Non-adiabatic EXcited-state Molecular Dynamics (NEXMD) Reference Manual

Version 1.1

Contents

1 Contributors (Alphabetical Order)

1.1 Principal Investigators

- Sebastian Fernandez-Alberti (UNQui)
  sfalberti@gmail.com
- Adrian E. Roitberg (UF)
  roitberg@ufl.edu
- Sergei Tretiak (LANL)
  serg@lanl.gov

1.2 Developers

- Josiah A. Bjorgaard (LANL)
  jbjorgaard@lanl.gov
- Benjamin Nebgen (LANL)
  bnebgen@lanl.gov
- Tammie R. Nelson (LANL)
  tammien@lanl.gov
- Andrew E. Sifain (LANL & USC)
  sifain@usc.edu
- Dustin A. Tracy (UF)
  dtracy@ufl.edu
- Kirill Velizhanin (LANL)
  kirill@lanl.gov
- Alexander J. White (LANL)
  alwhite@lanl.gov

2 Introduction

The Born–Oppenheimer (or adiabatic) approximation assumes that electronic and nuclear degrees of freedom are essentially decoupled and evolve on different time scales. In other words, nuclei adjust very slowly to changes in electronic structure. Throughout the dynamics, however, excited-state energies may become closely-spaced, so that even the smallest changes in nuclear geometry can greatly affect excited-state properties. In these cases, nuclear-electronic coupling becomes strong and the Born–Oppenheimer approximation is no longer valid. Instead, the chemical system of interest may choose to transition between excited states and evolve non-adiabatically. One of the primary uses of the Non-adiabatic EXcited-State Molecular Dynamics (`NEXMD`) software is to simulate the non-dynamics of molecules after photo-excitation. The goal being to explain processes such as excited-state lifetimes, charge and energy transfer, photoisomerization, photodissociation, nuclear vibrational dependences on excited-state dynamics, etc. The program is capable of simulating molecules on the order of 100 atoms and for time-scales of up to 50 ps. The purpose of this document is not to give an in-depth discussion of the theory or methods implemented in `NEXMD`, but rather, it is to provide a comprehensive introduction on how to use the `NEXMD` program. For the former, we refer readers to Refs. [1, 2]. A number of papers using development versions of the code have been published, and can be catagorized under methods development (Refs. [3–11]) and applications (Refs. [12–31]). Included in the Appendix A.6 of this document, are all benchmark tests of the most current version of the code. This section is updated during every release.

3 Compiling NEXMD

NEXMD has been tested to work with both GNU and Intel compilers. To compile with a given version, first add the compiler to your path. For BASH, this is accomplished by:

```
PATH = $PATH:<directory of compiler>
```

Once your selected compiler is added to your path, run one of the two following make commands to compile, depending on your desired compiler

```
Intel: make ic
GNU: make gnu
```

This will compile a version of the code that is not linked to any math libraries that you have installed, such as MKL. For a more advanced compilation, you may wish to use some of the other make options. Examples are shown within the Makefile. Please contact Benjamin Nebgen (bnebgen@lanl.gov) with any further questions regarding compilation and installation.

4 General Procedure

Figure 1 shows the general procedure of how to simulate non-adiabatic dynamics. A description of each step along with how they are executed in NEXMD will be discussed throughout this document.

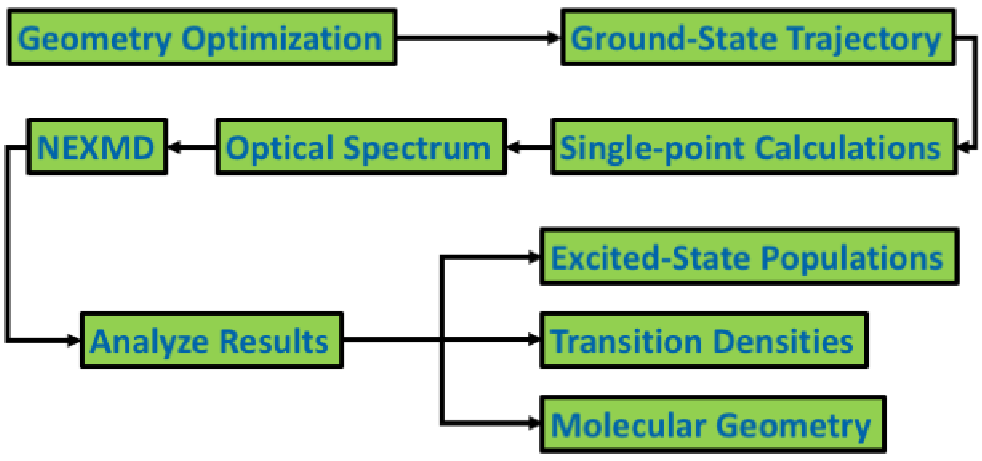


Figure 1: A schematic of the general procedure for simulating non-adiabatic dynamics.

5 The Input File

The contents of the input file for `NEXMD` can be categorized as follows: geometry optimization, ground-state parameters, excited-state parameters, solvent models, molecular dynamics, initial geometries and excited-state coefficients. An overview of all input parameters will be mentioned here, which should be sufficient for getting started with `NEXMD`. The input file will be discussed one section at a time. The numerical values in square brackets are default parameters.

The following section contains input parameters pertaining to geometry optimization.

```
&qmmm
   !***** Geometry Optimization
   maxcyc=0, ! Number of cycles for geometry optimization [0]
   ntpr=1, ! Print results every ntpr cycles [1]
   grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]
```

- `maxcyc` sets the maximum number of cycles for geometry optimization. If the number of cycles reaches `maxcyc`, an error message reads: `Maximum number of iterations reached without convergence`. Therefore, to optimize geometry, `maxcyc` must be set to a sufficiently large number greater than zero. If the geometry reaches the maximum number of cycles, you must review the situation carefully to determine if the input geometry was appropriate or if `grms_tol` (shown below) was set too low. A negative `maxcyc` is treated as `0`.

- `ntpr` sets how often results for geometry optimization are printed to the standard output file. The output of the iteration is written as `iter`, `energy` in eV, and `rms gradient` in eV per Å. The latter quantity is explained in `grms_tol`, shown below.

- `grms_tol` sets the convergence criteria for geometry optimization. The units for `grms_tol` are in eV per Å, so that a smaller `grms_tol` is a smaller change in energy per change in bond length (i.e. a tighter convergence criteria).

The following section contains input parameters pertaining to the ground-state calculation and its associated outputs.

```
   !***** Ground-State and Output Parameters
   qm_theory='AM1', ! Integral type, check Amber's SQM for more options [AM1]
   scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]
   verbosity=0, ! QM/MM output verbosity (0-minimum, 5-maximum)
   ! [1 for dynamics and optimization, 5 for others]
   printdipole=2, ! (0) Unrelaxed transitions, (1) Unrelaxed transitions plus
   ! total molecular, or (2) Unrelaxed/relaxed transitions plus
   ! total molecular [1 for dynamics and 2 for single-point]
   printbondorders=0, ! (0) No or (1) Yes [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   density_predict=0, ! (0) None, (1) Reversible MD,
   ! or (2) XL-BOMD [0] *** ALL ARE UNDER DEVELOPMENT, DO NOT USE ***
```

```
itrmax=300, ! Max SCF iterations for ground state
! (negative to ignore convergence) [300]
```

- `qm_theory` sets the semi-empirical model Hamiltonian. A list of available Hamiltonians can be found in Ref. [32]. The majority of studies with `NEXMD` have used Austin Model 1 (AM1).
- `scfconv` sets the convergence criteria for the self-consistent field (SCF) ground-state energy in eV. In other words, the user requests that the ground-state energy be determined to within `scfconv` eV.
- `verbosity` sets the level of printing of QM/MM related outputs. The outputs of each level can be found in Ref. [32]. The verbosity should be set to at least `1` in order to obtain ground-to-excited state oscillator strengths. This is important for single-point calculations and obtaining an optical spectrum, as described in Subsections 6.4 and 6.5, respectively.
- `printdipole` sets the printing level for dipole moments. There are three options to choose from which are: `(0)`, ground-to-excited state transition dipole moments, `(1)`, same as `(0)` plus the total molecular dipole moment, and `(2)`, same as `(1)` plus relaxed and unrelaxed difference dipole moments.
- `itrmax` sets the maximum number of cycles for the ground-state SCF calculation. If the number of cycles reaches `itrmax`, the code will stop. A negative value for `itrmax` means the ground-state SCF calculation will go through |`itrmax`| cycles, regardless of whether or not the convergence criteria in `scfconv` has been met.

