Received: 7 May 2021 | Revised: 10 June 2021 | Accepted: 16 June 2021
DOI: 10.1002/jcc.26711

SOFTWARE NOTE

Journal of COMPUTATIONAL CHEMISTRY WILEY

PyUNIxMD: A Python-based excited state molecular dynamics package

In Seong Lee | Jong-Kwon Ha | Daeho Han | Tae In Kim | Sung Wook Moon | Seung Kyu Min

Department of Chemistry, Ulsan National Institute of Science and Technology (UNIST), Ulsan, South Korea

Correspondence
Seung Kyu Min, Department of Chemistry, Ulsan National Institute of Science and Technology (UNIST), 50 UNIST-gil, Ulju-gun, Ulsan 44919, South Korea.
Email: skmin@unist.ac.kr

Funding information
National Research Foundation of Korea, Ministry of Science and ICT, Grant/Award Numbers: NRF-2019R1A2C1007744, NRF-2019R1A4A1027934; Ulsan National Institute of Science and Technology, Grant/Award Number: 1.190123.01

Abstract

Theoretical/computational description of excited state molecular dynamics is nowadays a crucial tool for understanding light-matter interactions in many materials. Here we present an open-source Python-based nonadiabatic molecular dynamics program package, namely PyUNIxMD, to deal with mixed quantum-classical dynamics for correlated electron-nuclear propagation. The PyUNIxMD provides many interfaces for quantum chemical calculation methods with commercial and noncommercial ab initio and semiempirical quantum chemistry programs. In addition, the PyUNIxMD offers many nonadiabatic molecular dynamics algorithms such as fewest-switch surface hopping and its derivatives as well as decoherence-induced surface hopping based on the exact factorization (DISH-XF) and coupled-trajectory mixed quantum-classical dynamics (CTMQC) for general purposes. Detailed structures and flows of PyUNIxMD are explained for the further implementations by developers. We perform a nonadiabatic molecular dynamics simulation for a molecular motor system as a simple demonstration.

KEYWORDS
decoherence, exact factorization, mixed quantum-classical dynamics, nonadiabatic molecular dynamics

1 | INTRODUCTION

Nonadiabatic molecular dynamics (NAMD) is a theoretical/computational tool to describe excited state phenomena such as a photosynthesis,[1–4] photovoltaics,[5–8] vision process,[9–14] and photochemical reactions.[15–19] Especially, understanding molecular mechanisms in photosynthesis and photovoltaics is crucial for the development of future technology toward sustainable energy. One of the most popular concepts for NAMD simulation is the mixed quantum-classical (MQC) approach which treats electronic degrees of freedom quantum mechanically while nuclear degrees of freedom are governed by the classical Newtonian equation of motion.[20] The most standard nonadiabatic MQC methods are ab initio multiple spawning (AIMS),[21,22] Ehrenfest dynamics,[23] and Tully's fewest switches surface hopping (FSSH).[24] In particular, FSSH and Ehrenfest dynamics are by far the most popular algorithms due to their efficiency and simplicity. Moreover, these methods can be implemented readily into modern quantum chemistry and physics program packages.

Due to the importance of NAMD simulations, lots of program packages for NAMD have been developed.[25–31] Furthermore, there are many quantum mechanical (QM) programs that provide NAMD functionalities.[32–37] Among them, SHARC[25] and Newton-X[26] are the

In Seong Lee and Jong-Kwon Ha contributed equally to this study.

most dominant programs based on the FSSH algorithm and its derivatives. They provide multiple interfaces to QM programs and NAMD capabilities such as spin-orbit/laser couplings, absorption/emission spectra, and QM/MM simulations. Most of the NAMD programs are written in legacy programming languages such as C/C++ and Fortran due to the efficiency and the richness of scientific libraries. Nowadays, Python becomes the most popular language due to its simplicity, and lots of scientific libraries and environments are optimized for Python. Furthermore, since Python is suitable for object-oriented programming, modularization can be done directly.

From the perspective of chemical theory, FSSH and Ehrenfest dynamics have an important limitation, the so-called overcoherence problem, which is an intrinsic problem due to the independent trajectory approximation.[38] Ehrenfest dynamics propagates a classical nuclear trajectory on an averaged potential energy surface while the electronic state is evolved by electronic time-dependent Schrödinger equation (TDSE) along the classical nuclear trajectory. The incorrect account for the electron-nuclear correlation results in the overestimation of electronic coherence along the classical trajectory,[39] that is, finite off-diagonal elements in electronic density matrix, and independent nuclear trajectories do not bifurcate in phase space. On the other hand, FSSH uses a single Born-Oppenheimer (BO) potential energy surface in a stochastic way to reproduce the nuclear wave packet splitting properly. However, off-diagonal elements in electronic density matrix are finite since FSSH exploits the Ehrenfest electronic equation of motion. Thus, the separated nuclear wave packets are still in coherent, that is, the electronic density matrix as a function of nuclear degrees of freedom is not captured correctly, resulting in an overestimation of nuclear coherence.[40] Thus, many derivatives of FSSH have been developed so far to account for the correct electron-nuclear correlation for quantum (de)coherence.[41–48] Recently, based on the exact factorization framework,[49,50] the coupled-trajectory MQC (CTMQC) algorithm has been developed to account for quantum (de)coherence.[51–55] Subsequently, decoherence-induced surface hopping based on the exact factorization (DISH-XF)[48] has been developed for efficiency. Compared to Ehrenfest dynamics and FSSH, the methods for quantum (de)coherence attempt to describe the nuclear nonadiabatic couplings,[40] that is, derivative coupling among nuclear wave functions, effectively in addition to electronic nonadiabatic couplings, that is, derivative couplings among electronic states. So far NAMD programs with DISH-XF and CTMQC for the general purpose have not been published yet.

In this article, we introduce a new open-source program package for NAMD simulations, namely PyUNIxMD, a Python-based universal excited state molecular dynamics which is aiming for easy-to-use, versatile MQC dynamics simulations including CTMQC and DISH-XF algorithms with interfaces for multiple QM programs. PyUNIxMD is mainly written in Python, with minimal interfaces to C codes via Cython for a time-consuming electronic propagation part. Since the codes are well-modularized as classes, it is straightforward to add other MQC algorithms or QM interfaces.

This article is organized as follows: First, we provide brief introductions for MQC methods implemented in PyUNIxMD. Next, we introduce the structure of PyUNIxMD package, a role of each class, and a workflow of an MQC simulation with PyUNIxMD. Finally, we show an actual NAMD simulation result with a molecular motor system as an example, and we summarize our work.

2 | METHODOLOGY

2.1 | Available mixed quantum-classical methods

The PyUNIxMD package employs multiple MQC approaches for correlated electron-nuclear dynamics simulation. In addition to Born-Oppenheimer molecular dynamics (BOMD), Ehrenfest dynamics,[23] and FSSH[24] with the decoherence corrections such as the instantaneous decoherence correction (IDC)[45] and the energy-based decoherence correction (EDC)[42] are implemented in the current version of PyUNIxMD. Especially, CTMQC[51–55] and DISH-XF[48] algorithms are included. In this section, we briefly review MQC methods implemented in PyUNIxMD.

2.1.1 | Ehrenfest dynamics

In the Ehrenfest dynamics, nuclei move according to the gradient of the mean potential while electrons propagate using the TDSE along the classical nuclear trajectory, $\underline{\underline{\mathbf{R}}}(t)=\{\mathbf{R}_\nu(t)\}$ where ν represents a nuclear index. The equations of motions are further simplified by expanding the electronic wave function $\Phi\left(\underline{\underline{\mathbf{r}}},t\right)$ with Born-Oppenheimer (BO) basis functions, $\Phi_k\left(\underline{\underline{\mathbf{r}}};\underline{\underline{\mathbf{R}}}(t)\right)$, that is, $\Phi\left(\underline{\underline{\mathbf{r}}},t\right)=\sum_k c_k(t)\Phi_k\left(\underline{\underline{\mathbf{r}}};\underline{\underline{\mathbf{R}}}(t)\right)$ where Φ_k is the k th eigenfunction of the BO Hamiltonian with the corresponding eigenvalue E_k, and $\underline{\underline{\mathbf{r}}}$ is the electronic degrees of freedom. As a result, the coupled motion of nuclei and electrons in the Ehrenfest dynamics is described by the following equations:

$$M_\nu\ddot{\mathbf{R}}_\nu(t)=-\sum_k c_k^*(t)c_k(t)\nabla_\nu E_k-\sum_{j,k} c_j^*(t)c_k(t)\left(E_k-E_j\right)\mathbf{d}_{jk,\nu} \tag{1}$$

$$\dot{c}_k(t)=-\frac{i}{\hbar}E_k c_k(t)-\sum_j\sum_\nu \mathbf{d}_{kj,\nu}\cdot\dot{\mathbf{R}}_\nu(t)c_j(t) \tag{2}$$

where M_ν is the mass of ν th nucleus, $\mathbf{d}_{kj,\nu}$ is the nonadiabatic coupling vectors (NACVs) as $\mathbf{d}_{kj,\nu}=\int d\underline{\underline{\mathbf{r}}}\Phi_k^*\nabla_\nu\Phi_j$. $\dot{\mathbf{R}}_\nu(t)$ and $\ddot{\mathbf{R}}_\nu(t)$ are the nuclear velocity and acceleration for the ν th nuclei, respectively. According to Equation (1), Ehrenfest dynamics requires NACVs intrinsically. Thus, Ehrenfest dynamics is only available with QM methods which can provide NACVs in PyUNIxMD. Equation (2) produces a coherent electronic state in presence of nonzero NACVs. Thus, an independent classical trajectory follows the mean-field potential according to Equation (1) preserving the coherent state even after the coupling region, which results in the overestimation of electronic coherence.

