to be used in production calculations and its linear scaling allows us to consider much larger molecules than the nile red molecule.

5.4 ADC(2) Energies

The `adcc` support for `ddX` was tested by performing ADC(2) calculations on the p-nitroaniline molecule. This molecule is a push-pull system characterized by a bright absorption due to a charge transfer transition. The band is subject to a large solvatochromic red-shift when going from low to high polarity[109].

The initial structures were obtained by performing MP2 geometry optimizations with the cc-pVDZ basis set. The initial structure was first optimized in vacuum, and then the resulting structure was used as a starting point for three geometry optimizations in the investigated solvents. The solvation effects were included using ddPCM. As the analytical forces are not yet supported in Psi4/`ddX`, we used the automatic numerical differentiation provided by Psi4 to carry out the optimizations.

On the final four structures, we performed excited state calculations using ADC(2). The solvation effect on the excited states was included through a perturbative linear response correction[88].

The results are reported in table 6. The ground state energy (Hartree-Fock) progressively decreases with an increasing dielectric constant of the environment. The first and third excited states are dark and

Solvent	Model	Energy (Hartree)	Exc (eV)				
wat	ddPCM	-1032.4381	2.80	3.56	3.63	4.11	4.33
wat	PCM	-1032.4382	2.79	3.56	3.63	4.11	4.33
wat	ddCOSMO	-1032.4382	2.78	3.56	3.63	4.11	4.33
wat	COSMO	-1032.4382	2.78	3.56	3.63	4.11	4.33
acn	ddPCM	-1032.4379	2.80	3.56	3.64	4.11	4.33
acn	PCM	-1032.4380	2.79	3.56	3.63	4.11	4.33
acn	ddCOSMO	-1032.4380	2.78	3.56	3.63	4.11	4.33
acn	COSMO	-1032.4381	2.78	3.56	3.63	4.11	4.33
cyhex	ddPCM	-1032.4326	2.85	3.45	3.68	4.12	4.31
cyhex	PCM	—	—	—	—	—	—
cyhex	ddCOSMO	-1032.4334	2.82	3.46	3.68	4.12	4.30
cyhex	COSMO	-1032.4334	2.82	3.46	3.68	4.12	4.30
vac	—	-1032.4292	2.99	3.37	3.72	4.12	4.30

Table 5: Results of TDDFT calculations on nile red, done in three solvents (water, acetonitrile and cyclohexane), with methods from PCMSolver (PCM, COSMO) and from ddX (ddPCM, ddCOSMO). The results of calculation in vacuum are added as a reference.

Solvent	Energy (Hartree)	Exc (eV)		
wat	-489.2550	3.80	3.97	4.54
acn	-489.2547	3.80	3.98	4.54
cyhex	-489.2483	3.81	4.26	4.69
vac	-489.2436	3.76	4.53	4.72

Table 6: Results of ADC(2) calculations on p-nitroaniline, done in three solvents (water, acetonitrile and cyclohexane) and in vacuum.

are only slightly influenced by the different dielectric constants of the environment, whereas the second excited state is bright and undergoes the expected red-shift when going from a low polarity solvent to a high polarity solvent. The general trend is in agreement with experimental results[110–112], however, a quantitative agreement is missing mostly due to limitations of the continuum solvation models[109].

6 Conclusion and outlook

The `ddX` library represents the culmination of over ten years of dedicated effort in the field of PCSM. By combining three PCSMs based on domain decomposition, namely ddCOSMO, ddPCM, and ddLPB, and easy-to-use API in multiple programming languages, this library is a robust and efficient tool for treating solvation effects in a wide variety of cases and different host codes. Notably, the implemented methods are robust and exhibit a linear scaling regime in both time and memory, making them suitable for applications in both quantum chemistry and biophysics, thus bridging the gap between the two communities.

However, the current state of the library and its interfaces is only the starting point of a more ambitious project. There are multiple perspectives which we plan to explore, from interfacing the library to additional QM and MM packages, making the existing interfaces more rich in features, for example by implementing the forces in the Psi4/`ddX` interface, to finally further improving the numerical efficiency of the methods, by additional optimization and by porting the core algorithms to GPU.

Funding Information

AJ and BS acknowledge support from the German Research Foundation (DFG) under project 440641818. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 40/575-1 FUGG (JUSTUS 2 cluster).

A Matrix Formulation

In this appendix, we present the structure of the system of equations presented in Sec. 2.3 and Sec. 2.4; also, for the ease of notation, we denote $\mathbf{F}^\lambda - \mathbf{M}^\lambda \mathbf{X}$ by $\mathbf{H}$.

A.1 ddCOSMO

For ddCOSMO, the structure of the problem has the following form:

$$\mathbf{M} = \begin{pmatrix} \mathbf{L} & 0 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{X} = \begin{pmatrix} \overline{\mathbf{X}} \\ 0 \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} -\mathbf{\Phi} \\ 0 \end{pmatrix}, \qquad \text{and,} \qquad \mathbf{\Psi} = f(\varepsilon_s) \begin{pmatrix} \Psi \\ 0 \end{pmatrix},$$

where $\mathbf{\Phi}$ related to the discretization of the function Φ.

Note that only the first component is needed for the ddCOSMO. The notation used in the ddCOSMO literature, e.g., [60, 61, 84], uses g to denote $\mathbf{F}$. Here, the matrix $\mathbf{L}$ is of size $M(\ell_{\max}+1)^2 \times M(\ell_{\max}+1)^2$ and the vectors $\overline{\mathbf{X}}, \mathbf{\Phi}$, and Ψ are of size $M(\ell_{\max}+1)^2$, where $\ell_{\max}$ is the maximum number of spherical harmonics. The references [57, 60] also give the entries of the matrix $\mathbf{L}$ and the vectors $\mathbf{\Phi}$, Ψ as well as the scalar function $f(\varepsilon_s)$.