The following section contains input parameters pertaining to the excited-state calculation.

```
!***** Excited-State Parameters
exst_method=1, ! CIS (1) or RPA (2) [1]
dav_guess=1, ! Restart Davidson from (0) Scratch, (1) Previous,
! or (2) XL-BOMD [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]
ftol1=1.0d-8, ! Acceptance tolerance for residual norm [1.0d-5]
! *** UNDER DEVELOPMENT, DO NOT USE ***
dav_maxcyc=200, ! Max cycles for Davidson diagonalization
! (negative to ignore convergence) [100]
printcharges=0, ! Print (1) or do not print (0) Mulliken charges of QM atoms [0]
calcxdens=.false., ! Print (.true.) or do not print (.false.)
! excited-to-excited transition dipole moments [.false.]
```

- `exst_method` sets the approximate excited-state wavefunction, which is used to compute excited-state properties. There are two options to choose from in `NEXMD`. The first is the configuration interactions singles (`CIS`) wavefunction and the other is the random phase approximation wavefunction (`RPA`). The RPA wavefunction is a slight extension of the CIS wavefunction and includes more electron correlation effects. Therefore, it is common for RPA to be more computationally demanding than CIS. The CIS wavefunction is stable and has been used for the majority of studies with `NEXMD`.

- `dav_guess` sets the initial guess for the Davidson algorithm. The Davidson algorithm is used to calculate excited-state eigenvalues and eigenvectors. One option is to start Davidson from `scratch`. However, this option may increase computation time. Another option is start Davidson from results of the `previous` calculation. The latter option should be used for realistic simulations.

- `ftol0` sets the convergence criteria on excited-state energies. In other words, the user requests that excited-state energies be determined to within `ftol0` eV.

- `dav_maxcyc` sets the maximum number of cycles for the Davidson algorithm. If the number of cycles exceeds `dav_maxcyc`, an error message will read: `Number of Davidson iterations exceeded, exiting.` A negative value for `dav_maxcyc` means the excited-state calculation will go through |`dav_maxcyc`| cycles, regardless of whether or not the convergence criteria in `ftol0` has been met.

- `printcharges` sets whether or not to print Mulliken charges of QM atoms to the standard output file.

- `calcxdens` sets whether or not to print excited-to-excited transition dipole moments. This option can only be set to `true` during single-point calculations. An error will occur if `calcxdens` is set to `true` during dynamics. A file called `muab.out` will be generated if `calcxdens=.true.`, which contains excited-to-excited transition dipoles in arbitrary units. The number of excited states included in `muab.out` depends on the number of excited states being propagated, which is controlled by `n_exc_states_propagate`, located in an upcoming section of the input file.

The following section contains input parameters pertaining to solvent models and external electric fields.

```
!***** Solvent Models and External Electric Fields
solvent_model=0, ! (0) None, (1) Linear response, (2) Vertical excitation,
! or (3) State-specific [0]
potential_type=1, ! (1) COSMO or (2) Onsager [1]
onsager_radius=2, ! Onsager radius, A (system dependent) [2]
ceps=10, ! Dielectric constant, unitless [10]
linmixparam=1 ! Linear mixing parameter for vertical excitation
! or state-specific SCF calculation [1]
cosmo_scf_ftol=1.0d-5, ! Vertical excitation or state-specific
! SCF tolerance, eV [1.0d-5]
doZ=.false. ! Use relaxed (.true.) or unrelaxed (.false) density for
! vertical excitation or state-specific COSMO or Onsager [.false.]
index_of_refraction=100, ! Dielectric constant for linear response
! solvent in excited state, unitless [100] *** UNDER DEVELOPMENT, DO NOT USE ***
EF=0, ! (0) None or (1) Electric field in ground and excited state [0]
```

```
   Ex=0, ! Electric field vector X, eV/A [0]
   Ey=0, ! Electric field vector Y, eV/A [0]
   Ez=0, ! Electric field vector Z, eV/A [0]
&endqmmm
```

- `solvent_model` sets the solvent model. A description of the models can be found in Refs. [8,9].
- `potential_type` sets the potential of the solvent model. The Onsager model assumes that the solute is placed in a spherical cavity inside the solvent. The latter is described as a homogeneous, polarizable medium of constant dielectric constant given by `ceps`. The solute dipole moment induces a dipole moment of opposite direction in the surrounding medium. Polarization of the medium in turn polarizes the charge distribution in the solvent. Treating this mutual polarization in a self-consistent manner leads to the Onsager reaction field model. COSMO (Conductor-like Screening Model) generalizes the Onsager potential where the cavity surface is defined by the shape of the solute. COSMO is a more complete description of the electrostatic interactions.
- `onsager_radius` defines the radius of the spherical cavity for the Onsager reaction field model.
- `ceps` sets the dielectric constant of the solvent. A list of solvents and their dielectric constants can be found in Ref. [33]. Be sure to reference Ref. [33].
- `linmixparam` sets the degree to which the last two SCF iterations are mixed. The mixed solution is used as an input for the following SCF iteration. The goal of introducing `linmixparam` is to significantly reduce the high cost of finding the SCF solution by inputting a solvent potential into the current iteration that is extrapolated from previous iterations. See Refs. [8,9] for more information.
- `cosmo_scf_ftol` sets the convergence criteria of the vertical excitation or state-specific solvent model SCF calculation. In other words, the user requests that the energy be determined to within `cosmo_scf_ftol` eV. The `cosmo_scf_ftol` flag sets the tolerance for both the Onsager and COSMO potentials.
- `EF` sets whether or not there is an external electric field applied to the system.
- `Ex`, `Ey`, `Ez` sets the magnitude and direction of the external electric field in the x, y, and z axes, respectively. The user must also set `EF to 1` if an external electric field is desired in the simulation.

General Note: The default tolerances within the section enclosed by `&qmmm` and `&endqmmm` are within accepted levels of convergence. However, these values may be increased depending on the size of the system or time of simulation.

The following section contains general input parameters pertaining to molecular dynamics.

```
&moldyn
   !***** General Parameters
   natoms=12, ! Number of atoms
   ! (must be equal to the number of atoms in system)
   rnd_seed=19345, ! Seed for the random number generator
   bo_dynamics_flag=0, ! (0) Non-BO or (1) BO [1]
   exc_state_init=6, ! Initial excited state (0 - ground state) [0]
   n_exc_states_propagate=8, ! Number of excited states [0]
```

- `natoms` sets the number of atoms in the system being studied. This number is important for memory allocation and must be equal to or greater than the number of atoms in the system.

- `rnd_seed` sets the seed for the random number generator. For each `rnd_seed`, there is a well-defined sequence of random numbers. For non-adiabatic ensemble simulations, `rnd_seed` must be different from one trajectory to another. This ensures the stochastic nature of the simulation, which is important for both the nuclear Langevin dynamics (i.e. coupling of the system to a heat bath) and the surface hopping algorithm, [34] which governs electronic transitions between electronic states (i.e. non-adiabatic dynamics). The details of how `rnd_seed` is chosen for non-adiabatic dynamics will be discussed in Subsection 6.6.

- `bo_dynamics` sets whether the dynamics is non-Born–Oppenheimer (non-adiabatic) or Born–Oppenheimer (adiabatic). If the simulation is non-adiabatic, this typically means the user is running an ensemble of trajectories. This may also be the case for an adiabatic simulation, depending on the study.

- `exc_state_init` sets the initial excited-state of the system. For a non-adiabatic ensemble simulation, a distribution of initial excited-states is needed to model a photoexcited wavepacket of different nuclear geometries. Therefore, `exc_state_init` may be different from one trajectory to another. The details of how `exc_state_init` is chosen for non-adiabatic dynamics will be discussed in Subsection 6.6.

- `n_exc_states_propagate` sets the total number of excited-states to be propagated in the dynamics. The user must be careful not to include unnecessary higher-energy states if it is unlikely for the system to access those states, as this may greatly increase computation time. The number of excited-states to include in the simulation is determined by the electronic structure and optical spectrum of the system, which will be discussed in Subsections 6.4 and 6.5.

The following section contains more inputs for molecular dynamics.

```
!***** Dynamics Parameters
time_init=0.0, ! Initial time, fs [0.0]
time_step=0.1, ! Time step, fs [0.1]
n_class_steps=10000, ! Number of classical steps [1]
n_quant_steps=4, ! Number of quantum steps for each classical step [4]
```

```
moldyn_deriv_flag=1, ! (0) None, (1) Analytical, or (2) Numerical [1]
num_deriv_step=1.0d-3, ! Displacement for numerical derivatives, A [1.0d-3]
rk_tolerance=1.0d-7, ! Tolerance for the Runge-Kutta propagator [1.0d-7]
```

- `time_init` sets the initial time of the trajectory. If the trajectory has not yet begun, then this number is set 0.0 fs. However, it is common for a trajectory to be restarted from where it left off. In these cases, the initial time depends on when the previous simulation has ended. Restart input files will be discussed in Section 10.

- `time_step` sets the classical time-step of the trajectory. This is the time-step at which nuclear degrees of freedom are integrated.

- `n_class_steps` sets the total number of classical time-steps in the trajectory. An error message will read: `You must run dynamics (n_class_steps > 0) or geometry optimization (maxcyc > 0). Running both simultaneously is not possible.`

- `n_quant_steps` sets the number of quantum steps per classical step. The nuclear degrees of freedom are integrated with the Velocity Verlet (VV) algorithm, while the electronic degrees of freedom (i.e. the quantum coefficients) are integrated with the Runge-Kutta (RK) algorithm. The nuclear dynamics are more computationally stable than the quantum coefficients. The VV algorithm, while less computationally demanding than RK, is sufficient for nuclear dynamics. Quantum coefficients are more susceptible to computational instabilities and require a more rigorous method for integrating their equations of motion (i.e. the Schrödinger equation).