2.1.2 | Fewest switches surface hopping dynamics and decoherence corrections

While nuclei follow the averaged potential in Ehrenfest dynamics, nuclei in FSSH algorithm follow a single potential energy surface stochastically to describe nuclear branching phenomena after a nonadiabatic coupling region. While the electronic equation of motion is same as in the Ehrenfest dynamics (Equation (2)), nuclei propagate on a single potential energy surface of a given BO state (E_l), that is,

$$M_\nu \ddot{\mathbf{R}}_\nu(t) = -\nabla_\nu E_l(t) \tag{3}$$

where l is a "running" state for the given trajectory. The nuclear trajectory can choose the running state at any time depending on hopping probabilities where the hopping probability from l to another state k at time t with a time interval Δt is given as:

$$P_{l\to k}(t) = \max\left\{0, \frac{2\mathrm{Re}\left[\rho_{lk}(t)\sum_\nu \mathbf{d}_{lk,\nu}\cdot\dot{\mathbf{R}}_\nu(t)\right]}{\rho_{ll}(t)}\Delta t\right\}, \tag{4}$$

with an electron density matrix element $\rho_{lk}(t) = c_l^*(t)c_k(t)$. At every time step, a uniform random number ξ ranging from 0 to 1 is generated. Only if ξ satisfies the condition, $\sum_j^{k-1} P_{l\to j} < \xi < \sum_j^{k} P_{l\to j}$, the hop from l to k can occur. When a hop ($l \to k$) occurs during a FSSH simulation, we rescale the momentum or velocity of the nuclear trajectory to compensate a sudden change of potential energy for energy conservation, that is, $\sum_\nu \frac{1}{2}M_\nu\left|\dot{\mathbf{R}}_\nu\right|^2 + E_l(t) = \sum_\nu \frac{1}{2}M_\nu\left|\dot{\mathbf{R}}'_\nu\right|^2 + E_k(t)$. Various rescaling methods have been proposed to achieve the energy conservation.[56] In PyUNIxMD, three rescaling methods are implemented: (a) momentum rescaling along the NACV as $M_\nu\dot{\mathbf{R}}'_\nu(t) = M_\nu\dot{\mathbf{R}}_\nu(t) + \alpha\mathbf{d}_{lk,\nu}$ where α is the solution with a smaller absolute value of the equation $\alpha^2\sum_\nu \frac{\left|\mathbf{d}_{lk,\nu}\right|^2}{M_\nu} + \alpha\sum_\nu 2\dot{\mathbf{R}}_\nu\cdot\mathbf{d}_{lk,\nu} + 2(E_k(t) - E_l(t)) = 0$, (b) an isotropic rescaling of all atomic velocities as $\dot{\mathbf{R}}'_\nu(t) = \alpha\dot{\mathbf{R}}_\nu(t)$ with $\alpha = \sqrt{1 - \frac{E_k(t)-E_l(t)}{E_{kin}}}$, and (c) the combination of (a) and (b) where (a) has priority over (b). If we cannot determine a suitable α, the hop is rejected. In this case, the momentum can be either unchanged or reversed to the direction specified for the rescaling option above. Although the Equations (2) and (4) require NACVs, the calculation of the NACVs, which is computationally expensive, can be avoided by direct numerical evaluation of nonadiabatic coupling matrix elements (NACMEs), $\sigma_{ij} = \sum_\nu \mathbf{d}_{ij,\nu}\cdot\dot{\mathbf{R}}_\nu$.[57–59] Therefore, in contrast to the Ehrenfest dynamics, it is possible for FSSH to run NAMD simulations without NACVs. However, in this case, only the option (b) is available for the rescaling since $\mathbf{d}_{lk,\nu}$'s are not calculated. In addition, due to the stochastic nature of FSSH, multiple NAMD simulations should be performed with a suitable initial distribution of nuclear trajectories from Boltzmann sampling and Wigner sampling to obtain statistically meaningful results.

As mentioned earlier, FSSH suffers from the well-known overcoherence problem.[38] Many decoherence correction methods have been developed to solve the overcoherence problem. PyUNIxMD provides IDC,[45] EDC,[42] and DISH-XF.[48] Here we preserve DISH-XF for the latter section, and we discuss IDC and EDC approaches first. The IDC approach is the simplest decoherence correction scheme which resets the BO coefficient to 1 for the running state and 0 for the other states whenever the running state is changed or a hop is rejected. In the EDC method, instead of instantaneous collapse of the electronic wavefunction to the running state, exponential decay of the coherence is introduced via a decay-of-mixing lifetime based on BO energies, E_k, and a pre-determined parameter, C. The BO coefficients, $c_k(t)$, are scaled as $c'_k(t)$ by:

$$c'_k(t) = \left(\begin{array}{ll} c_k(t)e^{-\Delta t/\tau_k(t)} & \text{if } k \neq l \\ c_k(t)\left[\frac{1-\sum\limits_{j\neq l}\left|c'_j(t)\right|^2}{|c_k(t)|^2}\right]^{\frac{1}{2}} & \text{if } k = l \end{array}\right. \tag{5}$$

with the running state l, and the decoherence time, τ_k, is given by,

$$\tau_k(t) = \frac{\hbar}{|E_k(t) - E_l(t)|}\left(1 + \frac{C}{E_{kin}(t)}\right). \tag{6}$$

Here, E_{kin} is the nuclear kinetic energy and C is a constant parameter. In PyUNIxMD, we choose C as 0.1 Hartree by default.[42] Thus, IDC and EDC are rather an artificial correction for decoherence.

2.1.3 | Coupled-trajectory mixed quantum-classical dynamics

Based on the exact factorization, one can derive the coupled equations of motion for correlated electron-nuclear dynamics where the coupling is mediated by the so-called electron-nuclear correlation term.[49,50] Systematic approximations provide a MQC approach with coupled nuclear trajectories, namely the CTMQC method.[51–55] Here we provide working equations for CTMQC approach simply while the detailed derivations and theoretical explanations can be found in the original references.[51–53]

The coupled nuclear and electronic equations of motion are given as

$$\begin{aligned} M_\nu\ddot{\mathbf{R}}_\nu(t) = &-\sum_k \rho_{kk}(t)\nabla_\nu E_k - \sum_{j,k}\rho_{jk}(t)\left(E_k - E_j\right)\mathbf{d}_{jk,\nu} \\ &-\sum_j \rho_{jj}(t)\left(\sum_{\nu'}\frac{2}{\hbar M_{\nu'}}\mathcal{P}_{\nu'}(t)\cdot\mathbf{f}_{j,\nu'}(t)\right)\left[\sum_k \rho_{kk}(t)\mathbf{f}_{k,\nu}(t) - \mathbf{f}_{j,\nu}(t)\right], \end{aligned} \tag{7}$$

and

$$\begin{aligned} \dot{c}_k(t) &= -\frac{i}{\hbar}E_k c_k(t) - \sum_j\sum_\nu \sigma_{kj}(t)c_j(t) \\ &\quad -\sum_\nu \frac{\mathcal{P}_\nu(t)}{\hbar M_\nu}\cdot\left[\sum_j \rho_{jj}(t)\mathbf{f}_{j,\nu}(t) - \mathbf{f}_{k,\nu}(t)\right]c_k(t), \end{aligned} \tag{8}$$

respectively, where $\mathcal{P}_\nu(t) = -\left.\frac{\hbar\nabla_\nu|\chi\left(\underline{\underline{\mathbf{R}}},t\right)|}{|\chi\left(\underline{\underline{\mathbf{R}}},t\right)|}\right|_{\underline{\underline{\mathbf{R}}}(t)}$ is the so-called nuclear quantum momentum with a nuclear wave function $\chi\left(\underline{\underline{\mathbf{R}}},t\right)$, and $\mathbf{f}_{k,\nu}(t)$

is a phase term from the BO coefficient. Compared to Ehrenfest dynamics (Equations (1) and (2)), Equations (7) and (8) have additional terms with the quantum momentum $\mathcal{P}_\nu$. In the extreme case where the nuclear wave function is localized in a space as a delta function, the nuclear quantum momentum becomes zero, thus CTMQC equations collapse to the Ehrenfest dynamics. In CTMQC algorithm, we approximate $\mathcal{P}_\nu$ from multiple trajectories by putting a multi-dimensional Gaussian on each trajectory where the variance of each Gaussian is determined from the variance of the atomic positions of the overall trajectories. Thus, all nuclear trajectories are *coupled to* each other to determine the quantum momentum. In addition, the phase term $\mathbf{f}_{k,\nu}(t)$ is approximated as the time integration of BO forces as $\mathbf{f}_{k,\nu}(t) = -\int^t \nabla_\nu E_k(t')dt'$. As a result of the additional terms, electronic populations can change in the absence of nonadiabatic couplings which can describe decoherence of the electronic state. Since CTMQC method requires multiple trajectories running at the same time and the evaluation of BO force for all BO states, CTMQC usually needs a larger computational cost compared to independent trajectory approaches as FSSH and Ehrenfest. Moreover, the stability of QM calculations is of paramount importance. Therefore, CTMQC can be more useful when QM calculations are coupled to machine learning approaches.[60] The CTMQC method has successfully described the decoherence effects on excited state molecular dynamics simulations on various model systems and real molecules.[51–55,61]

2.1.4 | Decoherence-induced surface hopping based on exact factorization

The DISH-XF method complements the weakness of CTMQC method while maintaining its strength. The equations of motion for DISH-XF method consist of electronic part of CTMQC (Equation 8) and nuclear part of FSSH (Equation 3), thus DISH-XF is another derivative of FSSH with the decoherence correction. The quantum momentum $\mathcal{P}_\nu$ in Equation (8) is evaluated from the auxiliary trajectories. There are several algorithms which exploit auxiliary trajectories such as augmented FSSH (A-FSSH)[46] and overlap-based decoherence correction (ODC).[43] The ODC method estimates the decoherence by calculating nuclear wavefunction overlaps from auxiliary trajectories and renormalizes BO coefficients whenever the overlap becomes zero. The A-FSSH stochastically collapses the electronic state to the running state using decoherence rate calculated from auxiliary trajectories. In contrast to the above methods, DISH-XF does not require discontinuous renormalization of BO coefficients and the decoherence effect is included directly in the electronic equation of motion derived from the exact factorization formalism.