For the adjoint system,

$$\mathbf{M}^* = \begin{pmatrix} \mathbf{L}^* & 0 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{X}_{\text{adj}} = \begin{pmatrix} \overline{\mathbf{X}}_{\text{adj}} \\ 0 \end{pmatrix}, \qquad \mathbf{H} = \begin{pmatrix} -\mathbf{\Phi}^\lambda - \mathbf{L}^\lambda \overline{\mathbf{X}} \\ 0 \end{pmatrix}.$$

The term $\langle \mathbf{X}_{\text{adj}}, \mathbf{H} \rangle$ then writes

$$\langle \mathbf{X}_{\text{adj}}, \mathbf{H} \rangle = -\left\langle \overline{\mathbf{X}}_{\text{adj}}, \mathbf{\Phi}^\lambda \right\rangle - \left\langle \overline{\mathbf{X}}_{\text{adj}}, \mathbf{L}^\lambda \overline{\mathbf{X}} \right\rangle.$$

A.2 ddPCM

For ddPCM, the structure of the problem has the form

$$\mathbf{M} = \begin{pmatrix} \mathbf{L} & \mathbb{I} \\ 0 & \mathbf{R}_\varepsilon \end{pmatrix}, \qquad \mathbf{X} = \begin{pmatrix} \overline{\mathbf{X}} \\ \mathbf{\Phi}_\varepsilon \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} 0 \\ \mathbf{R}_\infty \mathbf{\Phi} \end{pmatrix}, \qquad \text{and,} \qquad \mathbf{\Psi} = \begin{pmatrix} \Psi \\ 0 \end{pmatrix},$$

where $\mathbb{I}$ denotes the identity matrix of the corresponding size. The notation used in the references [64, 65] uses σ to denote $\overline{\mathbf{X}}$. The references [64, 68] also give the definition of the matrices $\mathbf{R}_\varepsilon$ and $\mathbf{R}_\infty$.

For the adjoint system

$$\mathbf{M}^* = \begin{pmatrix} \mathbf{L}^* & 0 \\ \mathbb{I} & \mathbf{R}_\varepsilon^* \end{pmatrix}, \qquad \mathbf{X}_{\text{adj}} = \begin{pmatrix} \overline{\mathbf{X}}_{\text{adj}} \\ -\mathbf{Y} \end{pmatrix}, \qquad \mathbf{H} = \begin{pmatrix} -\mathbf{L}^\lambda \overline{\mathbf{X}} \\ \mathbf{R}^\lambda(\mathbf{\Phi} - \mathbf{\Phi}_\varepsilon) + \mathbf{R}_\infty \mathbf{\Phi}^\lambda \end{pmatrix}.$$

The minus sign in front of $\mathbf{Y}$ appears for consistency with the notation introduced in [66].

Remark 1. *There holds the relation* $\overline{\mathbf{X}}_{\text{adj}} = \mathbf{R}_\varepsilon^* \mathbf{Y}$ *and since* $\mathbf{R}_\infty - \mathbf{R}_\varepsilon = -\frac{4\pi}{\varepsilon_s - 1}$ *is constant there holds* $\mathbf{R}^\lambda := \mathbf{R}_\varepsilon^\lambda = \mathbf{R}_\infty^\lambda$. *Hence, it is advantageous to write*

$$\langle \mathbf{X}_{\text{adj}}, \mathbf{H} \rangle = -\left\langle \overline{\mathbf{X}}_{\text{adj}}, \mathbf{L}^\lambda \overline{\mathbf{X}} \right\rangle + \left\langle \mathbf{Y}, \mathbf{R}^\lambda(\mathbf{\Phi}_\varepsilon - \mathbf{\Phi}) \right\rangle - \left\langle \mathbf{Q}, \mathbf{\Phi}^\lambda \right\rangle \qquad \mathbf{Q} := \overline{\mathbf{X}}_{\text{adj}} - \frac{4\pi}{\varepsilon_s - 1} \mathbf{Y}.$$

A.3 ddLPB

Finally, ddLPB has the following structure

$$\mathbf{M} = \begin{pmatrix} \mathbf{A} & 0 \\ 0 & \mathbf{B} \end{pmatrix} + \mathbf{C}, \qquad \mathbf{C} = \begin{pmatrix} \mathbf{C}_1 & \mathbf{C}_2 \\ \mathbf{C}_1 & \mathbf{C}_2 \end{pmatrix}, \qquad \mathbf{X} = \begin{pmatrix} \overline{\mathbf{X}} \\ \mathbf{X}_\mathrm{e} \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} \mathbf{F}_0 - \mathbf{\Phi} \\ \mathbf{F}_0 \end{pmatrix}, \qquad \text{and,} \qquad \mathbf{\Psi} = \begin{pmatrix} \Psi \\ 0 \end{pmatrix}.$$

The notation used in the ddLPB literature, see e.g. [67, 69], uses $\mathbf{X}_\mathrm{r}$ to denote $\overline{\mathbf{X}}$ and $\mathbf{G}_0$ to denote $-\mathbf{\Phi}$. These references also provide the entries of the matrices $\mathbf{A}$ (which is a scaled version of $\mathbf{L}$), $\mathbf{B}$, $\mathbf{C}_1$, and $\mathbf{C}_2$ as well as the vectors $\mathbf{F}_0$, and $\mathbf{G}_0 (= -\mathbf{\Phi})$.