- `moldyn_deriv_flag` sets how gradients are calculated. The options here are numerical or analytical. Generally, analytical derivatives should be used because they are more computationally stable and less computationally expensive than numerical derivatives.

- `num_deriv_step` sets the derivative step-size in units of angstroms (Å) when the `moldyn_deriv_flag` is set to `numerical`.

- `rk_tolerance` sets the convergence criteria on the quantum coefficients in arbitrary units. The smaller the `rk_tolerance`, the tighter the convergence criteria. The user should be careful not to set this quantity too low, as RK may not be able to handle very low tolerance levels. In these cases, the output file will show an error message that reads, `RK tolerance may be too low`. The `rk_tolerance` should be increased and trajectories should be restarted from 0.0 fs.

The following section contains input parameters pertaining to non-adiabatic dynamics.

```
!***** Non-Adiabatic Parameters
decoher_type=2, ! Type of decoherence: Reinitialize (0) Never,
! (1) At successful hops, (2) At successful plus frustrated hops...
! (3) Persico/Granucci, or (4) Truhlar [2]
! *** (3) AND (4) ARE UNDER DEVELOPMENT, DO NOT USE ***
decoher_e0=0.0, ! Decoherence parameter E0, Hartrees [0.1]
! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
```

```
decoher_c=0.0, ! Decoherence parameter C, unitless [0.1]
! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
dotrivial=1, ! Do unavoided (trivial) crossing routine (1) or not (0) [1]
quant_step_reduction_factor=2.5d-2, ! Quantum step reduction factor [2.5d-2]
```

- `decoher_type` sets the method for decoherence. The surface hopping method treats nuclear degrees of freedom classically and electronic degrees of freedom quantum mechanically. Due to the classical treatment of nuclei, there is an overcoherence between quantum states. In realistic simulations, there should always be some form of decoherence. Method `0` does not introduce any form of decoherence and should only be used for code testing or benchmarking purposes. By reinitializing the quantum coefficients after hops, this introduces a form of decoherence that is instantaneous. Method `1` collapses the wavefunction at all successful hops, whereas `2` collapses the wavefunction at all successful plus frustrated hops. Frustrated hops are those that were unable to satisfy energy conservation and were rejected. The methods defined by `3` and `4` are categorized as energy-based decoherence corrections, where coefficients are rescaled at classical steps, such that the coherence between two states decays as a function of their energy gap (i.e. a larger energy gap means faster decoherence). It has been shown that the simple collapse method is sufficient for introducing decoherence. [5] In general, method `2` should be used for realistic simulations.

- `decoher_e0` sets the value of E_0 in units of Hartrees in the following expression for the decoherence time,

$$\tau_{\beta\alpha} = \frac{\hbar}{\left|E_\beta(t) - E_\alpha(t)\right|}\left(C + \frac{E_0}{\left(\mathbf{P}\cdot\hat{d}_{\beta\alpha}\right)^2/2\mu}\right), \tag{1}$$

  where $\mathbf{P}\cdot\hat{d}_{\beta\alpha}$ is momentum in the direction of non-adiabatic coupling, $E_i(t)$ is the energy of the PES associated to state $|i\rangle$, and μ is the reduced mass. In Grannuci's formalism, [35, 36] the kinetic energy term, $\left(\mathbf{P}\cdot\hat{d}_{\beta\alpha}\right)^2/2\mu$, is replaced by the *total* kinetic energy, which reduces computational cost since non-adiabatic coupling vector can be ignored. Truhlar's formalism can be found in Refs. [37, 38].

- `decoher_c` sets the value of the unitless parameter, C, in Eq. (1).

- `dotrivial` sets whether or not to reduce the time-step in the vicinity of trivial crossings. See description under `quant_step_reduction_factor` for more details. Trivial crossings are identified by the method described in Ref. [4].

- `quant_step_reduction_factor` sets how much to reduce the quantum step in the vicinity of an unavoided or trivial crossing. At trivial crossings, the energy gap between the adjacent states is vanishingly small. The coupling between these states is essentially localized to the spatial *point* at which they are degenerate. Therefore, in order to resolve the non-adiabatic coupling between these states and determine whether or not an electronic transition occurs, the time-step must be reduced. It is defined as quant_step_reduction_factor $\times$ quant_time_step, where quant_time_step is determined by time_step and n_quant_steps.

The following section contains input parameters pertaining to the thermostat.

```
!***** Thermostat Parameters
therm_type=1, ! Thermostat type: (0) Newtonian, (1) Langevin,
! or (2) Berendsen [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
therm_temperature=300, ! Thermostat temperature, K [300]
therm_friction=20, ! Thermostat friction coefficient, 1/ps [20]
berendsen_relax_const=0.4, ! Bath relaxation constant for Berendsen
! thermostat, ps [0.4] *** UNDER DEVELOPMENT, DO NOT USE ***
heating=0, ! Equilibrated (0) or heating (1) [0]
! *** UNDER DEVELOPMENT, DO NOT USE ***
heating_steps_per_degree=100, ! Number of steps per degree
! *** UNDER DEVELOPMENT, DO NOT USE ***
! during heating [100] *** UNDER DEVELOPMENT, DO NOT USE ***
```

- therm_type sets the type of thermostat to be used in the simulation. This determines the equation of motion governing nuclear dynamics. There are three options for this input, one of which is Newtonian. This is the same as introducing no thermostat, which should only be used for code testing or benchmarking purposes. The other two options are Langevin and Berendsen. For the latter, velocities are rescaled to attain the user-defined temperature. This suppresses thermal fluctuations, which is not consistent with the canonical ensemble. The Langevin equation of motion is a stochastic differential equation that introduces terms for viscosity and a Gaussian random force that controls temperature in such a way that obeys the canonical ensemble. In realistic simulations, the thermostat should be set to Langevin.
- therm_temperature sets the temperature of the thermostat in units of Kelvin (K).
- therm_friction sets the friction parameter for the Langevin thermostat in units of inverse picoseconds (ps^{-1}). This input generally depends on the viscosity of the solvent that is being modeled, however, in most cases, the default parameter should be used.
- berendsen_relax_const sets the relaxation time constant of the Berendsen thermostat. The velocities of the nuclei in the system are rescaled at a rate given by inverse berendsen_relax_const in order to reach thermal equilibrium at the user-defined temperature in therm_temperature.

The following section contains input parameters pertaining to output data.

```
!***** Output & Log Parameters
verbosity=2, ! NEXMD output verbosity (0-minimum, 3-maximum)
! [2 for dynamics, 3 for optimization and single-point]
out_data_steps=1, ! Number of steps to write data [1]
out_coords_steps=10, ! Number of steps to write the restart file [10]
```

```
   out_data_cube=0, ! Write (1) or do not write (0) view files to generate cubes [0]
   out_count_init=0, ! Initial count for view files [0]
&endmoldyn
```

- `verbosity` sets the level of printing of NEXMD related outputs.
- `out_data_steps` sets number of steps to write data. For example, if `time_step=0.1` and `out_data_steps=2`, data will be written to output files every 0.2 fs.
- `out_coords_steps` sets the number of steps to write coordinates, velocities, and quantum coefficients. The output files for these quantities are in `coords.xyz`, `velocity.xyz`, and `coefficient.out`, respectively. The rate at which these data are written to the output files also depends on `out_data_steps`. For example, if `time_step=0.1`, `out_data_steps=2`, and `out_coords_steps=2`, these data will be written every 0.4 fs. In general, coordinates, velocities, and quantum coefficients are written to their respective output files every `time_step` $\times$ `out_data_steps` $\times$ `out_coords_steps` fs.
- `out_data_cube` sets whether or not to generate `.DATA` files. These files are later used to generate cube files. If `out_data_cube` is set to `1`, `.DATA` files are generated for every excited state and for every time step. For example, the file `view0003-0007.DATA` refers to the 3rd time step and 7th excited state.
- `out_count_init` sets the initial time step of the `.DATA` files. At the beginning of a trajectory, `out_count_init` should be set to `0`. During generation of restart input files, `out_count_init` will be changed accordingly.