We generate auxiliary trajectories on BO states with nonzero population except the running state. The momenta of the auxiliary trajectories are calculated with uniform scaling of the real trajectory based on the energy conservation law. Similar to CTMQC, a multi-dimensional frozen Gaussian wave function centered at each auxiliary trajectory is used to calculate the quantum momentum term. However, in DISH-XF, the variances of the Gaussian are fixed as an initial parameter. A reasonable choice would be the variance of the initial distribution. The phase term $\mathbf{f}_{k,\nu}$ is approximated as a time integration of the momenta change of auxiliary trajectories on the BO state. The detailed equations and explanations can be found in the literature.[48] Therefore, DISH-XF is an independent trajectory approach, and we can preserve the efficiency of FSSH algorithm. Compared to the conventional FSSH, DISH-XF requires a few additional memories for decoherence quantities which can be calculated from the information for FSSH simulations straightforwardly. Thus, the electrons and nuclei can be propagated using DISH-XF method with almost same cost as the FSSH method.

In contrast to the fact that the previous ad-hoc corrections rather artificially force the wave-function to collapse into a running state, CTMQC and DISH-XF naturally introduce the decoherence and provide the norm conservation of the electronic wave function as a result of the electron-nuclear correlation. The DISH-XF method has been successfully applied to the excited state molecular dynamics simulations on various models and molecular systems.[18,48,62–65]

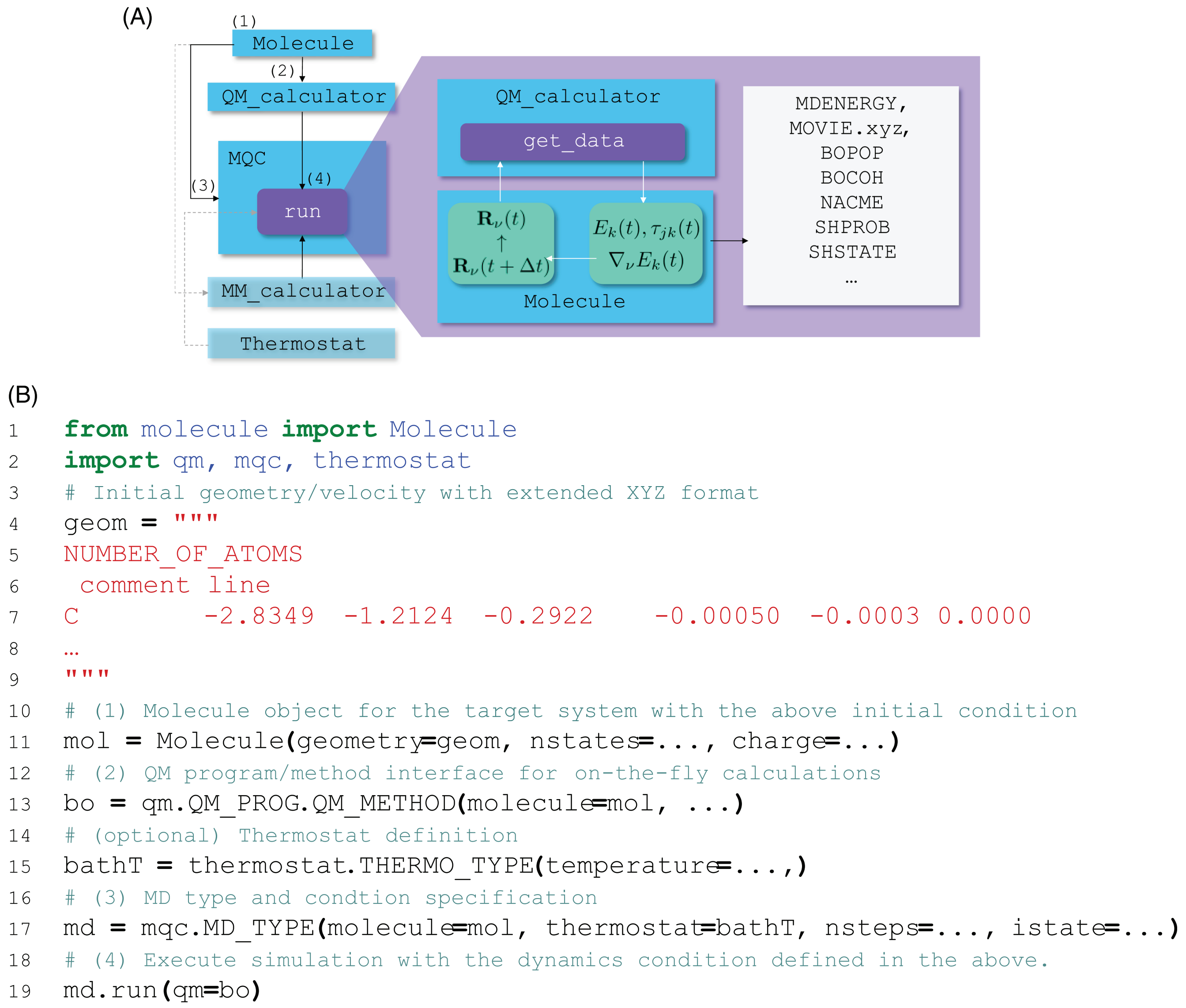
2.2 | PyUNIxMD package

In this section, we introduce the code structure and flow of the PyUNIxMD package. Then, we explain main components of PyUNIxMD package for MQC dynamics simulation. Finally, we explain post-processing scripts for analysis of MQC simulations included in the PyUNIxMD package. The names used in PyUNIxMD for classes, objects, methods, files, and variables are represented in italic font. In addition, we use the name of an object related to a class as same as the name of the class. PyUNIxMD is distributed via GitHub repository (https://github.com/skmin-lab/unixmd) under the MIT license.

2.2.1 | Structure and flow

The PyUNIxMD package consists of three main classes: *Molecule*, *QM_calculator* and *MQC* classes. *Molecule* defines a target system. *QM_calculator* interfaces several QM programs and methodologies to calculate the electronic structures. *MQC* has information about molecular dynamics. Detailed contents about each class will be discussed in the following sections. In addition, *MM_calculator* handles QM/MM calculation and *Thermostat* controls temperature of a target system.

Figure 1 shows the flow of the dynamics to execute PyUNIxMD and the corresponding running script. The flow is summarized as follows: (a) Define information for a target system in the *Molecule* object. (b) Define a *QM_calculator* object which has the level of computations to calculate the target system. One can also define optional condition such as thermostat. (c) Create a *MQC* object which includes the dynamics information as well as the previously defined *Molecule* object. (d) Execute the *run* method of the *MQC* object with the defined *QM_calculator* object. As a result, the information about the target system such as atomic positions, energies and BO populations is updated and written in several output files during time propagation.



```
from molecule import Molecule
import qm, mqc, thermostat
# Initial geometry/velocity with extended XYZ format
geom = """
NUMBER_OF_ATOMS
 comment line
C        -2.8349  -1.2124  -0.2922    -0.00050  -0.0003 0.0000
...
"""
# (1) Molecule object for the target system with the above initial condition
mol = Molecule(geometry=geom, nstates=..., charge=...)
# (2) QM program/method interface for on-the-fly calculations
bo = qm.QM_PROG.QM_METHOD(molecule=mol, ...)
# (optional) Thermostat definition
bathT = thermostat.THERMO_TYPE(temperature=...,)
# (3) MD type and condtion specification
md = mqc.MD_TYPE(molecule=mol, thermostat=bathT, nsteps=..., istate=...)
# (4) Execute simulation with the dynamics condition defined in the above.
md.run(qm=bo)
```

FIGURE 1 (A) Structure of the PyUNIxMD package and data flow in a NAMD simulation using PyUNIxMD. The *Molecule* object is updated by the *MQC* object during the dynamics while properties are calculated by the *QM_calculator* (and *MM_calculator*, optionally). (B) An example of a typical running script for the PyUNIxMD execution

Molecule class

Molecule has basic information about the system such as geometries, velocities, charge and the number of states. Since the *Molecule* object is used for creating *MQC* and *QM_calculator* objects, the *Molecule* object must be defined prior to other objects. *Molecule* employs an extended XYZ format to set a geometry and a velocity. Especially, PyUNIxMD handles state-specific information such as energy and force with *State* class for each BO state, and *Molecule.states* object is declared as the *State* class. Hence, one can easily access the state-specific information with *Molecule.states* objects for specific purposes.