Finally, derivatives of the ddLPB-energy can be cast as

$$\mathbf{M}^* = \begin{pmatrix} \mathbf{A}^* & 0 \\ 0 & \mathbf{B}^* \end{pmatrix} + \mathbf{C}^*, \qquad \mathbf{C}^* = \begin{pmatrix} \mathbf{C}_1^* & \mathbf{C}_1^* \\ \mathbf{C}_2^* & \mathbf{C}_2^* \end{pmatrix}, \qquad \mathbf{X}_\mathrm{adj} = \begin{pmatrix} \overline{\mathbf{X}}_\mathrm{adj} \\ \mathbf{X}_\mathrm{adj,e} \end{pmatrix}, \qquad \mathbf{H} = \begin{pmatrix} \mathbf{F}_0^\lambda - \mathbf{\Phi}^\lambda - \mathbf{A}^\lambda \overline{\mathbf{X}} - \mathbf{C}_1^\lambda \overline{\mathbf{X}} - \mathbf{C}_2^\lambda \mathbf{X}_\mathrm{e} \\ \mathbf{F}_0^\lambda - \mathbf{B}^\lambda \mathbf{X}_\mathrm{e} - \mathbf{C}_1^\lambda \overline{\mathbf{X}} - \mathbf{C}_2^\lambda \mathbf{X}_\mathrm{e} \end{pmatrix}.$$

The term $\langle \mathbf{X}_\mathrm{adj}, \mathbf{H} \rangle$ then writes

$$\langle \mathbf{X}_\mathrm{adj}, \mathbf{H} \rangle = \left\langle \overline{\mathbf{X}}_\mathrm{adj} \ , \mathbf{F}_0^\lambda - \mathbf{\Phi}^\lambda \right\rangle + \left\langle \overline{\mathbf{X}}_\mathrm{adj,e}, \mathbf{F}_0^\lambda \right\rangle - \left\langle \overline{\mathbf{X}}_\mathrm{adj}, \mathbf{A}^\lambda \overline{\mathbf{X}} \right\rangle - \left\langle \overline{\mathbf{X}}_\mathrm{adj} \ , \mathbf{B}^\lambda \mathbf{X}_\mathrm{e} \right\rangle - \left\langle \mathbf{X}_\mathrm{adj}, \mathbf{C}^\lambda \mathbf{X} \right\rangle.$$

References

[1] Christopher J. Cramer and Donald G. Truhlar. "Implicit Solvation Models: Equilibria, Structure, Spectra, and Dynamics". In: *Chemical Reviews* 99.8 (1999), pp. 2161–2200. DOI: 10.1021/cr960149m.

[2] Jacopo Tomasi, Benedetta Mennucci, and Roberto Cammi. "Quantum Mechanical Continuum Solvation Models". In: *Chemical Reviews* 105.8 (2005), pp. 2999–3094. DOI: 10.1021/cr9904009.

[3] Adrian W. Lange and John M. Herbert. "Polarizable Continuum Reaction-Field Solvation Models Affording Smooth Potential Energy Surfaces". In: *The Journal of Physical Chemistry Letters* 1.2 (2009), pp. 556–561. DOI: 10.1021/jz900282c.

[4] Benedetta Mennucci. "Polarizable continuum model". In: *WIREs Computational Molecular Science* 2.3 (2012), pp. 386–404. DOI: 10.1002/wcms.1086.

[5] Andreas Klamt. "The COSMO and COSMO-RS solvation models". In: *WIREs Computational Molecular Science* 8.1 (2017). DOI: 10.1002/wcms.1338.

[6] John M. Herbert. "Dielectric continuum methods for quantum chemistry". In: *WIREs Computational Molecular Science* 11.4 (2021). DOI: 10.1002/wcms.1519.

[7] Benoît Roux and Thomas Simonson. "Implicit solvent models". In: *Biophysical Chemistry* 78.1–2 (Apr. 1999), pp. 1–20. ISSN: 0301-4622. DOI: 10.1016/s0301-4622(98)00226-9.

[8] Omar Demerdash, Eng-Hui Yap, and Teresa Head-Gordon. "Advanced Potential Energy Surfaces for Condensed Phase Simulation". In: *Annual Review of Physical Chemistry* 65.1 (2014), pp. 149–174. DOI: 10.1146/annurev-physchem-040412-110040.

[9] Sergio Decherchi et al. "Implicit solvent methods for free energy estimation". In: *European Journal of Medicinal Chemistry* 91 (Feb. 2015), pp. 27–42. ISSN: 0223-5234. DOI: 10.1016/j.ejmech.2014.08.064.

[10] B. Mennucci, E. Cancès, and J. Tomasi. "Evaluation of Solvent Effects in Isotropic and Anisotropic Dielectrics and in Ionic Solutions with a Unified Integral Equation Method: theoretical Bases, Computational Implementation, and Numerical Applications". In: *The Journal of Physical Chemistry B* 101.49 (1997), pp. 10506–10517. DOI: 10.1021/jp971959k.

[11] J. Tomasi, B. Mennucci, and E. Cancès. "The IEF version of the PCM solvation method: an overview of a new method addressed to study molecular solutes at the QM ab initio level". In: *Journal of Molecular Structure: THEOCHEM* 464.1–3 (1999), pp. 211–226. DOI: 10.1016/s0166-1280(98)00553-3.

[12] E. Cancès, B. Mennucci, and J. Tomasi. "A new integral equation formalism for the polarizable continuum model: Theoretical background and applications to isotropic and anisotropic dielectrics". In: *The Journal of Chemical Physics* 107.8 (1997), pp. 3032–3041. DOI: 10.1063/1.474659.