The following section contains input parameters pertaining to the coordinates and velocities of the atoms that constitute the molecule being studied, as well as its excited-state coefficients.

```
&coord
  6      -7.9798271101     0.6776918081    -0.0532285388
  6      -7.0849928010     1.7602597759     0.0294961792
  6      -5.7058415294     1.5490364812     0.0312760931
  6      -5.2231419594     0.2195333448    -0.0446043010
  6      -6.1050960756    -0.8685564920     0.0220869421
  6      -7.5344099241    -0.6444487634     0.0248126135
  1      -9.0268081830     0.8587716724    -0.0794794940
  1      -7.4774606514     2.7566353436     0.1635393862
  1      -5.0939335779     2.4479885163     0.1481876938
  1      -4.1292016456     0.0999373674    -0.1580811639
  1      -5.6916991654    -1.8878557992     0.1151090966
  1      -8.2838388636    -1.4313798704     0.1051927200
&endcoord

&veloc
     3.3718248255   -5.6032885851   -1.1970845430
     2.5106648755    2.0978837936   -1.0696411897
```

```
    -5.9135180273  -3.7505826950   1.1689299883
     7.7194332369   4.8702351843   0.6576546539
    -7.1851218597  -2.0113572464  -0.6329683366
    -1.7276579899   0.3919019235  -0.0257452789
   -17.0279163131   9.9875659542   5.3513734186
    -4.7222747943  18.9640275032  11.9601977632
    10.9539809532  17.0164104392  -9.7113209726
    25.7548696749   2.2116651958  -0.5444198125
   -16.5303708308  -2.3313274630  -3.2147489925
    16.2776787026   2.2582071549   9.3572624705
&endveloc

&coeff
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  1.00  0.00
  0.00  0.00
  0.00  0.00
&endcoeff
```

- Between `&coord` and `&endcoord` are coordinates of the atoms in angstroms (Å). The first column identifies each atom with its atomic number. The following three columns are the x, y, and z coordinates, respectively. The coordinates shown above are those of a benzene molecule.

- Between `&veloc` and `&endveloc` are three columns showing velocities of each atom along the x, y, and z axes, respectively. The units of velocity are Å/fs.

- Between `&coeff` and `&endcoeff` are two columns showing the magnitude and phase of the excited-state coefficients, respectively. The number of rows should be equal to the number of excited states being propagated, `n_exc_states_propagate`. In this example, eight excited-states are being propagated and the system is initially fully excited in the sixth excited-state.

6 Modeling Photoinduced Dynamics and Preparing Input Files

6.1 Geometry Optimization

The first step before any simulation is to build the chemical system of interest in some molecular visualization tool such as `Jmol` or `Avogadro`. Then, the system's ground-state geometry must be determined according to some model quantum-chemistry. The geometry may be optimized within `NEXMD` using the available semi-empirical model Hamiltonians. An example input file is available in Subsection A.1. Jobs submissions are discussed in Section 9. Note: It may be necessary to use other levels of theory, such as density functional theory, to obtain an accurate ground-state geometry. This can be done with electronic structure packages such as `Gaussian`.

6.2 Ground-State Trajectory

The molecule is in an absolute ground-state after it is optimized in Subsection 6.1. However, in reality, it is in contact with a heated environment, which causes thermal motion due to coupling with vibrational modes. To some degree, there is no well-defined chemical structure at finite temperature. The procedure of Subsection 6.1 is still required, however, since the initial geometry cannot be far off from the true ground-state. The ground-state geometry of the molecule should be sampled from an ensemble of geometries. This forms the basis of trajectory-based dynamics, where each trajectory starts with coordinates and velocities sampled at different times throughout a ground-state trajectory. To obtain these geometries, a long ground-state trajectory of the molecule must be calculated, from which coordinates and velocities are outputted at a rate defined in `input.ceon`. Typically, the ground-state trajectory should be on the order of nanoseconds and the number of coordinates and velocities outputted should be greater than the intended number of non-adiabatic trajectories that are to be computed in Subsection 6.6. The latter depends on a number of factors including the size of the molecule, the number of excited-states being propagated, and the time of the simulation, but in most cases, this number will not exceed one to two thousand. An example input file is available in Subsection A.2.

6.3 General Outline of getexcited.py

After completing a long ground-state trajectory and obtaining a good ground-state sampling, the remaining steps shown in Figure 1 requires making and manipulating many input and output files in order to simulate and analyze the results from non-adiabatic dynamics. Many of these steps can be done with `getexcited.py`. The `getexcited.py` script is the main script that calls specific functions from the `getexcited_package`. This package is written in `Python 2.7` which should be loaded in the terminal with command `module load <version 2.7 of Python>`. To get the exact name, type `module avail` before loading the module. **Important**: Before executing `getexcited.py`, it must be opened and the variable `PATHTOPACK` must be set to the path to where the `getexcited_package` exists. The `getexcited_package` is typically located in a bin folder in your home directory. A general outline of how one may

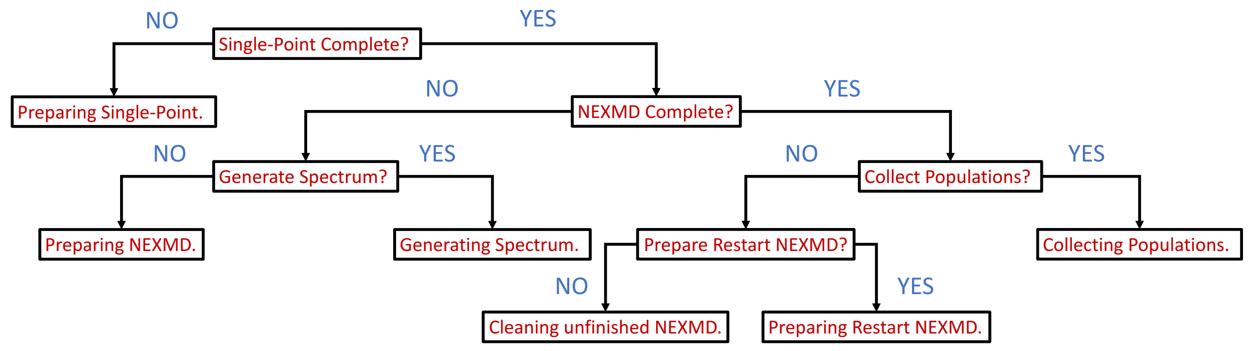


Figure 2: A schematic in YES or NO question format that proceeds through the different steps of modeling non-adiabatic dynamics.

proceed through steps of modeling non-adiabatic dynamics using `getexcited.py` is shown in Figure 2. Each of these steps will be discussed in this document. It is worth noting that more steps, not shown in Figure 2 and not discussed in this document, may be necessary depending on the study.

6.4 Single-Point Calculations

Throughout the ground-state trajectory, a set of coordinates and velocities will be written to `coords.xyz` and `velocity.xyz`. These files contain snap shots of the system taken throughout the ground-state trajectory at a rate which was specified in Subsection 6.2. After the ground-state trajectory is complete, single-point calculations at these geometries determine the electronic structure and optical spectrum of the system. It is important to consider many single-point calculations before generating an optical spectrum since the geometry of the system, and therefore electronic structure, changes from one time-step to another due to its interaction with the heat bath. It is good to plot temperature as a function of time from `temperature.out` of the ground-state trajectory to know at which time the system has equilibrated. This happens once the temperature oscillates around the set temperature of the bath (typically 300 K). *The ground-state sampling should be taken at times after the system has reached thermal equilibrium.*

To prepare input files for single-point calculations, use `getexcited.py`. First, create a directory for the single-point calculations (e.g. `singlepoint`). Second, create an input file with the name `header` inside `singlepoint`. In `header`, you must set the parameters which are to be the same for all single-point calculations. Be sure to set `verbosity` within `&qmmm` and `&endqmmm` to at least `1` for single-point calculations! The only varying parameter among the different single-point calculations is geometry. An example input file is available in Subsection A.3. In place of both nuclear coordinates and velocities, type in the flag, `nucl_coord_veloc`, as shown below.

```
⋮
&endmoldyn
```

```
nucl_coord_veloc

&coeff
⋮
```

The `getexcited.py` script searches for this flag and generates an input file for each geometry. Lastly, type `python getexcited.py`. An example of preparing input files for single-point calculations is shown below.

```
[sifain@wc-fe1 realistic]$ ls
getexcited.py gsdynamics singlepoint
[sifain@wc-fe1 realistic]$ cd singlepoint/
[sifain@wc-fe1 singlepoint]$ ls
header
[sifain@wc-fe1 singlepoint]$ cd ..
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 1
Preparing input files for single-point calculations.
Ground-state dynamics directory: gsdynamics
Output directory [e.g. singlepoint]: singlepoint
A total of 1001 coordinates files, ranging from 0.00 to
100.00 fs in increments of 0.10 fs, were found.
Enter requested range of the ground-state sampling by
coordinate files and the number of single-point calculations.
Input an array of the form [start, end, number]: [1,1000,500]
You have requested 500 evenly-spaced coordinate files in the range 1 to 1000.
Continue? Answer YES [1] or NO [0]: 1
Number of single-point calculations per NEXMD folder [e.g. 100]: 100
singlepoint/NEXMD1/0001
```

```
singlepoint/NEXMD1/0003
singlepoint/NEXMD1/0005
⋮
singlepoint/NEXMD5/0995
singlepoint/NEXMD5/0997
singlepoint/NEXMD5/0999
[sifain@wc-fe1 realistic]$ cd singlepoint/
[sifain@wc-fe1 singlepoint]$ ls
header NEXMD1 NEXMD2 NEXMD3 NEXMD4 NEXMD5
```