MQC class

The *MQC* class is the central class of PyUNIxMD which defines and runs NAMD simulation. Subclasses of the *MQC* class determine a MQC dynamics method as *BOMD* for BOMD, *Eh* for Ehrenfest dynamics, *SH* for FSSH and its derivatives, *SHXF* for DISH-XF and *CT* for CTMQC. As common variables, the subclasses requires a geometry object (*molecule*), the number of nuclear time steps (*nsteps*), a time step size (*dt*), and the initial electronic state (*istate*). Of course, each subclass has its own input parameters. For example, one can choose the rescaling method by defining *hop_rescale* variable for *SH* and *SHXF*. For IDC and EDC decoherence corrections, one can define *dec_correction* as “idc” and “edc”, respectively, in SH.

- *run* method

After the *MQC* object is created, the dynamics simulation is executed by calling *run* method of the *MQC* object. In order to integrate the equation of motions for the dynamics, the *run* method requires electronic structure calculation of the molecule at every time step.

Therefore, the *run* method takes a *QM_calculator* object that interfaces with the external QM program. If one wants to conduct the dynamics with a thermostat, a *Thermostat* object must be additionally provided. Detailed description about *QM_calculator* and *Thermostat* classes will be given in the following sections.

In the *MQC.run* method, the following process is repeated until the dynamics reaches the maximum time step (*nsteps*): (a) The molecular geometry is transferred to the *QM_calculator* object. (b) QM calculation is executed, and the calculated properties are stored to the *Molecule* object. (c) Using the properties, the *run* method updates the atomic positions and the electronic properties according to the selected MQC algorithm. (d) If *Thermostat* object exists, the velocities are adjusted according to the thermostat. (e) Finally, the *run* method writes the information about the trajectory. PyUNIxMD writes the following output files during a simulation depending on the type of MQC algorithm: *RESTART.bin*, *MDENERGY*, *BOPOP*, *BOCOH*, *NACME*, *MOVIE.xyz*, *SHPROB*, and *SHSTATE* contain dynamics information at the last successful step, energies, BO populations, BO coherence, NACMEs, the nuclear trajectory, hopping probabilities, and the running state, respectively (Figure 2).

Since QM calculations at different time steps are independent, the phases of the wavefunctions and the NACVs are arbitrary. PyUNIxMD aligns the phases of NACVs at every nuclear time step after QM calculations using the dot products of normalized NACVs at t and $t-dt$, that is, $\mathbf{d}_{jk,\nu}(t) \rightarrow -\mathbf{d}_{jk,\nu}(t)$ if $\frac{\sum_{\nu} \mathbf{d}_{jk,\nu}(t)\cdot\mathbf{d}_{jk,\nu}(t-dt)}{\left\|\mathbf{d}_{jk,\nu}(t)\right\|\left\|\mathbf{d}_{jk,\nu}(t-dt)\right\|} < 0$. PyUNIxMD package employs velocity-Verlet[66] algorithm for the nuclear propagation and the 4th order Runge–Kutta scheme (RK4)[67] for the electronic propagation (Equations 2 and 8). Electronic state is more sensitive to the time interval than nuclear motion, therefore, a smaller electronic time step is usually used and propagated with linear interpolation of properties between adjacent nuclear time steps. PyUNIxMD uses the nuclear time step size (*dt*) as 0.5 fs and the number of electronic time steps for each nuclear step (*nesteps*) as 20 by default, which is obtained from convergence tests with several models and realistic molecular systems. However, simulation results may be different from the choice of *nesteps* depending on the system. Therefore, the convergence test is required for more rigorous simulations. Especially, a system with extremely localized nonadiabatic coupling regions requires a completely different electronic propagation scheme such as local diabatization with exponential time propagator[68] which is under development in PyUNIxMD. Since the electronic propagation is computationally expensive, the corresponding codes are written in C and interfaced to the *MQC* class via Cython.

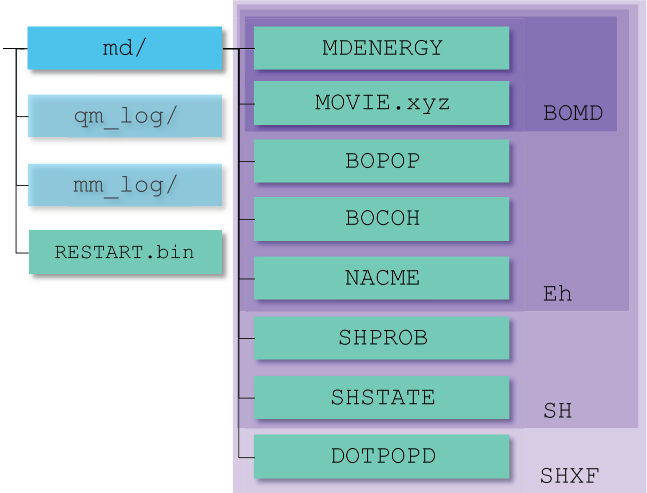


FIGURE 2 A file tree generated by PyUNIxMD. PyUNIxMD generates directories for MD outputs, logs from QM and MM calculations. The MD outputs vary according to the MQC method. The blue and light green boxes represent directories and files, respectively. The purple boxes distinguish output files that vary according to the MQC method

QM_calculator class

QM_calculator class provides interfaces to various external (or internal, for simple model calculations) QM programs. PyUNIxMD supports complete active space self-consistent field (CASSCF) in Columbus[69] and Molpro,[33] multireference configuration interaction (MRCI) in Columbus, time-dependent density-functional theory (TDDFT) in Gaussian 09[70] and Q-Chem,[32] state-interaction state-averaged spin-restricted ensemble Kohn-Sham (SSR) in TeraChem,[71] time-dependent density-functional tight-binding (TDDFTB) and density-functional tight-binding based on SSR (DFTB/SSR) in DFTB+.[72] PyUNIxMD has two level subclasses related to *QM_calculator*: A higher subclass which inherits the *QM_calculator* class to determine a QM program, and a lower subclass to further choose a specific QM method in the corresponding QM program. For example, an interfacing class for CASSCF calculation using Molpro is *CASSCF* subclass of *Molpro* subclass that inherits the *QM_calculator* class.

Depending on the availability of NACVs calculations, PyUNIxMD supports Ehrenfest and CTMQC dynamics. For example, NACVs can be calculated with CASSCF or SSR method. Thus, all MQC dynamics are possible with these methods. In contrast, the NACVs cannot be provided from TDDFT method except Q-Chem. Instead, the numerical evaluation of NACMEs[59] is supported in PyUNIxMD so that surface hopping based dynamics can be exploited with TDDFT of Gaussian 09 or TDDFTB of DFTB +. The compatibility between MQC dynamics and QM methods in the current version of PyUNIxMD is given in Table 1.

- *get_data* method

In the *MQC.run* method, the *get_data* method of the *QM_calculator* class carries out the QM calculation by executing the QM program with an input based on the information of the *Molecule* object. After QM calculation, the *get_data* reads the resulting output data, and assigns the energies, forces and NACVs to *Molecule* object for the nuclear and electronic propagations. One can easily add new QM interfaces to PyUNIxMD by making subclasses of *QM_calculator* class with proper *get_data* method. In addition, *get_data* provides NACMEs calculations from CIS coefficients (C_{ia}^{K}) and Kohn-Sham molecular orbitals (ϕ_i) for TDDFTB in DFTB+ and TDDFT in Gaussian 09 as[59]:

TABLE 1 Compatibility between QM programs and MQC methods in PyUNIxMD

Program	Method	BOMD	Ehrenfest	FSSH	DISH-XF	CTMQC
Columbus[69]	CASSCF	✓	✓	✓	✓	✓
	MRCI	✓	✓	✓	✓	✓
DFTB +[72]	TDDFTB	✓	–	✓	✓	–
	DFTB/SSR	✓	✓	✓	✓	✓
Gaussian 09[70]	TDDFT	✓	–	✓	✓	–
Molpro[33]	CASSCF	✓	✓	✓	✓	✓
Q-Chem[32]	TDDFT	✓	✓	✓	✓	✓
TeraChem[71]	SSR	✓	✓	✓	✓	✓

$$\sigma_{KJ} = \sum_{ia} C_{ia}^{K} \partial_t C_{ia}^{J} + \sum_{iab} C_{ia}^{K} C_{ib}^{J} \langle \phi_a | \partial_t \phi_b \rangle - \sum_{ija} P_{ij} C_{ia}^{K} C_{ja}^{J} \langle \phi_j | \partial_t \phi_i \rangle, \tag{9}$$

where (i,j), (a,b) and (K,J) are indices for occupied orbitals, virtual orbitals, and total electronic states, respectively, and P_{ij} is a phase factor (1 or $(-1)^{|i-j|}$) depending on the orbital ordering convention for slater determinants. The phase freedom of the NACMEs are adjusted during the calculation of wavefunction overlaps between t and $t-dt$. Similar to the electronic propagation, Equation (9) is written in C interfaced via Cython due to the scaling with the orbital size ($\sim N^3$).