[13] Andreas Klamt. "The COSMO and COSMO-RS solvation models". In: *WIREs Computational Molecular Science* 1.5 (2011), pp. 699–709. DOI: 10.1002/wcms.56.

[14] A. Klamt and G. Schüürmann. "COSMO: a new approach to dielectric screening in solvents with explicit expressions for the screening energy and its gradient". In: *J. Chem. Soc., Perkin Trans. 2* 5 (1993), pp. 799–805. DOI: 10.1039/p29930000799.

[15] Thanh N. Truong and Eugene V. Stefanovich. "A new method for incorporating solvent effect into the classical, ab initio molecular orbital and density functional theory frameworks for arbitrary shape cavity". In: *Chemical Physics Letters* 240.4 (1995), pp. 253–260. DOI: 10.1016/0009-2614(95)00541-b.

[16] Vincenzo Barone and Maurizio Cossi. "Quantum Calculation of Molecular Energies and Energy Gradients in Solution by a Conductor Solvent Model". In: *The Journal of Physical Chemistry A* 102.11 (1998), pp. 1995–2001. DOI: 10.1021/jp9716997.

[17] Brent A. Gregersen et al. "Smooth Solvation Method for d-Orbital Semiempirical Calculations of Biological Reactions. 2. Application to Transphosphorylation Thio Effects in Solution". In: *The Journal of Physical Chemistry B* 109.19 (Apr. 2005), pp. 9810–9817. ISSN: 1520-5207. DOI: 10.1021/jp0440611.

[18] Tobias Benighaus and Walter Thiel. "Efficiency and Accuracy of the Generalized Solvent Boundary Potential for Hybrid QM/MM Simulations: Implementation for Semiempirical Hamiltonians". In: *Journal of Chemical Theory and Computation* 4.10 (2008), pp. 1600–1609. DOI: 10.1021/ct800193a.

[19] Tobias Benighaus and Walter Thiel. "A General Boundary Potential for Hybrid QM/MM Simulations of Solvated Biomolecular Systems". In: *Journal of Chemical Theory and Computation* 5.11 (2009), pp. 3114–3128. DOI: 10.1021/ct900437b.

[20] Kristian Kříž and Jan Řezáč. "Reparametrization of the COSMO Solvent Model for Semiempirical Methods PM6 and PM7". In: *Journal of Chemical Information and Modeling* 59.1 (2019), pp. 229–235. DOI: 10.1021/acs.jcim.8b00681.

[21] M.A. Aguilar, F.J.Olivares del Valle, and J. Tomasi. "Electron correlation and solvation effects. II. The description of the vibrational properties of a water molecule in a dielectric given by the application of the polarizable continuum model with inclusion of correlation effects". In: *Chemical Physics* 150.2 (1991), pp. 151–161. DOI: 10.1016/0301-0104(91)80125-2.

[22] F.J.Olivares del Valle and J. Tomasi. "Electron correlation and solvation effects. I. Basic formulation and preliminary attempt to include the electron correlation in the quantum mechanical polarizable continuum model so as to study solvation phenomena". In: *Chemical Physics* 150.2 (1991), pp. 139–150. DOI: 10.1016/0301-0104(91)80124-z.

[23] Filippo Lipparini, Giovanni Scalmani, and Benedetta Mennucci. "Non covalent interactions in RNA and DNA base pairs: a quantum-mechanical study of the coupling between solvent and electronic density". In: *Physical Chemistry Chemical Physics* 11.48 (2009), p. 11617. DOI: 10.1039/b915898g.

[24] R. Cammi. "Quantum cluster theory for the polarizable continuum model. I. The CCSD level with analytical first and second derivatives". In: *The Journal of Chemical Physics* 131.16 (2009). DOI: 10.1063/1.3245400.

[25] Marco Caricato. "Coupled cluster theory with the polarizable continuum model of solvation". In: *International Journal of Quantum Chemistry* 119.1 (2018). DOI: 10.1002/qua.25710.

[26] Giovanni Scalmani et al. "Achieving linear-scaling computational cost for the polarizable continuum model of solvation". In: *Theoretical Chemistry Accounts* 111.2–6 (2004), pp. 90–100. DOI: 10.1007/s00214-003-0527-2.

[27] M. Gouy. "Sur la constitution de la charge électrique à la surface d'un électrolyte". In: *Journal de Physique Théorique et Appliquée* 9.1 (1910), pp. 457–468. DOI: 10.1051/jphystap:019100090045700.

[28] David Leonard Chapman. "LI. A contribution to the theory of electrocapillarity". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 25.148 (1913), pp. 475–481. DOI: 10.1080/14786440408634187.

[29] P. Debye and E. Hückel. "Zur Theorie der Elektrolyte. I. Gefrierpunktserniedrigung und verwandte Erscheinungen". In: *Physikalische Z.* 24.185 (1923), p. 305.

[30] Michael K. Gilson et al. "On the calculation of electrostatic interactions in proteins". In: *Journal of Molecular Biology* 184.3 (Aug. 1985), pp. 503–516. ISSN: 0022-2836. DOI: 10.1016/0022-2836(85)90297-9.

[31] Nathan A. Baker et al. "Electrostatics of nanosystems: Application to microtubules and the ribosome". In: *Proceedings of the National Academy of Sciences* 98.18 (2001), pp. 10037–10041. DOI: 10.1073/pnas.181342398.

[32] Elizabeth Jurrus et al. "Improvements to the APBS biomolecular solvation software suite". In: *Protein Science* 27.1 (2017), pp. 112–128. DOI: 10.1002/pro.3280.

[33] Chuan Li et al. "DelPhi Suite: New Developments and Review of Functionalities". In: *Journal of Computational Chemistry* 40.28 (June 2019), pp. 2502–2508. ISSN: 1096-987X. DOI: 10.1002/jcc.26006.