6.5 Optical Spectrum

It is important to generate an optical spectrum from single-point calculations to determine the energy at which to excite the system. To do so, use `getexcited.py`, which generates an average spectrum over all geometries (i.e. the sum of all spectra, divided by the number of geometries). The oscillator strength of each excitation energy is given a Gaussian lineshape with some user-defined width. This is a Gaussian-shaped Franck-Condon window. To be more explicit, the optical absorbance of each excitation energy is broadened by a Gaussian lineshape and is weighted by the oscillator strength at that energy,

$$A_e^i\left(\Omega\right) = f_{ge}^i\left(\Omega_e\right) \times \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{\left(\Omega_e - \Omega\right)^2}{2\sigma^2}\right] , \tag{2}$$

where A_e^i defines the contribution to the absorbance of the i^{th} geometry from excited-state $|e\rangle$. The absorbance over all excitation energies becomes

$$A^i\left(\Omega\right) = \sum_e A_e^i\left(\Omega\right) . \tag{3}$$

To obtain the combined optical spectrum, the spectra from all geometries are averaged,

$$A\left(\Omega\right) = \frac{1}{N}\sum_i A^i\left(\Omega\right) . \tag{4}$$

where N is the number of geometries. You may also choose a Lorentzian lineshape of the form

$$A_e^i\left(\Omega\right) = f_{ge}^i\left(\Omega_e\right) \times \frac{2}{\Gamma\pi}\frac{1}{1+4\left(\frac{\Omega-\Omega_{ge}}{\Gamma}\right)^2} , \tag{5}$$

where Γ is a full width at half maximum (FWHM).

One of three files will be generated in the current working directory if an optical spectrum is requested, `ceo.err`, `ceo_gauss.out` or `ceo_lorentz.out`. Shown in `ceo.err`, are directories of single-point calculations that are problematic. The latter two output files cannot be generated unless all single-point calculations are complete. There are two types of spectral lineshapes to choose from, Gaussian or Lorentzian. To obtain the spectrum, type `python getexcited.py`, which generates a set of questions, one of which is spectral broadening of the Franck-Condon window. An example of generating an optical spectrum is shown below.

```
[sifain@wc-fe1 realistic]$ ls
getexcited.py gsdynamics singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 2
Generating optical spectrum.
Single-point calculations directory: singlepoint
Checking energies and oscillator strengths. Please wait ...
Spectral lineshape? Answer GAUSSIAN [0] or LORENTZIAN [1]: 0
Spectral broadening (i.e. Gaussian standard deviation) in eV [e.g. 0.15]: 0.15
singlepoint/NEXMD1/1000
singlepoint/NEXMD1/1004
singlepoint/NEXMD1/1008
⋮
singlepoint/NEXMD5/4988
singlepoint/NEXMD5/4992
singlepoint/NEXMD5/4996
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics singlepoint
```

The columns in `ceo_gauss.out` or `ceo_lorentz.out` are the following: energy in eV, followed the spectra of all excited states, respectively. The last column is the total spectrum, which is the sum of all spectra from columns 1 to N, where N is the number of excited states. Example spectra are shown in Figure 3.

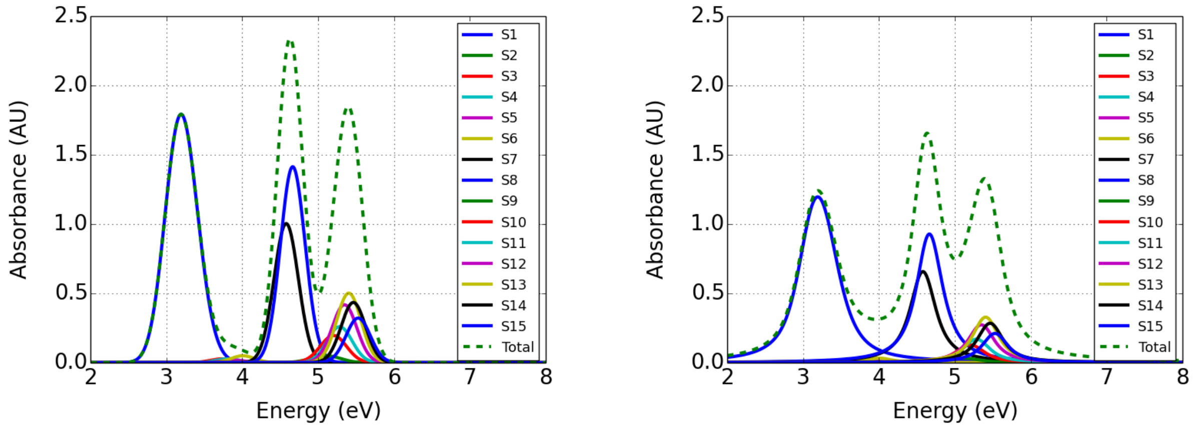


Figure 3: Absorption spectra of a Poly(p-phenylene vinylene) molecule. Each spectrum is an average over 1000 spectra obtained from 1000 single-point calculations. The spectra are generated with Gaussian lineshapes (left) and Lorentzian lineshapes (right).

6.6 NEXMD

Input files for NEXMD may be prepared once single-point calculations are complete, and you know at which energy to excite the system. First, create a directory for NEXMD (e.g. `nexmd`). Inside `nexmd`, place an input file with the name `header`. Similar to Subsection 6.4, you must set the parameters which are to be the same for all trajectories. An example input file is available in Subsection A.5. The four parameters that are different among the trajectories are: random seed, initial excited-state, initial nuclear coordinates and velocities, and initial quantum amplitudes and phase. In place of these parameters, insert their respective flags: `rnd_seed`, `exc_state_init_flag`, `nucl_coord_veloc`, and `quant_amp_phase`. The `getexcited.py` script searches for these flags and generates an input file for each initial geometry.

To prepare input files for NEXMD, type `python getexcited.py`. Among the questions asked is, `New random seeds? Answer YES [1] or NO [0]: `. By answering `YES`, you are requesting for a new list of random seeds. By answering `NO`, you are requesting to use a pre-generated list of random seeds. If you request the former, new random seeds will be used to create input files and the complete list of these random seeds will be generated in the `nexmd` directory with name `rseedslist`. If you request the latter, the path to the pre-generated list of random seeds must be supplied in the proceeding question. The pre-generated list of random seeds must strictly be a list of random seeds with no header or footer, and the number of random seeds must be at least equal to the number of trajectories requested. The pre-generated list of random seeds option may be important for code testing or benchmarking purposes. *It is important to keep in mind that different compilers may have different sequences of random numbers for a given random seed.*

You must supply both the excitation energy and spectral broadening of the Franck-Condon window. The latter quantity should be the same as that supplied in Subsection 6.5, which was used to generate the optical spectrum. These two quantities are used to determine `exc_state_init` and `quant_amp_phase`. For each initial geometry, excitation energies and oscillator strengths were determined from single-point calculations in Subsection 6.4. In order to determine the initial excited-state of each geometry upon photo-excitation, a Gaussian-shaped Franck-Condon window (centered at the user-defined excitation energy, Ω, with spectral broadening, σ) is multiplied by each oscillator strength, f_{eg},

$$P'\left(\Omega_e\right) = f_{ge}\left(\Omega_e\right) \times \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{\left(\Omega_e - \Omega\right)^2}{2\sigma^2}\right] . \tag{6}$$

Eq. (6) is an unnormalized probability of exciting the system at Ω_e with laser energy, Ω. To determine the probability, $P\left(\Omega_e\right)$, Eq. (6) must be divided by the norm such that,

$$P\left(\Omega_e\right) = \frac{P'\left(\Omega_e\right)}{\sum_e P'\left(\Omega_e\right)} . \tag{7}$$

The `getexcited.py` script then chooses a random number between zero and one and determines `exc_state_init` for each initial geometry. In `quant_amp_phase`, the system is set to fully populate `exc_state_init` (i.e. $c_e = 1.0$). All phase factors are initially set to zero. An example of preparing input files for NEXMD is shown below.

```
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd singlepoint
[sifain@wc-fe1 realistic]$ cd nexmd/
[sifain@wc-fe1 nexmd]$ ls
header
[sifain@wc-fe1 nexmd]$ cd ..
[sifain@wf-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 3
Preparing input files for NEXMD.
Ground-state dynamics directory: gsdynamics
Single-point calculations directory: singlepoint
Output directory [e.g. nexmd]: nexmd
```

```
A total of 1001 coordinate files, ranging from 0.00 to
100.00 fs in increments of 0.10 fs, were found.
Note: Only coordinate files used for single-point
calculations can be used for NEXMD.
How many trajectories for NEXMD? Enter a number no greater than 1000: 1000
Number of trajectories per NEXMD folder: 200
New random seeds? Answer YES [1] or NO [0]: 1
Spectral lineshape? Answer GAUSSIAN [0] or LORENTZIAN [1]: 0
Laser excitation energy in eV: 5.40
Spectral broadening (i.e. Gaussian standard deviation) in eV [e.g. 0.15]: 0.15
nexmd/NEXMD1/0001
nexmd/NEXMD1/0002
nexmd/NEXMD1/0003
⋮
nexmd/NEXMD5/0998
nexmd/NEXMD5/0999
nexmd/NEXMD5/1000
[sifain@wc-fe1 realistic]$ cd nexmd/
[sifain@wc-fe1 nexmd]$ ls
header NEXMD1 NEXMD2 NEXMD3 NEXMD4 NEXMD5 rseedslist
```