MM_calculator class

MM_calculator is an optional class of PyUNIxMD that interfaces to the MM programs for QM/MM calculations. Currently only Tinker[73] is interfaced with PyUNIxMD. As in the *QM_calculator* object, the *get_data* method runs the external MM calculation by generating an input file based on *Molecule* object. After the MM calculation, *get_data* assigns the resulting energies and forces to *Molecule* object. As a summary, one can perform QM/MM-based NAMD simulations by putting the *MM_calculator* object in the *MQC.run* method after defining the *MM_calculator* object. Other MM programs or detailed QM/MM schemes will be implemented in the future.

Thermostat class

A temperature of the system can be controlled by a thermostat during the dynamics. PyUNIxMD provides three types of thermostat classes; velocity-rescaling, Berendsen,[74] and Nose-Hoover chain.[75] Two types of velocity-rescaling thermostat are implemented in PyUNIxMD: A simple rescaling at every *n* step to a target temperature, and a rescaling only if a current temperature deviates from the target temperature by a certain amount. Berendsen and Nose-Hoover chain thermostats are implemented as demonstrated in the references.[74,75] Since the *MQC* object uses *Thermostat* object, one must define the *Thermostat* prior to the *MQC* object.

Auxiliary scripts

PyUNIxMD provides several scripts for analysis with the multiple output files in Figure 2. One can calculate an averaged value of any observable $\langle O(t) \rangle$ over multiple trajectories as:

$$\langle O(t) \rangle = \sum_{I}^{N_{traj}} \frac{O^{(I)}(t)}{N_{traj}}, \tag{10}$$

where $O^{(I)}(t)$ is an observable of the I th trajectory at time t. The script *statistical_analysis.py* calculates averaged values of BO populations ($\langle \rho_{ii}(t) \rangle$), coherence indicators ($\left\langle \left| \rho_{ij}(t) \right|^2 \right\rangle$), and NACMEs ($\left\langle \left| \sigma_{ij}(t) \right| \right\rangle = \left\langle \left| \sum_{\nu} \mathbf{d}_{ij,\nu}(t) \cdot \dot{\mathbf{R}}_{\nu}(t) \right| \right\rangle$). For FSSH simulations, the script calculates another average BO populations based on running states of the trajectories given by:

$$p_i(t) = \frac{N_i(t)}{N_{traj}}, \tag{11}$$

where $N_i(t)$ is the number of trajectories with the running state i. While a molecule undergoes a nonadiabatic process, the nuclear wave packet may branch into different reaction pathways. The script *motion_analysis.py* extracts bond lengths, bond angles, or dihedral angles between selected atoms from the *MOVIE.xyz* file and store values in separate files. The script also provides mean values of bond lengths, bond angles, or dihedral angles over an ensemble of trajectories. One can analyze the reaction pathways by monitoring these fundamental geometric parameters. The example results generated by these scripts are illustrated in the following section.

3 | EXAMPLES

As a demonstration of the NAMD simulation with the PyUNIxMD package, we study photoisomerization dynamics of a molecular motor in the gas phase.[76] The chemical structure and numbering of the molecular motor used in this article are given in Figure 3. The initial conditions for photoisomerization dynamics can be obtained from Boltzmann or Wigner sampling. Here we prepare the initial conditions based on Boltzmann sampling by performing BOMD on the ground state for 20 ps with a time step of 0.5 fs using randomly distorted structures from the optimized structure. We employ the velocity rescaling thermostat after every 10 fs at 300 K. After the thermal equilibrium, 100 trajectories from 10 ps to 20 ps are extracted for the initial conditions of excited state molecular dynamics simulations. Using the 100 initial conditions, we perform FSSH, FSSH with EDC (SHEDC), and DISH-XF dynamics with a time step of 0.5 fs without

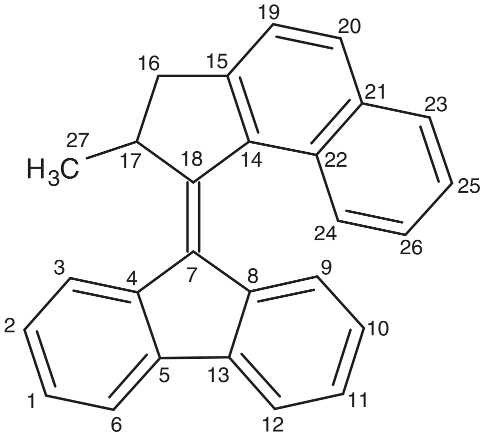


FIGURE 3 Chemical structure of the molecular motor used in the molecular dynamics. The motor shows photoisomerization dynamics around center C7 = C18 double bond after a light absorption. All carbon atoms are labeled. The dihedral angle (θ) around the central C7 = C18 is defined as the angle between a plane with C7, C4, and C8 and a plane with C18, C17, and C14

thermostat for 5 ps on the S_1 state initially. In addition, we use the momentum rescaling along NACVs after a hop occurs and preserve the momentum for a rejected hop. For SHEDC, we use a default value for a pre-determined parameter, C as 0.1 Hartree. For DISH-XF, we use width of 0.1 a.u. initially. We choose long-range corrected and onsite corrected DFTB/SSR (LC-OC-DFTB/SSR) method with ob2 parameters[77] for QM calculation where the LC-OC-DFTB/SSR method is a modification of LC-DFTB/SSR method[78] by implementing onsite correction scheme.[79]

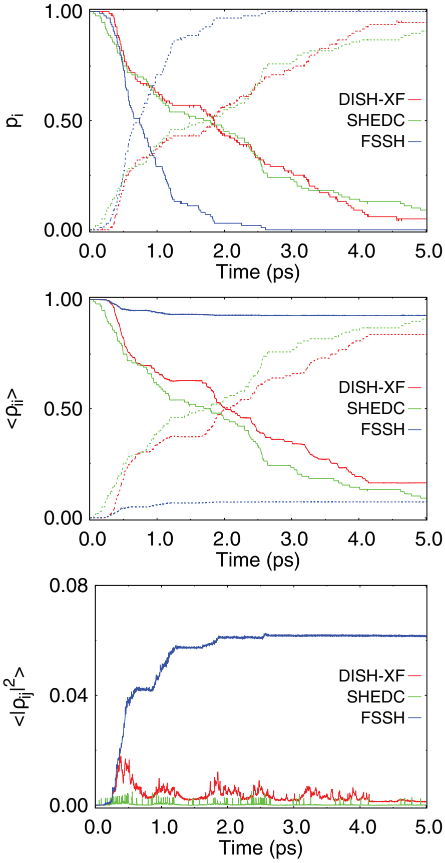


FIGURE 4 Averaged electronic populations, p_i and $\langle\rho_{ii}\rangle$, and coherences, $\left\langle|\rho_{ij}|^2\right\rangle$ from FSSH, SHEDC, and DISH-XF simulations for the molecular motor system. Red, green, and blue solid lines represent the results calculated from DISH-XF, SHEDC, and FSSH, respectively. The bold and dotted lines for electronic populations represent the population of S_1 and S_0 states, respectively

Figure 4 shows average populations and coherences with the FSSH, SHEDC, and DISH-XF methods. We can calculate the average populations from BO populations ($\langle\rho_{ii}\rangle$) or running states (p_i) using Equations (10) and (11). While p_i and $\langle\rho_{ii}\rangle$ show completely different behaviors for FSSH, the average populations are almost on top of each other for DISH-XF and SHEDC indicating decoherence correction recovers internal consistency ($p_i \approx \langle\rho_{ii}\rangle$). We notice that three methods show similar behavior up to earlier 500 fs and yield different propagation for p_i, $\langle\rho_{ii}\rangle$, and $\left\langle|\rho_{ij}|^2\right\rangle$ afterwards. The lifetimes on the S_1 state can be calculated from the monoexponential fitting process for S_1 population. The lifetime of the S_1 state calculated from the FSSH method is 600 fs similar to 710 fs from FSSH with OM2/MRCI.[76] While the DISH-XF and SHEDC method shows the lifetime as 1.97 and 2.15 ps, respectively. In FSSH, the lifetime becomes small because of the overestimation of the hopping probabilities. The lack of decoherence induces the accumulation of the off-diagonal elements in the electronic density matrix which results in the increase of hopping probabilities. In addition, once the system goes to the ground state, the opposite hopping is mostly forbidden due to the energy conservation in this particular example. For DISH-XF and SHEDC, they show similar averaged electronic populations although the equations of motion are different. For SHEDC, the decoherence depends on a damping parameter determined from the kinetic energy and the difference between potential energies, see Equations (5) and (6). Thus, the electronic populations exponentially decay to the running state at every step. In DISH-XF, the decoherence term becomes nonzero which induces further population transfer from one state to the other state after nonadiabatic coupling region. Depending on auxiliary trajectories and phase factors, the direction of population transfer is determined (Equation 8).[48] Due to the decoherence, the trajectory with DISH-XF and SHEDC tends to stay longer on the initial S_1 state showing the slower decay of S_1 population. In lower panel of Figure 5, average coherences for DISH-XF and SHEDC show several peaks with the period of ~800 fs, while the FSSH shows an accumulation of coherence. SHEDC shows several sharp peaks caused from the fast exponential decay, while DISH-XF shows broad peaks due to the relatively slow decoherence. Thus, in this example, the decoherence of SHEDC is stronger than DISH-XF. Overall, the results of DISH-XF and SHEDC indicate that the S_1 population decays discretely with a certain period.