[34] Stephen D. Bond et al. "A first-order system least-squares finite element method for the Poisson-Boltzmann equation". In: *Journal of Computational Chemistry* 31.8 (Nov. 2009), pp. 1625–1635. ISSN: 1096-987X. DOI: 10.1002/jcc.21446.

[35] M. Holst et al. "Adaptive Finite Element Modeling Techniques for the Poisson-Boltzmann Equation". In: *Communications in Computational Physics* 11.1 (Jan. 2012), pp. 179–214. ISSN: 1991-7120. DOI: 10.4208/cicp.081009.130611a.

[36] Dexuan Xie. "New solution decomposition and minimization schemes for Poisson–Boltzmann equation in calculation of biomolecular electrostatics". In: *Journal of Computational Physics* 275 (2014), pp. 294–309. DOI: 10.1016/j.jcp.2014.07.012.

[37] Benzhuo Lu et al. "Order N algorithm for computation of electrostatic interactions in biomolecular systems". In: *Proceedings of the National Academy of Sciences* 103.51 (Dec. 2006), pp. 19314–19319. ISSN: 1091-6490. DOI: 10.1073/pnas.0605166103.

[38] Chandrajit Bajaj, Shun-Chuan Chen, and Alexander Rand. "An Efficient Higher-Order Fast Multipole Boundary Element Solution for Poisson–Boltzmann-Based Molecular Electrostatics". In: *SIAM Journal on Scientific Computing* 33.2 (Jan. 2011), pp. 826–848. ISSN: 1095-7197. DOI: 10.1137/090764645.

[39] Christopher D. Cooper, Jaydeep P. Bardhan, and L.A. Barba. "A biomolecular electrostatics solver using Python, GPUs and boundary elements that can handle solvent-filled cavities and Stern layers". In: *Computer Physics Communications* 185.3 (Mar. 2014), pp. 720–729. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2013.10.028.

[40] Christopher D. Cooper. "A Boundary-Integral Approach for the Poisson–Boltzmann Equation with Polarizable Force Fields". In: *Journal of Computational Chemistry* 40.18 (2019), pp. 1680–1692. DOI: 10.1002/jcc.25820.

[41] Stefan D. Search, Christopher D. Cooper, and Elwin van't Wout. "Towards optimal boundary integral formulations of the Poisson-Boltzmann equation for molecular electrostatics". In: *Journal of Computational Chemistry* 43.10 (2022), pp. 674–691. DOI: 10.1002/jcc.26825.

[42] Ian Addison-Smith, Horacio V. Guzman, and Christopher D. Cooper. "Accurate Boundary Integral Formulations for the Calculation of Electrostatic Forces with an Implicit-Solvent Model". In: *Journal of Chemical Theory and Computation* 19.10 (2023), pp. 2996–3006. DOI: 10.1021/acs.jctc.3c00021.

[43] Weihua Geng and Robert Krasny. "A treecode-accelerated boundary integral Poisson–Boltzmann solver for electrostatics of solvated biomolecules". In: *Journal of Computational Physics* 247 (2013), pp. 62–78. DOI: 10.1016/j.jcp.2013.03.056.

[44] Robert Krasny and Lei Wang. "A treecode based on barycentric Hermite interpolation for electrostatic particle interactions". In: *Computational and Mathematical Biophysics* 7.1 (Jan. 2019), pp. 73–84. ISSN: 2544-7297. DOI: 10.1515/cmb-2019-0006.

[45] Leighton Wilson, Weihua Geng, and Robert Krasny. "TABI-PB 2.0: An Improved Version of the Treecode-Accelerated Boundary Integral Poisson-Boltzmann Solver". In: *The Journal of Physical Chemistry B* 126.37 (2022), pp. 7104–7113. DOI: 10.1021/acs.jpcb.2c04604.

[46] Itay Lotan and Teresa Head-Gordon. "An Analytical Electrostatic Model for Salt Screened Interactions between Multiple Proteins". In: *Journal of Chemical Theory and Computation* 2.3 (2006), pp. 541–555. DOI: 10.1021/ct050263p.

[47] Eng-Hui Yap and Teresa Head-Gordon. "Calculating the Bimolecular Rate of Protein–Protein Association with Interacting Crowders". In: *Journal of Chemical Theory and Computation* 9.5 (2013), pp. 2481–2489. DOI: 10.1021/ct400048q.

[48] Nadia Rega, Giuseppe Brancato, and Vincenzo Barone. "Non-periodic boundary conditions for ab initio molecular dynamics in condensed phase using localized basis functions". In: *Chemical Physics Letters* 422.4–6 (2006), pp. 367–371. DOI: 10.1016/j.cplett.2006.02.051.

[49] Giuseppe Brancato, Vincenzo Barone, and Nadia Rega. "Theoretical modeling of spectroscopic properties of molecules in solution: toward an effective dynamical discrete/continuum approach". In: *Theoretical Chemistry Accounts* 117.5–6 (2007), pp. 1001–1015. DOI: 10.1007/s00214-006-0216-z.

[50] Giuseppe Brancato, Nadia Rega, and Vincenzo Barone. "A hybrid explicit/implicit solvation method for first-principle molecular dynamics simulations". In: *The Journal of Chemical Physics* 128.14 (2008). DOI: 10.1063/1.2897759.

[51] Giordano Mancini et al. "Molecular Dynamics Simulations Enforcing Nonperiodic Boundary Conditions: New Developments and Application to the Solvent Shifts of Nitroxide Magnetic Parameters". In: *Journal of Chemical Theory and Computation* 18.4 (2022), pp. 2479–2493. DOI: 10.1021/acs.jctc.2c00046.