An example of preparing input files for NEXMD with a pre-generated list of random seeds is shown below.

```
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd rseedslist singlepoint
[sifain@wc-fe1 realistic]$ wc -l rseedslist
1000 rseedslist
[sifain@wf-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 3
Preparing input files for NEXMD.
```

```
Ground-state dynamics directory: gsdynamics
Single-point calculations directory: singlepoint
Output directory [e.g. nexmd]: nexmd
A total of 1001 coordinate files, ranging from 0.00 to
100.00 fs in increments of 0.10 fs, were found.
Note: Only coordinate files used for single-point
calculations can be used for NEXMD.
How many trajectories for NEXMD? Enter a number no greater than 1000: 1000
Number of trajectories per NEXMD folder: 200
New random seeds? Answer YES [1] or NO [0]: 0
Path to random-seeds list: rseedslist
Spectral lineshape? Answer GAUSSIAN [0] or LORENTZIAN [1]: 0
Laser excitation energy in eV: 5.40
Spectral broadening (i.e. Gaussian standard deviation) in eV [e.g. 0.15]: 0.15
nexmd/NEXMD1/0001
nexmd/NEXMD1/0002
nexmd/NEXMD1/0003
⋮
nexmd/NEXMD1/0998
nexmd/NEXMD1/0999
nexmd/NEXMD1/1000
[sifain@wc-fe1 realistic]$ cd nexmd/
[sifain@wc-fe1 nexmd]$ ls
header NEXMD1 NEXMD2 NEXMD3 NEXMD4 NEXMD5 rseedslist
```

6.7 Adiabatic Simulation from NEXMD Geometries

In many cases, the adiabatic dynamics on a low-lying surface can describe important processes, one such example is photo-isomerization observed in a *trans*-to-*cis* conformation change of a molecule. Upon photo-excitation, the molecule evolves on a high-level surface and then relaxes by exchanging energy to nuclear kinetic energy via transitions to lower-lying excited states. It is common for the molecule to reach its first excited state and evolve for a considerable amount of time before relaxing to its ground state (possibly in the form of radiative emission). This is what is known as adiabatic dynamics on the first excited state. Instead of a non-adiabatic simulation continuing on the first excited-state, it may be necessary to continue dynamics adiabatically. The advantages of doing so include, reducing computational cost by eliminating the propagation of unnecessary states and increasing the time-step of simulation in order to propagate for longer times. For this to happen, the final geometries of trajectories in the non-adiabatic simulation must be carried over to the adiabatic simulation. This can be done with `getexcited.py`. An example of starting an adiabatic simulation on the first excited-state using geometries from a non-adiabatic simulation is shown below. *It is important for the molecule to have fully reached the first excited state before beginning adiabatic dynamics.* See Subsection 7.1 to calculate populations. It is also important to remember to change `exc_state_init` and `n_exc_states_propagate` to `1` in `header` located in a newly-created directory when adiabatic dynamics on the first excited-state is desired. The coefficients between `&coeff` and `&endcoeff` should be changed to fully populate the first excited state.

```
[sifain@wc-fe1 nexmd_solvent]$ ls
adiabatic_S1  optimize  singlepoint  gsdynamics  getexcited.py  nexmd
[sifain@wc-fe1 nexmd_solvent]$ cd adiabatic_S1/
[sifain@wc-fe1 adiabatic_S1]$ ls
header
[sifain@wc-fe1 adiabatic_S1]$ cd ..
[sifain@wc-fe1 nexmd_solvent]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 4
Preparing input files for adiabatic dynamics with geometries taken from another simulation.
Ground-state dynamics directory: gsdynamics
NEXMD directory where geometries should be taken from: nexmd
NEXMD directory for new simulation: adiabatic_S1
The coordinates in nexmd began at 0.00 fs and were printed to coords.xyz every 1.00 fs.
New random seeds? Answer YES [1] or NO [0]: 1
At what time, in femtoseconds, should geometries be taken from nexmd? 1000
Finding coordinates and velocities from nexmd. Please wait ...
adiabatic_S1/NEXMD1/0041
adiabatic_S1/NEXMD1/0100
adiabatic_S1/NEXMD1/0159
⋮
adiabatic_S1/NEXMD5/29836
adiabatic_S1/NEXMD5/29895
adiabatic_S1/NEXMD5/29954
[sifain@wc-fe1 nexmd_solvent]$ cd adiabatic_S1/
[sifain@wc-fe1 adiabatic_S1]$ ls
header NEXMD1 NEXMD2 NEXMD3 NEXMD4 NEXMD5 rseedslist
```

7 Analyzing Results

7.1 Relaxation Rates

The relaxation rate of the system after photo-induced excitation is a common quantity of interest. This can be obtained by fitting excited-state populations as a function of time to some fitting function. One such example is an exponential function of the form $P(t) = A - B\exp[-k(t-t_0)]$, where $A - B$ is the initial population at time $t = t_0$, A is the final population as $t \to \infty$, and k is the relaxation rate. Other fitting functions may be better-suited depending on how the population evolves. In any case, you must first collect populations as a function of time from NEXMD simulations. To do so, use `getexcited.py`.

If collecting populations is requested, two files may be generated in the current working directory, `pop.err` and `pop.out`. Shown in `pop.err`, are directories of trajectories that did not complete within the user-defined time. The last column in `pop.err` shows the last time-step of the simulation. In `pop.out`, first column is time in femtoseconds, followed by the average population on each potential energy surface (PES), followed by the sum of all PES populations (i.e. which should be 1.0), followed by the average populations from quantum coefficients, followed by the sum of quantum populations (i.e. which should also be 1.0). `Completed trajectories` are those that finish within the user-defined time. `Excellent trajectories` are those that finish with the time defined in `header` located in the NEXMD directory (e.g. `nexmd`). Unlike Subsection 6.5, `pop.out` can be generated even if a set of trajectories did not complete within the user-defined time. However, it is highly recommended that a sufficient sampling size be obtained before analyzing relaxation rates. Incomplete trajectories can be restarted from their last time-steps. This is discussed in Section 10. An example of collecting populations is shown below.

```
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 5
Collecting populations.
NEXMD directory: nexmd
```

```
Collect populations up to what time in femtoseconds: 1000
nexmd/NEXMD1/0101 1000.00
nexmd/NEXMD1/0102 1000.00
nexmd/NEXMD1/0103 1000.00
⋮
nexmd/NEXMD4/0997 1000.00
nexmd/NEXMD4/0999 1000.00
nexmd/NEXMD4/1000 1000.00
Total Trajectories: 0400
Completed Trajectories: 0399
Excellent Trajectories: 0399
One or more trajectories did not finish within 1000.00 femtoseconds, check pop.err.
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd pop.err pop.out singlepoint
```

Once `pop.out` is obtained, excited-state populations may be plotted. Select states may be fit to fitting functions to determine rates. An example is shown in Figure 4.

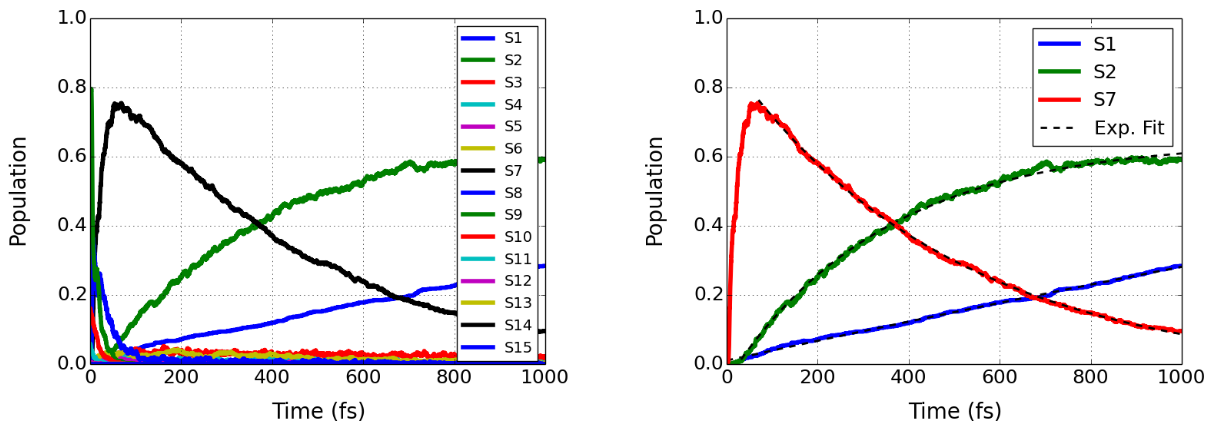


Figure 4: Left: Populations of all 15 excited-states from surface hopping in a Poly(p-phenylene vinylene) molecule. Right: Fitted populations of the states of interest. In this case, exponential fits were made to S1, S2, and S7.