The role of decoherence is more clear when we analyze individual trajectories. Figure 5 shows BO energies, BO populations, and a

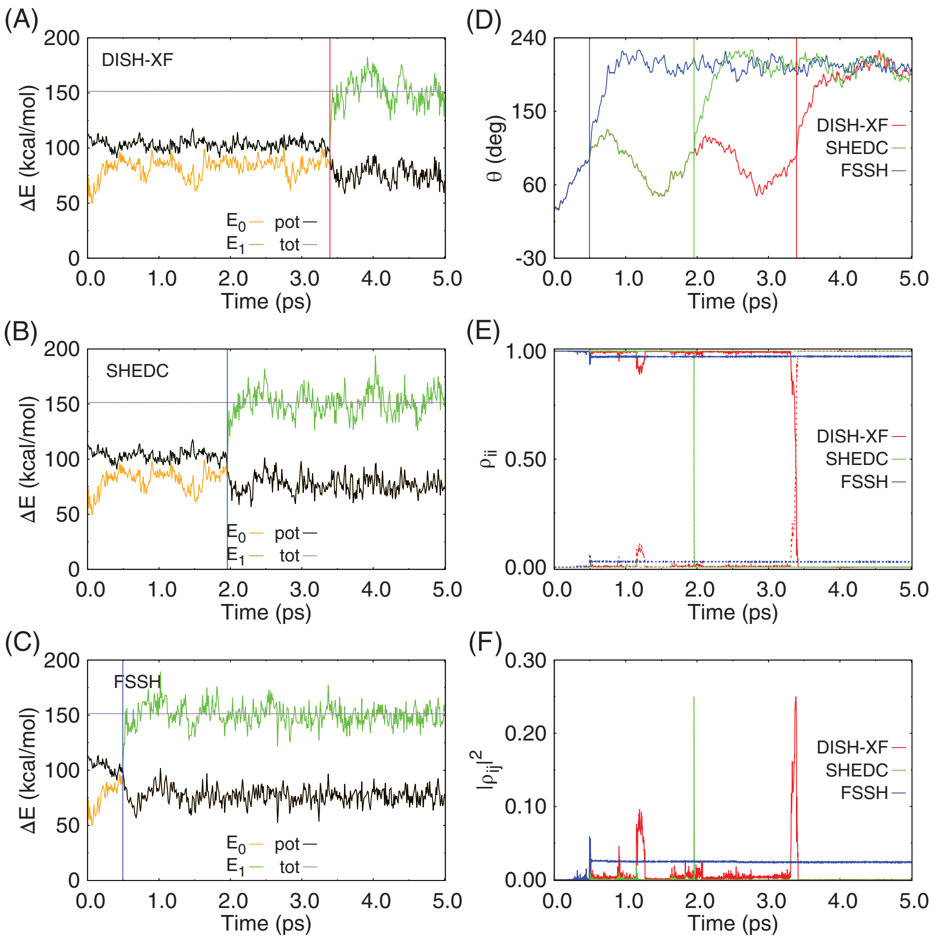


FIGURE 5 The time evolution of (A, B, C) energies, (D) the dihedral angle along the central C=C bond, (E) BO populations and (F) the coherence for a selected trajectory in DISH-XF, SHEDC, and FSSH simulations. In (A, B, C), yellow, green, black, and purple lines represent potential energies of S_0, S_1, and the running state, and the total energy, respectively

coherence indicator of a selected trajectory for FSSH, SHEDC, and DISH-XF. The initial conditions are all identical in this case. BO populations of FSSH shows gradual increase (or decrease) and the hop from S_1 to S_0 occurs directly in the earlier time. However, the additional decoherence effect makes the BO population recover the pure electronic state ($\rho_{11} = 1$) for DISH-XF and SHEDC. Thus, the overall BO population exchanges are slower in DISH-XF and SHEDC. We can also notice that ρ_{11} becomes 1 or 0 eventually when the running state is S_1 or S_0, respectively. As a result, $|\rho_{12}(t)|^2$ shows a trivial difference between FSSH and the other two methods. In FSSH, the $|\rho_{12}(t)|^2$ remains finite after a hop without any decay. However, the $|\rho_{12}(t)|^2$ from DISH-XF and SHEDC increases as the trajectory passes the nonadiabatic coupling region, and decreases as the trajectory leaves the region.

The structure analysis with motion_analysis.py provides the time evolution of the dihedral angle around C7 = C18 double bond (see Figure 3) from the selected trajectory for the DISH-XF, SHEDC, and FSSH. FSSH shows a fast isomerization via a conical intersection while DISH-XF and SHEDC preserve the running state for a while showing an oscillation of the dihedral angle. Based on time-dependent energy profiles and dihedral angles in Figure 5, the molecular motor reaches a coupling region with a small energy gap as the diheral angle becomes $\sim 90^\circ$. In this case, the DISH-XF and SHEDC trajectories cross the coupling region five and three times, respectively, while the FSSH trajectory crosses the coupling region one time, which indicates the DISH-XF and SHEDC trajectories stay on the upper state longer than the FSSH trajectory. Overall, these trajectories eventually show successful isomerization with the dihedral angle of $\sim 210^\circ$. Finally, we calculate the quantum yield of photoisomerization based on the number of trajectories showing the isomerization over the total number of trajectories. The quantum yield for the isomerization from the FSSH is 0.68 while the yield from the DISH-XF and SHEDC becomes 0.59 and 0.46, respectively. The quantum yield in the reference is 0.6 based on semiempirical methods.[76] We notice that the calculated quantum yields are considerably larger than the experimental result showing 0.14 in presence of toluene solvents.[80] Thus, the correct consideration of thermal effects due to environments is crucial to describe molecular rotors in a solution.

4 | CONCLUSIONS

In this article, we presented an open-source Python-based NAMD program, PyUNIxMD package. PyUNIxMD provides wide choices of MQC methods and QM interfaces. Especially, CTMQC and DISH-XF are implemented for general purposes for the first time. PyUNIxMD also offers several auxiliary scripts for preparation of NAMD simulations and analysis. We performed FSSH and DISH-XF simulations for photoisomerization dynamics of a molecular motor system as a demonstration of the PyUNIxMD package. Compared to FSSH, DISH-XF exhibits distinct decoherence effect with better internal consistency, and a longer lifetime. Since the overall code structure of the PyUNIxMD is straightforward and well-organized, it is easy to perform NAMD simulations and add new interfaces and features for QM and MQC methods. We expect that PyUNIxMD provides an efficient NAMD simulation tool and development platform of new NAMD methods. In addition, automation of multiple NAMD simulations can easily be achieved via simple scripts with PyUNIxMD. Such automation is important in terms of congruency, reproducibility, and user independency as addressed in molecular model construction studies.[81,82] We plan to implement AIMS, and to update interfaces for QM/MM environments, other QM methods such as CASPT2, various electronic propagators, and additional scripts for Wigner sampling and result analysis.

ACKNOWLEDGMENTS

This work was supported by the research fund (1.190123.01) of Ulsan National Institute of Science and Technology (UNIST) and the research fund (NRF-2019R1A2C1007744 and NRF-2019R1A4A1027934) of the National Research Foundation of Korea (NRF) by the Ministry of Science and ICT.

DATA AVAILABILITY STATEMENT

The PyUNIxMD code, presented and used within this study, is openly available at GitHub (https://github.com/skmin-lab/unixmd). Initial sample is available in Supporting Information.

ORCID

Jong-Kwon Ha https://orcid.org/0000-0003-4694-5627
Seung Kyu Min https://orcid.org/0000-0001-5636-3407