[52] Arnfinn Hykkerud Steindal et al. "Excitation Energies in Solution: The Fully Polarizable QM/MM/PCM Method". In: *The Journal of Physical Chemistry B* 115.12 (2011), pp. 3027–3037. DOI: 10.1021/jp1101913.

[53] Stefano Caprasecca, Carles Curutchet, and Benedetta Mennucci. "Toward a Unified Modeling of Environment and Bridge-Mediated Contributions to Electronic Energy Transfer: A Fully Polarizable QM/MM/PCM Approach". In: *Journal of Chemical Theory and Computation* 8.11 (2012), pp. 4462–4473. DOI: 10.1021/ct300620w.

[54] Stefano Caprasecca et al. "Achieving Linear Scaling in Computational Cost for a Fully Polarizable MM/Continuum Embedding". In: *Journal of Chemical Theory and Computation* 11.2 (2015), pp. 694–704. DOI: 10.1021/ct501087m.

[55] Filippo Lipparini et al. "Quantum Calculations in Solution for Large to Very Large Molecules: A New Linear Scaling QM/Continuum Approach". In: *The Journal of Physical Chemistry Letters* 5.6 (2014), pp. 953–958. DOI: 10.1021/jz5002506.

[56] Franco Egidi, Ivan Carnimeo, and Chiara Cappelli. "Optical rotatory dispersion of methyloxirane in aqueous solution: assessing the performance of density functional theory in combination with a fully polarizable QM/MM/PCM approach". In: *Optical Materials Express* 5.1 (2014), p. 196. DOI: 10.1364/ome.5.000196.

[57] Michele Nottoli et al. "Energy, Structures, and Response Properties with a Fully Coupled QM/AMOEBA/ddCOSMO Implementation". In: *Journal of Chemical Theory and Computation* 17.9 (2021), pp. 5661–5672. DOI: 10.1021/acs.jctc.1c00555.

[58] Filippo Lipparini et al. "Polarizable Molecular Dynamics in a Polarizable Continuum Solvent". In: *Journal of Chemical Theory and Computation* 11.2 (2015), pp. 623–634. DOI: 10.1021/ct500998q.

[59] Michael J. Schnieders et al. "Polarizable atomic multipole solutes in a Poisson-Boltzmann continuum". In: *The Journal of Chemical Physics* 126.12 (2007). DOI: 10.1063/1.2714528.

[60] Eric Cancès, Yvon Maday, and Benjamin Stamm. "Domain decomposition for implicit solvation models". In: *The Journal of Chemical Physics* 139.5 (2013), p. 054111. DOI: 10.1063/1.4816767.

[61] Filippo Lipparini et al. "Fast Domain Decomposition Algorithm for Continuum Solvation Models: Energy and First Derivatives". In: *Journal of Chemical Theory and Computation* 9.8 (2013), pp. 3637–3648. DOI: 10.1021/ct400280b.

[62] Michele Nottoli et al. "Coarse-Graining ddCOSMO through an Interface between Tinker and the ddX Library". In: *The Journal of Physical Chemistry B* 126.43 (Oct. 2022), pp. 8827–8837. ISSN: 1520-5207. DOI: 10.1021/acs.jpcb.2c04579.

[63] Filippo Lipparini et al. "Quantum Calculations in Solution for Large to Very Large Molecules: A New Linear Scaling QM/Continuum Approach". In: *The Journal of Physical Chemistry Letters* 5.6 (Feb. 2014), pp. 953–958. ISSN: 1948-7185. DOI: 10.1021/jz5002506.

[64] Benjamin Stamm et al. "A new discretization for the polarizable continuum model within the domain decomposition paradigm". In: *The Journal of Chemical Physics* 144.5 (2016), p. 054101. DOI: 10.1063/1.4940136.

[65] Paolo Gatto, Filippo Lipparini, and Benjamin Stamm. "Computation of forces arising from the polarizable continuum model within the domain-decomposition paradigm". In: *The Journal of Chemical Physics* 147.22 (2017), p. 224108. DOI: 10.1063/1.5008329.

[66] Michele Nottoli et al. "Quantum Calculations in Solution of Energies, Structures, and Properties with a Domain Decomposition Polarizable Continuum Model". In: *Journal of Chemical Theory and Computation* 15.11 (2019), pp. 6061–6073. DOI: 10.1021/acs.jctc.9b00640.

[67] Chaoyu Quan, Benjamin Stamm, and Yvon Maday. "A Domain Decomposition Method for the Poisson–Boltzmann Solvation Models". In: *SIAM Journal on Scientific Computing* 41.2 (2019), B320–B350. DOI: 10.1137/18m119553x.

[68] A. Mikhalev, M. Nottoli, and B. Stamm. "Linearly scaling computation of ddPCM solvation energy and forces using the fast multipole method". In: *The Journal of Chemical Physics* 157.11 (2022), p. 114103. DOI: 10.1063/5.0104536.

[69] Abhinav Jha et al. "Linear scaling computation of forces for the domain-decomposition linear Poisson–Boltzmann method". In: *The Journal of Chemical Physics* 158.10 (2023). DOI: 10.1063/5.0141025.

[70] Daniel G. A. Smith et al. "PSI4 1.4: Open-source software for high-throughput quantum chemistry". In: *The Journal of Chemical Physics* 152.18 (2020), p. 184108. DOI: 10.1063/5.0006002.

[71] Michael F. Herbst et al. "adcc: A versatile toolkit for rapid development of algebraic-diagrammatic construction methods". In: *WIREs Computational Molecular Science* 10.6 (2020). DOI: 10.1002/wcms.1462.