The parameters of the fits are shown below.

```
S1: [A, B, k] = [2.27824e+00, 2.26779e+00, 1.27228e-04]
S2: [A, B, k] = [6.47593e-01, 6.95430e-01, 2.90113e-03]
S7: [A, B, k] = [-3.36844e-02, -7.97627e-01, 2.03588e-03]
```

7.2 PESs and NACTs

Analyzing excited-state lifetimes may involve explicity looking into the coupling between states, given by non-adiabatic coupling terms (NACTs). They are of the form $\dot{\mathbf{R}} \cdot \mathbf{d}_{\alpha\beta}$, where $\dot{\mathbf{R}}$ is velocity and $\mathbf{d}_{\alpha\beta}$ is the non-adiabatic coupling vector between states $|\alpha\rangle$ and $|\beta\rangle$. The latter can be expressed as

$$\mathbf{d}_{\alpha\beta} = \frac{\langle \alpha | \nabla H | \beta \rangle}{E_\beta - E_\alpha} \,, \tag{8}$$

where ∇H is the gradient of the Hamiltonian and E_i is the potential energy surface (PES) corresponding to state $|i\rangle$. Eq. (8) shows how non-adiabatic coupling between two states is inversely proportional to the energy gap between those states. As $|E_\beta - E_\alpha|$ becomes small, $|\mathbf{d}_{\alpha\beta}|$ becomes large and the system may transition between these excited-states. The strength of non-adiabatic coupling largely depends on energy gap, making it a useful parameter to analyze.

There are several files that can be generated if collecting NACTs and/or PESs are requested. You may choose to request PESs and NACTs at all time-steps or only at time-steps where hops occur. If the latter is requested, two files can be generated, `nact_hop_ensemble.out` and `pes_hop_ensemble.out`. In `nact_hop_ensemble.out`, columns from left to right are: directory of trajectory, excited-state at time t, excited-state at time $t + \Delta t$, and time in fs. The remaining columns are NACTs, which are consecutive rows of the upper triangle of the NACT matrix. For example, for a 2-state system, the indices of these terms would be 11, 12, and 22, respectively. In `pes_hop_ensemble.out`, the first four columns are the same as those in `nact_hop_ensemble.out`. The remaining columns are PESs of the ground state followed by all the excited-state PESs in units of eV. If data at all time-steps are requested, two files can be generated, `nact_raw_ensemble.out` and `pes_raw_ensemble.out`. The format of these files are the same as those of `nact_hop_ensemble.out` and `pes_hop_ensemble.out`, respectively, with the exception that the first column is trajectory directory. A specific time of collection may be set for cases in which the user is only interested in data up to some time less than the total simulation time. An example of collecting NACTs and PESs is shown below.

```
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd pop.out singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools
```

```
Enter the number corresponding to the desired task: 6
Collecting PESs and NACTs.
NEXMD directory: nexmd
Collect PESs and NACTs up to what time in femtoseconds: 1000
Collect PESs [1], NACTs [2], or BOTH [3]: 3
Collect PESs and NACTs from All TIME-STEPS [1] or HOPS ONLY [2]: 2
nexmd/NEXMD1/1000 1000.00
nexmd/NEXMD1/1004 1000.00
nexmd/NEXMD1/1008 1000.00
⋮
nexmd/NEXMD5/4988 1000.00
nexmd/NEXMD5/4992 1000.00
nexmd/NEXMD5/4996 1000.00
Total Trajectories: 1000
Completed Trajectories: 1000
Excellent Trajectories: 1000
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nact_hop_ensemble.out nexmd pes_hop_ensemble.out
pop.out singlepoint
```

An example histogram from the data in `pes_hop_ensemble.out` is shown in Figure 5. In this

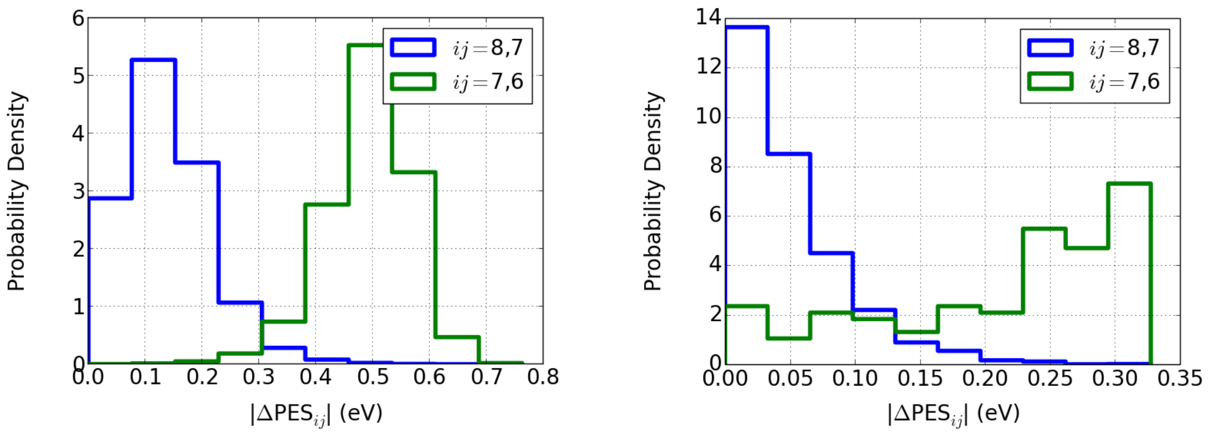
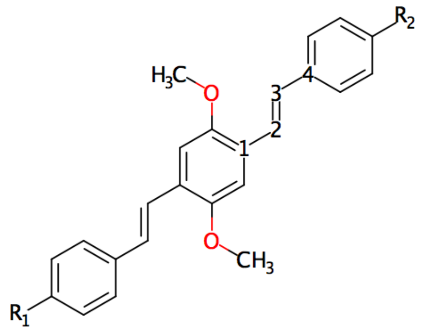


Figure 5: (Left) Histogrammed data of energy gaps, $|\Delta\text{PES}_{ij}| = |E_j - E_i|$ for $ij = \{87, 76\}$, at all time-steps in a 1000 trajectory ensemble, where each trajectory is 1.0 ps in length. (Right) Histogrammed data of the same energy gaps, but only at hops $8 \to 7$ and $7 \to 6$, respectively. The number of hops $8 \to 7$ and $7 \to 6$ are 3044 and 573, respectively.

example, hops from $8 \to 7$ and $7 \to 6$ as a function of their energy gap are histogrammed. The number of hops $8 \to 7$ is 3044, whereas the number of hops $7 \to 6$ is 573. The energy gap, $|\Delta\text{PES}_{87}|$, is noticeably smaller than $|\Delta\text{PES}_{87}|$, which is likely correlated to the large difference in the number of hops.

7.3 Geometry Analysis

There may be conformational changes of the molecule due to photoinduced dynamics. This can be analyzed as a function of a time. There are several geometrical quantities that can be calculated with `getexcited.py`, which include dihedrals, bond lengths, and bond length alternations (BLA). *Dihedral:* The dihedral is calculated from four atoms, where the first three atoms define one plane and the last three atoms define the other plane. The dihedral is defined as the positive angle between these planes and ranges from 0 to 180 degrees. *Bond length:* The bond length is calculated using Pythagorean theorem. *BLA:* BLA is calculated from four atoms giving three consecutive bond lengths, where the first and third have the same bond order and the second has a different bond order. The BLA is defined as $(d_1 + d_3)/2 - d_2$, where d_i is the i^{th} bond length. An example of calculating the mean dihedral angle as a function of time over all trajectories is shown below. The four atoms defining the dihedral are shown below.



```
[sifain@wc-fe1 realistic]$ ls
getexcited.py gsdynamics nexmd singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 9
```

```
Select a task from the following list:

[1] Calculate a dihedral angle
[2] Calculate a bond length
[3] Calculate a bond length alternation

Enter the number corresponding to the desired task: 1
Calculating a dihedral angle as a function of time.
Calculate a dihedral along one trajectory or an ensemble of trajectories?
Answer ONE [1] or ENSEMBLE [0]: 0
Ensemble directory [e.g. nexmd]: nexmd_opt
Ouput mean dihedral in time or output dihedrals at all time-steps and trajectories?
Answer MEAN [0] or ALL [1]: 0
Calculate dihedral up to what time in femtoseconds?
Note that averaged results will only include trajectories that are complete up to this time: 1
Input the line numbers labeling the coordinates of the four atoms.
Input an array of the form [ .., .., .., .. ]: [1,2,3,4]
nexmd/NEXMD1/1000 1000.00
nexmd/NEXMD1/1004 1000.00
nexmd/NEXMD1/1008 1000.00
⋮
nexmd/NEXMD5/4988 1000.00
nexmd/NEXMD5/4992 1000.00
nexmd/NEXMD5/4996 1000.00
Total Trajectories: 1000
Completed Trajectories: 1000
Excellent Trajectories: 1000
[sifain@wc-fe1 realistic]$ ls
dihedral.out getexcited.py gsdynamics nexmd singlepoint
```