REFERENCES

[1] G. D. Scholes, *Nature* **2017**, *543*, 647.
[2] E. Romero, V. I. Novoderezhkin, R. van Grondelle, *Nature* **2017**, *543*, 355.
[3] M. Kaucikas, K. Maghlaoui, J. Barber, T. Renger, J. J. Van Thor, *Nat. Commun.* **2016**, *7*, 13977.
[4] D. J. Nürnberg, J. Morton, S. Santabarbara, A. Telfer, P. Joliot, L. A. Antonaru, A. V. Ruban, T. Cardona, E. Krausz, A. Boussac, A. Fantuzzi, A. W. Rutherford, *Science* **2018**, *360*, 1210.
[5] J. C. Blancon, *Science* **2017**, *355*, 1288.
[6] S. Nah, B. Spokoyny, C. Stoumpos, C. Soe, M. Kanatzidis, E. Harel, *Nat. Photonics* **2017**, *11*, 285.
[7] J. Huang, Y. Yuan, Y. Shao, Y. Yan, *Nat. Rev. Mater.* **2017**, *2*, 17042.
[8] L. Qiao, W.-H. Fang, R. Long, O. V. Prezhdo, *J. Phys. Chem. Lett.* **2020**, *11*, 7066.
[9] M. Garavelli, P. Celani, F. Bernardi, M. A. Robb, M. Olivucci, *J. Am. Chem. Soc.* **1997**, *119*, 6891.
[10] L. M. Frutos, T. Andruniów, F. Santoro, N. Ferré, M. Olivucci, *Proc. Natl. Acad. Sci. USA* **2007**, *104*, 7764.
[11] D. Polli, P. Altoé, O. Weingart, K. M. Spillane, C. Manzoni, D. Brida, G. Tomasello, G. Orlandi, P. Kukura, R. A. Mathies, M. Garavelli, G. Cerullo, *Nature* **2010**, *467*, 440.
[12] E. Nango, A. Royant, M. Kubo, T. Nakane, C. Wickstrand, T. Kimura, T. Tanaka, K. Tono, C. Song, R. Tanaka, T. Arima, A. Yamashita, J. Kobayashi, T. Hosaka, E. Mizohata, P. Nogly, M. Sugahara, D. Nam, T. Nomura, D. Shimamura, T. Im, T. Fujiwara, Y. Yamanaka, B. Jeon, T. Nishizawa, K. Oda, M. Fukuda, R. Andersson, P. Båth, J. Dods, R. Davidsson, S. Matsuoka, S. Kawatake, M. Murata, O. Nureki, S. Owada, T. Kameshima, T. Hatsui, Y. Joti, G. Schertler, M. Yabashi, A.-N. Bondar, R. Standfuss, Jörg, Neutze, S. Iwata, *Science* **2016**, *354*, 1552.
[13] S. Gozem, H. L. Luk, I. Schapiro, M. Olivucci, *Chem. Rev.* **2017**, *117*, 13502.
[14] C. Schnedermann, X. Yang, M. Liebel, K. M. Spillane, J. Lugtenburg, I. Fernandez, A. Valentini, I. Schapiro, M. Olivucci, P. Kukura, R. A. Mathies, *Nat. Chem.* **2018**, *10*, 449.
[15] Z. Wang, C. Li, K. Domen, *Chem. Soc. Rev.* **2019**, *48*, 2109.
[16] V. K. Singh, C. Yu, S. Badgujar, Y. Kim, Y. Kwon, D. Kim, J. Lee, T. Akhter, G. Thangavel, L. S. Park, J. Lee, P. C. Nandajan, R. Wannemacher, B. N. Milián-Medina, Lüer, K. S. Kim, J. Gierschner, M. S. Kwon, *Nat. Catal.* **2018**, *1*, 794.
[17] S. Poplata, A. Troester, Y.-Q. Zou, T. Bach, *Chem. Rev.* **2016**, *116*, 9748.
[18] M. Filatov, S. K. Min, K. S. Kim, *Mol. Phys.* **2019**, *117*, 1128.
[19] T. J. A. Wolf, R. M. Parrish, R. H. Myhre, T. J. Martínez, H. Koch, M. Gühr, *J. Phys. Chem. A* **2019**, *123*, 6897.
[20] R. Crespo-Otero, M. Barbatti, *Chem. Rev.* **2018**, *118*, 7026.
[21] M. Ben-Nun, J. Quenneville, T. J. Martínez, *J. Chem. Phys. A* **2000**, *104*, 5161.
[22] B. F. E. Curchod, T. J. Martínez, *Chem. Rev.* **2018**, *118*, 3305.
[23] A. D. McLachlan, *Mol. Phys.* **1964**, *8*, 39.
[24] J. C. Tully, *J. Chem. Phys.* **1990**, *93*, 1061.
[25] S. Mai, P. Marquetand, L. González, *WIREs Comput. Mol. Sci.* **2018**, *8*, e1370.
[26] M. Barbatti, M. Ruckenbauer, F. Plasser, J. Pittner, G. Granucci, M. Persico, H. Lischka, *WIREs Comput. Mol. Sci.* **2014**, *4*, 26.
[27] A. V. Akimov, *J. Comput. Chem.* **2016**, *37*, 1626.
[28] A. V. Akimov, O. V. Prezhdo, *J. Chem. Theory Comput.* **2013**, *9*, 4959.
[29] L. Du, Z. Lan, *J. Chem. Theory Comput.* **2015**, *11*, 1360.
[30] D. A. Fedorov, S. Seritan, B. S. Fales, T. J. Martínez, B. G. Levine, *J. Chem. Theory Comput.* **2020**, *16*, 5485.
[31] W. Malone, B. Nebgen, A. White, Y. Zhang, H. Song, J. A. Bjorgaard, A. E. Sifain, B. Rodriguez-Hernandez, V. M. Freixas, S. Fernandez-Alberti, A. E. Roitberg, T. R. Nelson, S. Tretiak, *J. Chem. Theory Comput.* **2020**, *16*, 5771.
[32] Y. Shao, Z. Gan, E. Epifanovsky, A. T. Gilbert, M. Wormit, J. Kussmann, A. W. Lange, A. Behn, J. Deng, X. Feng, D. Ghosh, M. Goldey, P. R. Horn, L. D. Jacobson, I. Kaliman, R. Z. Khaliullin, T. Kuś, A. Landau, J. Liu, E. I. Proynov, Y. M. Rhee, R. M. Richard, M. A. Rohrdanz, R. P. Steele, E. J. Sundstrom, P. M. Zimmerman, D. Zuev, B. Albrecht, E. Alguire, B. Austin, G. J. O. Beran, Y. A. Bernard, E. Berquist, K. Brandhorst, K. B. Bravaya, S. T. Brown, D. Casanova, C.-M. Chang, Y. Chen, S. H. Chien, K. D. Closser, D. L. Crittenden, M. Diedenhofen, H. Do, A. D. Dutoi, R. G. Edgar, S. Fatehi, L. Fusti-Molnar, A. Ghysels, A. Golubeva-Zadorozhnaya, J. Gomes, M. W. Hanson-Heine, P. H. Harbach, A. W. Hauser, E. G. Hohenstein, Z. C. Holden, T.-C. Jagau, H. Ji, B. Kaduk, K. Khistyaev, J. Kim, J. Kim, R. A. King, P. Klunzinger, D. Kosenkov, T. Kowalczyk, C. M. Krauter, K. U. Lao, A. D. Laurent, K. V. Lawler, S. V. Levchenko, C. Y. Lin, F. Liu, E. Livshits, R. C. Lochan, A. Luenser, P. Manohar, S. F. Manzer, S.-P. Mao, N. Mardirossian, A. V. Marenich, S. A. Maurer, N. J. Mayhall, E. Neuscamman, C. M. Oana, R. Olivares-Amaya, D. P. O'Neill, J. A. Parkhill, T. M. Perrine, R. Peverati, A. Prociuk, D. R. Rehn, E. Rosta, N. J. Russ, S. M. Sharada, S. Sharma, D. W. Small, A. Sodt, T. Stein, D. Stück, Y.-C. Su, A. J. Thom, T. Tsuchimochi, V. Vanovschi, L. Vogt, O. Vydrov, T. Wang, M. A. Watson, J. Wenzel, A. White, C. F. Williams, J. Yang, S. Yeganeh, S. R. Yost, Z.-Q. You, I. Y. Zhang, X. Zhang, Y. Zhao, B. R. Brooks, G. K. Chan, D. M. Chipman, C. J. Cramer, M. S. Gordon, W. J. Hehre, A. Klamt, H. F. S. III, M. W. Schmidt, C. D.

Sherrill, D. G. Truhlar, A. Warshel, X. Xu, A. Aspuru-Guzik, R. Baer, A. T. Bell, N. A. Besley, J.-D. Chai, A. Dreuw, B. D. Dunietz, T. R. Furlani, S. R. Gwaltney, C.-P. Hsu, Y. Jung, J. Kong, D. S. Lambrecht, W. Liang, C. Ochsenfeld, V. A. Rassolov, L. V. Slipchenko, J. E. Subotnik, T. V. Voorhis, J. M. Herbert, A. I. Krylov, P. M. Gill, M. Head-Gordon, *Mol. Phys.* **2015**, *113*, 184.

[33] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, *WIREs Comput. Mol. Sci.* **2012**, *2*, 242.

[34] G. M. J. Barca, C. Bertoni, L. Carrington, D. Datta, N. De Silva, J. E. Deustua, D. G. Fedorov, J. R. Gour, A. O. Gunina, E. Guidez, T. Harville, S. Irle, J. Ivanic, K. Kowalski, S. S. Leang, H. Li, W. Li, J. J. Lutz, I. Magoulas, J. Mato, V. Mironov, H. Nakata, B. Q. Pham, P. Piecuch, D. Poole, S. R. Pruitt, A. P. Rendell, L. B. Roskop, K. Ruedenberg, T. Sattasathuchana, M. W. Schmidt, J. Shen, L. Slipchenko, M. Sosonkina, V. Sundriyal, A. Tiwari, J. L. G. Vallejo, B. Westheimer, M. Wloch, P. Xu, F. Zahariev, M. S. Gordon, *J. Chem. Phys.* **2020**, *152*, 154102.

[35] S. G. Balasubramani, G. P. Chen, S. Coriani, M. Diedenhofen, M. S. Frank, Y. J. Franzke, F. Furche, R. Grotjahn, M. E. Harding, C. Hättig, A. Hellweg, B. Helmich-Paris, C. Holzer, U. Huniar, M. Kaupp, A. Marefat Khah, S. Karbalaei Khani, T. Müller, F. Mack, B. D. Nguyen, S. M. Parker, E. Perlt, D. Rappoport, K. Reiter, S. Roy, M. Rückert, G. Schmitz, M. Sierka, E. Tapavicza, D. P. Tew, C. van Wüllen, V. K. Voora, F. Weigend, A. Wodyński, J. M. Yu, *J. Chem. Phys.* **2020**, *152*, 184107.