[72] Joshua A. Rackers et al. "Tinker 8: Software Tools for Molecular Design". In: *Journal of Chemical Theory and Computation* 14.10 (2018), pp. 5273–5289. DOI: 10.1021/acs.jctc.8b00529.

[73] B. Lee and F.M. Richards. "The interpretation of protein structures: Estimation of static accessibility". In: *Journal of Molecular Biology* 55.3 (1971), 379–IN4. DOI: 10.1016/0022-2836(71)90324-x.

[74] Frederic M. Richards. "AREAS, VOLUMES, PACKING, AND PROTEIN STRUCTURE". In: *Annual Review of Biophysics and Bioengineering* 6.1 (1977), pp. 151–176. DOI: 10.1146/annurev.bb.06.060177.001055.

[75] Chaoyu Quan, Benjamin Stamm, and Yvon Maday. "A domain decomposition method for the polarizable continuum model based on the solvent excluded surface". In: *Mathematical Models and Methods in Applied Sciences* 28.07 (June 2018), pp. 1233–1266. ISSN: 1793-6314. DOI: 10.1142/s0218202518500331.

[76] A. Jha and B. Stamm. "Domain decomposition method for Poisson–Boltzmann equations based on Solvent Excluded Surface". In: *arXiv preprint arXiv:2309.06862* (2023).

[77] Eric Cancès and Benedetta Mennucci. "The escaped charge problem in solvation continuum models". In: *The Journal of Chemical Physics* 115.13 (2001), pp. 6130–6135. DOI: 10.1063/1.1401157.

[78] Daniel M. Chipman. "Simulation of volume polarization in reaction field theory". In: *The Journal of Chemical Physics* 110.16 (1999), pp. 8012–8018. DOI: 10.1063/1.478729.

[79] Daniel M. Chipman. "New formulation and implementation for volume polarization in dielectric continuum theory". In: *The Journal of Chemical Physics* 124.22 (2006), p. 224111. DOI: 10.1063/1.2203068.

[80] Otto Stern. "ZUR THEORIE DER ELEKTROLYTISCHEN DOPPELSCHICHT". In: *Zeitschrift für Elektrochemie und angewandte physikalische Chemie* 30.21–22 (1924), pp. 508–516. DOI: 10.1002/bbpc.192400182.

[81] L Greengard and V Rokhlin. "A fast algorithm for particle simulations". In: *Journal of Computational Physics* 73.2 (1987), pp. 325–348. DOI: 10.1016/0021-9991(87)90140-9.

[82] Alex Pothen. "Graph Partitioning Algorithms with Applications to Scientific Computing". In: *Parallel Numerical Algorithms*. Springer Netherlands, 1997, pp. 323–368. ISBN: 9789401154123. DOI: 10.1007/978-94-011-5412-3_12.

[83] R. Leland and B. Hendrickson. "An empirical study of static load balancing algorithms". In: *Proceedings of IEEE Scalable High Performance Computing Conference*. SHPCC-94. IEEE Comput. Soc. Press. DOI: 10.1109/shpcc.1994.296707.

[84] Benjamin Stamm et al. "How to make continuum solvation incredibly fast in a few simple steps: A practical guide to the domain decomposition paradigm for the conductor-like screening model". In: *International Journal of Quantum Chemistry* 119.1 (2018), e25669. DOI: 10.1002/qua.25669.

[85] R. Cammi et al. "Electronic excitation energies of molecules in solution: State specific and linear response methods for nonequilibrium continuum solvation models". In: *The Journal of Chemical Physics* 122.10 (2005). DOI: 10.1063/1.1867373.

[86] Jochen Schirmer. "Beyond the random-phase approximation: A new approximation scheme for the polarization propagator". In: *Physical Review A* 26.5 (1982), pp. 2395–2416. DOI: 10.1103/physreva.26.2395.

[87] Andreas Dreuw, Antonia Papapostolou, and Adrian L. Dempwolff. "Algebraic Diagrammatic Construction Schemes Employing the Intermediate State Formalism: Theory, Capabilities, and Interpretation". In: *The Journal of Physical Chemistry A* 127.32 (2023), pp. 6635–6646. DOI: 10.1021/acs.jpca.3c02761.

[88] Maximilian Scheurer et al. "Polarizable Embedding Combined with the Algebraic Diagrammatic Construction: Tackling Excited States in Biomolecular Systems". In: *Journal of Chemical Theory and Computation* 14.9 (2018), pp. 4870–4883. DOI: 10.1021/acs.jctc.8b00576.

[89] Maximilian Scheurer et al. "CPPE: An Open-Source C++ and Python Library for Polarizable Embedding". In: *Journal of Chemical Theory and Computation* 15.11 (2019), pp. 6154–6163. DOI: 10.1021/acs.jctc.9b00758.

[90] Roberto Di Remigio et al. *PCMSolver/pcmsolver: Fluctuating charges MM*. PCMSolver, an open-source library for the polarizable continuum modelelectrostatic problem, written by R. Di Remigio,L. Frediani and contributors (see http://pcmsolver.readthedocs.io/). 2020. DOI: 10.5281/ZENODO.1156166.

[91] Bernd Lunkenheimer and Andreas Köhn. "Solvent Effects on Electronically Excited States Using the Conductor-Like Screening Model and the Second-Order Correlated Method ADC(2)". In: *Journal of Chemical Theory and Computation* 9.2 (2012), pp. 977–994. DOI: 10.1021/ct300763v.

[92] JUSTUS2 bwHPC (see https://wiki.bwhpc.de/e/JUSTUS2).

[93] Michele Nottoli et al. *Replication Data for: "ddX: Polarizable Continuum Solvation from Small Molecules to Proteins"*. Version V1. 2024. DOI: 10.18419/darus-4030.

[94] Junmei Wang, Piotr Cieplak, and Peter A. Kollman. "How well does a restrained electrostatic potential (RESP) model perform in calculating conformational energies of organic and biological molecules?" In: *Journal of Computational Chemistry* 21.12 (2000), pp. 1049–1074. DOI: 10.1002/1096-987x(200009)21:12<1049::aid-jcc3>3.0.co;2-f.

[95] Pamela R. Hall et al. "Characterization and NMR Solution Structure of a Novel Cyclic Pentapeptide Inhibitor of Pathogenic Hantaviruses". In: *Chemical Biology & Drug Design* 69.3 (2007), pp. 180–190. DOI: 10.1111/j.1747-0285.2007.00489.x.

[96] H Ozaki et al. "Molecular structure of the toxin domain of heat-stable enterotoxin produced by a pathogenic strain of Escherichia coli. A putative binding site for a binding protein on rat intestinal epithelial cell membranes." In: *Journal of Biological Chemistry* 266.9 (1991), pp. 5934–5941. DOI: 10.1016/s0021-9258(19)67688-x.

[97] Yingqi Xu et al. "Solution Structure of BmP02, a New Potassium Channel Blocker from the Venom of the Chinese ScorpionButhus martensiKarsch," in: *Biochemistry* 39.45 (2000), pp. 13669–13675. DOI: 10.1021/bi000860s.

[98] Michael W. Day et al. "X-ray crystal structures of the oxidized and reduced forms of the rubredoxin from the marine hyperthermophilic archaebacterium pyrococcus furiosus". In: *Protein Science* 1.11 (1992), pp. 1494–1507. DOI: 10.1002/pro.5560011111.

[99] Kaisheng Chen et al. "Atomically defined mechanism for proton transfer to a buried redox centre in a protein". In: *Nature* 405.6788 (2000), pp. 814–817. DOI: 10.1038/35015610.

[100] Joost P. Schanstra et al. "Kinetic Characterization and X-ray Structure of a Mutant of Haloalkane Dehalogenase with Higher Catalytic Activity and Modified Substrate Range," in: *Biochemistry* 35.40 (1996), pp. 13186–13195. DOI: 10.1021/bi961151a.

[101] Ingrid Dreveny et al. "The Hydroxynitrile Lyase from Almond". In: *Structure* 9.9 (2001), pp. 803–815. DOI: 10.1016/s0969-2126(01)00639-6.

[102] Karin Valegård et al. "Structural and functional analyses of Rubisco from arctic diatom species reveal unusual posttranslational modifications". In: *Journal of Biological Chemistry* 293.34 (2018), pp. 13033–13043. DOI: 10.1074/jbc.ra118.003518.

[103] Nenad Ban and Alexander McPherson. "The structure of satellite panicum mosaic virus at 1.9 Å resolution". In: *Nature Structural Biology* 2.10 (1995), pp. 882–890. DOI: 10.1038/nsb1095-882.

[104] John Gehrmann et al. "Solution Structure of α-Conotoxin ImI by1H Nuclear Magnetic Resonance". In: *Journal of Medicinal Chemistry* 42.13 (1999), pp. 2364–2372. DOI: 10.1021/jm990114p.

[105] Tzu-Ping Ko et al. "The Refined Crystal Structure of an Eel Pout Type III Antifreeze Protein RD1 at 0.62-Å Resolution Reveals Structural Microheterogeneity of Protein and Solvation". In: *Biophysical Journal* 84.2 (2003), pp. 1228–1237. DOI: 10.1016/s0006-3495(03)74938-8.

[106] Arnold Reusken and Benjamin Stamm. "Analysis of the Schwarz Domain Decomposition Method for the Conductor-like Screening Continuum Model". In: *SIAM Journal on Numerical Analysis* 59.2 (2021), pp. 769–796. DOI: 10.1137/20m1342872.

[107] Gabriele Ciaramella, Muhammad Hassan, and Benjamin Stamm. "On the Scalability of the Schwarz Method". In: *The SMAI journal of computational mathematics* 6 (Apr. 2020), pp. 33–68. ISSN: 2426-8399. DOI: 10.5802/smai-jcm.61.

[108] Viktor Hornak et al. "Comparison of multiple Amber force fields and development of improved protein backbone parameters". In: *Proteins: Structure, Function, and Bioinformatics* 65.3 (2006), pp. 712–725. DOI: 10.1002/prot.21123.

[109] Matteo Ambrosetti et al. "Quantum Mechanics/Fluctuating Charge Protocol to Compute Solvatochromic Shifts". In: *Journal of Chemical Theory and Computation* 17.11 (2021), pp. 7146–7156. DOI: 10.1021/acs.jctc.1c00763.

[110] S.A. Kovalenko et al. "Femtosecond relaxation of photoexcited para-nitroaniline: solvation, charge transfer, internal conversion and cooling". In: *Chemical Physics Letters* 323.3–4 (2000), pp. 312–322. DOI: 10.1016/s0009-2614(00)00432-2.

[111] M. Stähelin, D.M. Burland, and J.E. Rice. "Solvent dependence of the second order hyperpolarizability in p-nitroaniline". In: *Chemical Physics Letters* 191.3–4 (1992), pp. 245–250. DOI: 10.1016/0009-2614(92)85295-l.

[112] S. Millefiori et al. "Electronic spectra and structure of nitroanilines". In: *Spectrochimica Acta Part A: Molecular Spectroscopy* 33.1 (1977), pp. 21–27. DOI: 10.1016/0584-8539(77)80143-8.