[36] N. Tancogne-Dejean, M. J. T. Oliveira, X. Andrade, H. Appel, C. H. Borca, G. Le Breton, F. Buchholz, A. Castro, S. Corni, A. A. Correa, U. De Giovannini, A. Delgado, F. G. Eich, J. Flick, G. Gil, A. Gomez, N. Helbig, H. Hübener, R. Jestädt, J. Jornet-Somoza, A. H. Larsen, I. V. Lebedeva, M. Lüders, M. A. L. Marques, S. T. Ohlmann, S. Pipolo, M. Rampp, C. A. Rozzi, D. A. Strubbe, S. A. Sato, C. Schäfer, I. Theophilou, A. Welden, A. Rubio, *J. Chem. Phys.* **2020**, *152*, 124119.

[37] M. Valiev, E. J. Bylaska, N. Govind, K. Kowalski, T. P. Straatsma, H. J. J. Van Dam, D. Wang, J. Nieplocha, E. Apra, T. L. Windus, W. A. de Jong, *Comput. Phys. Commun.* **2010**, *181*, 1477.

[38] J. E. Subotnik, A. Jain, B. Landry, A. Petit, W. Ouyang, N. Bellonzi, *Annu. Rev. Phys. Chem.* **2016**, *67*, 387.

[39] A. W. Jasper, S. Nangia, C. Zhu, D. G. Truhlar, *Acc. Chem. Res.* **2006**, *39*, 101.

[40] O. V. Prezhdo, P. J. Rossky, *J. Chem. Phys.* **1997**, *107*, 5863.

[41] C. Zhu, S. Nangia, A. W. Jasper, D. G. Truhlar, *J. Chem. Phys.* **2004**, *121*, 7658.

[42] G. Granucci, M. Persico, *J. Chem. Phys.* **2007**, *126*, 134114.

[43] G. Granucci, M. Persico, A. Zoccante, *J. Chem. Phys.* **2010**, *133*, 134111.

[44] H. M. Jaeger, S. Fischer, O. V. Prezhdo, *J. Chem. Phys.* **2012**, *137*, 22A545.

[45] T. Nelson, S. Fernandez-Alberti, A. E. Roitberg, S. Tretiak, *J. Chem. Phys.* **2013**, *138*, 224111.

[46] A. Jain, E. Alguire, J. E. Subotnik, *J. Chem. Theory Comput.* **2016**, *12*, 5256.

[47] X. Gao, W. Thiel, *Phys. Rev. E* **2017**, *95*, 013308.

[48] J.-K. Ha, I. S. Lee, S. K. Min, *J. Phys. Chem. Lett.* **2018**, *9*, 1097.

[49] A. Abedi, N. T. Maitra, E. K. U. Gross, *Phys. Rev. Lett.* **2010**, *105*, 123002.

[50] A. Abedi, N. T. Maitra, E. K. U. Gross, *J. Chem. Phys.* **2012**, *137*, 22A530.

[51] S. K. Min, F. Agostini, E. K. U. Gross, *Phys. Rev. Lett.* **2015**, *115*, 073001.

[52] F. Agostini, S. K. Min, A. Abedi, E. K. U. Gross, *J. Chem. Theory Comput.* **2016**, *12*, 2127.

[53] S. K. Min, F. Agostini, I. Tavernelli, E. K. U. Gross, *J. Phys. Chem. Lett.* **2017**, *8*, 3048.

[54] B. F. E. Curchod, F. Agostini, I. Tavernelli, *Eur. Phys. J. B.* **2018**, *91*, 168.

[55] G. H. Gossel, F. Agostini, N. T. Maitra, *J. Chem. Theory Comput.* **2018**, *14*, 4513.

[56] A. Carof, S. Giannini, J. Blumberger, *J. Chem. Phys.* **2017**, *147*, 214113.

[57] S. Hammes-Schiffer, J. C. Tully, *J. Chem. Phys.* **1994**, *101*, 4657.

[58] J. Pittner, H. Lischka, M. Barbatti, *Chem. Phys.* **2009**, *356*, 147.

[59] I. G. Ryabinkin, J. Nagesh, A. F. Izmaylov, *J. Phys. Chem. Lett.* **2015**, *6*, 4200.

[60] J.-K. Ha, K. Kim, S. K. Min, *J. Chem. Theory Comput.* **2021**, *17*, 694.

[61] E. Marsili, M. Olivucci, D. Lauvergnat, F. Agostini, *J. Chem. Theory Comput.* **2020**, *16*, 6032.

[62] M. Filatov, S. K. Min, K. S. Kim, *J. Chem. Theory Comput.* **2018**, *14*, 4499.

[63] M. Filatov, M. Paolino, S. K. Min, K. S. Kim, *J. Phys. Chem. Lett.* **2018**, *9*, 4995.

[64] M. Filatov, S. K. Min, C. H. Choi, *Phys. Chem. Chem. Phys.* **2019**, *21*, 2489.

[65] M. Filatov, M. Paolino, S. K. Min, C. H. Choi, *Chem. Commun.* **2019**, *55*, 5247.

[66] L. Verlet, *Phys. Rev.* **1967**, *159*, 98.

[67] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes*, 3rd ed., The Cambridge University Press, New York **2007**.

[68] G. Granucci, M. Persico, A. Toniolo, *J. Chem. Phys.* **2001**, *114*, 10608.

[69] H. Lischka, T. Müller, P. G. Szalay, I. Shavitt, R. M. Pitzer, R. Shepard, *WIREs Comput. Mol. Sci.* **2011**, *1*, 191.

[70] M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheese-Man, G. Scalmani, V. Barone, B. Mennucci, G. A. Petersson, H. Nakatsuji, M. Caricato, X. Li, H. P. Hratchian, A. F. Izmaylov, J. Bloino, G. Zheng, J. L. Sonnenberg, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, T. Vreven, J. A. Montgomery, J. E. Peralta, F. Ogliaro, M. Bearpark, J. Heyd, E. Brothers, K. N. Kudin, V. N. Staroverov, R. Kobayashi, J. Normand, K. Raghavachari, A. Rendell, J. C. Burant, S. S. Iyengar, J. Tomasi, M. Cossi, N. Rega, M. Millam, M. Klene, J. E. Knox, J. B. Cross, V. Bakken, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, R. L. Martin, K. Morokuma, V. G. Zakrzewski, G. A. Voth, P. Salvador, J. J. Dannenberg, S. Dapprich, A. D. Daniels, O. Farkas, J. B. Foresman, J. V. Ortiz, J. Cioslowski, D. J. Fox, *Gaussian 09 Revision A.02*, Gaussian Inc, Wallingford CT **2009**.

[71] S. Seritan, C. Bannwarth, B. S. Fales, E. G. Hohenstein, S. I. L. Kokkila-Schumacher, N. Luehr, J. W. Snyder, C. Song, A. V. Titov, I. S. Ufimtsev, T. J. Martínez, *J. Chem. Phys.* **2020**, *152*, 224110.

[72] B. Hourahine, B. Aradi, V. Blum, F. Bonafé, A. Buccheri, C. Camacho, C. Cevallos, M. Y. Deshaye, T. Dumitrică, A. Dominguez, S. Ehlert, M. Elstner, T. van der Heide, J. Hermann, S. Irle, J. J. Kranz, C. Köhler, T. Kowalczyk, T. Kubař, I. S. Lee, V. Lutsker, R. J. Maurer, S. K. Min, I. Mitchell, C. Negre, T. A. Niehaus, A. M. N. Niklasson, A. J. Page, A. Pecchia, G. Penazzi, M. P. Persson, J. Řezáč, C. G. Sánchez, M. Sternberg, M. Stöhr, F. Stuckenberg, A. Tkatchenko, V. W.-Z. Yu, T. Frauenheim, *J. Chem. Phys.* **2020**, *152*, 124101.

[73] J. A. Rackers, Z. Wang, C. Lu, M. L. Laury, L. Lagardère, M. J. Schnieders, J.-P. Piquemal, P. Ren, J. W. Ponder, *J. Chem. Theory Comput.* **2018**, *14*, 5273.

[74] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak, *J. Chem. Phys.* **1984**, *81*, 3684.

[75] G. J. Martyna, M. E. Tuckerman, D. J. Tobias, M. L. Klein, *Mol. Phys.* **1996**, *87*, 1117.

[76] X. Pang, X. Cui, D. Hu, C. Jiang, D. Zhao, Z. Lan, F. Li, *J. Phys. Chem. A* **2017**, *121*, 1240.

[77] V. Q. Vuong, J. Akkarapattiakal Kuriappan, M. Kubillus, J. J. Kranz, T. Mast, T. A. Niehaus, S. Irle, M. Elstner, *J. Chem. Theory Comput.* **2018**, *14*, 115.

[78] I. S. Lee, M. Filatov, S. K. Min, *J. Chem. Theory Comput*. **2019**, *15*, 3021.

[79] A. Domínguez, B. Aradi, T. Frauenheim, V. Lutsker, T. A. Niehaus, *J. Chem. Theory Comput*. **2013**, *9*, 4901.

[80] J. Conyard, A. Cnossen, W. R. Browne, B. L. Feringa, S. R. Meech, *J. Am. Chem. Soc*. **2014**, *136*, 9692.

[81] L. Pedraza-González, L. De Vico, M. D. C. Marin, F. Fanelli, M. Olivucci, *J. Chem. Theory Comput*. **2019**, *15*, 3134.

[82] C. Brunken, M. Reiher, *J. Chem. Theory Comput*. **2021**, *17*, 3797.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: I. S. Lee, J.-K. Ha, D. Han, T. I. Kim, S. W. Moon, S. K. Min, *J Comput Chem* **2021**, *42*(24), 1755. https://doi.org/10.1002/jcc.26711