7.4 Dipole Analysis

The dipole moment of the molecule can be tracked as a function of time throughout its excited state dynamics. Given the occupied state at a given time-step, the dipole of that state can be extracted from the standard output file. Note that in order for permanent (or total) dipole data to be available in the standard output file, the user must set `printdipole = 1` or more. Dipole analysis can be done on a single trajectory (`permdipole_raw_single.out`) or an ensemble of trajectories. For the latter, the mean over all trajectories (`permdipole_mean_ensemble.out`) or the raw data over all trajectories and time-steps (`permdipole_raw_ensemble`) may be requested. In `permdipole_raw_single.out` and `permdipole_mean_ensemble.out`, columns from left to right are: time (fs) and dipole moment (Debye). For the latter, there is third column with standard deviation. Columns in `permdipole_raw_ensemble.out` are the same as those in `permdipole_raw_single.out` except that the first column is trajectory directory. *Note that this function only works for excited-state dynamics, and that the dipole moment of the ground state is not yet available.* An example of calculating the mean dipole as a function of time is shown below.

```
[sifain@wc-fe1 realistic]$ ls
getexcited.py   gsdynamics   nexmd   singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 10

Select a task from the following list:

[1] Collect excited-state permanent dipole moment

Enter the number corresponding to the desired task: 1
Calculating the permanent dipole moment as a function of time.
Calculate dipole moment along one trajectory or an ensemble of trajectories?
Answer ONE [1] or ENSEMBLE [0]: 0
Ensemble directory [e.g. nexmd]: nexmd/0009
```

```
Output mean dipole in time or output dipoles at all time-steps and trajectories?
Answer MEAN [0] or ALL [1]: 0
Calculate dipole up to what time in femtoseconds?
Note that averaged results will only include trajectories that are complete up to this time: 1
Checking permanent dipole moments and states. Please wait ...
nexmd/NEXMD1/0041 dipole lines in md.out found
nexmd/NEXMD1/0041 dipoles in md.out extracted
nexmd/NEXMD1/0100 dipole lines in md.out found
nexmd/NEXMD1/0100 dipoles in md.out extracted
⋮
nexmd/NEXMD5/29895 dipole lines in md.out found
nexmd/NEXMD5/29895 dipoles in md.out extracted
nexmd/NEXMD5/29954 dipole lines in md.out found
nexmd/NEXMD5/29954 dipoles in md.out extracted
nexmd/NEXMD1/0041 1000.00
nexmd/NEXMD1/0100 1000.00
⋮
nexmd/NEXMD5/29895 1000.00
nexmd/NEXMD5/29954 1000.00
Total Trajectories: 0508
Completed Trajectories: 0508
Excellent Trajectories: 0508
[sifain@wc-fe1 nexmd_solvent]$ ls
getexcited.py  gsdynamics   nexmd   permdipole.out  singlepoint
```

7.5 Transition Density Analysis

The evolution of the electronic wavefunction can be tracked by changes in the spatial localization of the transition density (TD). Diagonal elements of the TD matrices $(\rho^{g\alpha})_{nn}(t) \equiv \left\langle \phi_\alpha(t) | c_n^\dagger c_n | \phi_g(t) \right\rangle$ represent changes in the electronic density in an atomic orbital (AO) when undergoing a ground- to excited-state transition, where $c_n^\dagger$ and c_n are creation and annihilation operators, respectively, and n refers to the AO basis function. [39] The fraction of TD localized on a molecular fragment is the sum of all atomic contributions. [40] TD analysis describes evolution of the electronic wavefunction without relying on adiabatic state populations, thereby providing a simple way of determining the spatial localization of excitations in time. [4, 15, 16]

If TD analysis is requested, an input file must be provided which splits the molecule into fragments. This is accomplished by defining atom numbers according to how they are ordered in a general input file between `&coord` and `&endcoord`. The atom numbers are to be written in a user-generated file, where the keyword `break` is to be inserted wherever a new fragment of the molecule is defined. An example of which is shown below.

```
1
2
3
4
break
5
6
7
8
```

In this example, the molecule contains 8 atoms and is split in half where the first four atoms between `&coord` and `&endcoord` define one fragment and the remaining atoms define the other fragment. TD analysis can be done on a single trajectory (`td_single.out`) or an ensemble of trajectories. For the latter, the mean over all trajectories (`td_mean_ensemble.out`) or the raw data over all trajectories and time-steps (`td_raw_ensemble`) may be requested. In `td_single.out` and `td_mean_ensemble.out`, columns from left to right are: time (fs), followed by the occupations of all fragements in the same order as they were defined in the user-generated fragment file, followed by the sum of all contributions (should be equal to 1.0). Columns in `td_raw_ensemble.out` are the same as those in `td_mean_ensemble.out` except that the first column is trajectory directory. An example of performing TD analysis is shown below.

```
[sifain@wc-fe1 realistic]$ ls
frag_HH_asymm getexcited.py gsdynamics nexmd singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
```

```
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 11

Select a task from the following list:

[1] Analyze occupancy according to diagonal elements of the transition density matrix

Enter the number corresponding to the desired task: 1
Calculating occupation of excitation as a function of time according to transition densities.
Calculate occupation along one trajectory or an ensemble of trajectories?
Answer ONE [1] or ENSEMBLE [0]: 0
Ensemble directory [e.g. nexmd]: nexmd
Output mean occupation in time, at all time-steps and trajectories,
or up to some user-defined time? Answer MEAN [0], ALL [1], or USER-DEFINED [2]: 0
Directory with an input.ceon file for coordinates, do not include input.ceon in the path:
nexmd/NEXMD1/0041
Directory with fragment file, include name of file in the path: frag_HH_asymm
Calculate occupation up to what time in femtoseconds?
Note that averaged results will only include trajectories that are
complete up to this time: 100
nexmd/NEXMD1/0041 1000.00
nexmd/NEXMD1/0100 1000.00
nexmd/NEXMD1/0159 1000.00
⋮
nexmd/NEXMD5/29836 1000.00
nexmd/NEXMD5/29895 1000.00
nexmd/NEXMD5/29954 1000.00
Total Trajectories: 0508
Completed Trajectories: 0508
Excellent Trajectories: 0508
[sifain@wc-fe1 realistic]$ ls
frag_HH_asymm getexcited.py gsdynamics nexmd singlepoint td_mean_ensemble.out
```

8 Code Testing Tools

8.1 Timings

CPU times can be obtained from trajectories of an ensemble. The timing data is divided up among `ground state`, `excited states`, `adiabatic forces`, and `nonadibatic derivatives`. The total CPU time is approximately the sum of these contributions. These timing data are available at the end of each standard output file (i.e. `md.out`). If collecting timing data is requested, an output file called `timing.out` is generated in the current working directory. Columns from left to right are: directory of trajectory, total CPU time, and CPU times of all contributions in the aforementioned order. An example of obtaining timing data is shown below.

```
[sifain@wc-fe1 realistic]$ ls
getexcited.py  gsdynamics   nexmd   singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 13
Select a task from the following list:

[1] Collect timing data from trajectories.

Enter the number corresponding to the desired task: 1
Collecting timings from trajectories.
NEXMD directory: nexmd
Collecting timings. Please wait ...
nexmd/NEXMD1/0041
nexmd/NEXMD1/0100
nexmd/NEXMD1/0159
⋮
nexmd/NEXMD5/29836
```

```
nexmd/NEXMD5/29895
nexmd/NEXMD5/29954
Mean Total CPU [s]: 051718
Mean Ground State [s]: 001269
Mean Excited States [s]: 006342
Mean Adiabatic Forces [s]: 000756
Mean Non-Adiabatic Derivatives [s]: 023824
sifain@wc-fe1 realistic]$ ls
getexcited.py  gsdynamics   nexmd   singlepoint  timing.out
```

9 Submitting Jobs

Jobs submissions can be categorized as either single trajectory or an ensemble of trajectories. The former, for example, is used for the long ground-state trajectory described in Subsection 6.2. The submission scripts discussed herein are specific to Slurm-based HPC systems. They can be translated and used with other job schedulers, but these changes must be made by the user. To submit a single trajectory, the `nexmd_single.csh` script may be used in the same directory where `input.ceon` is located. You must open `nexmd_single.csh` and change the specifications of the job, such as `walltime`, `jobname`, etc. Once this step is complete, simply type `sbatch nexmd_single.csh` in the terminal. A job ID will appear in the terminal.

To obtain meaningful non-adiabatic results, an ensemble of trajectories must be calculated. Two scripts are used to submit an ensemble of trajectories, `launch_batch.csh` and `nexmd_batch.csh`. The former reads the list of trajectories in `dirlist` located in each `NEXMD` directory and calls `nexmd_batch.csh`, which then submits a trajectory to each processor (typically 16 processors per node). The `nexmd_batch.csh` script can be stored in a home directory, while the `launch_batch.csh` script must be executed in each `NEXMD` folder. You must open `launch_batch.csh` and change the path to where `nexmd_batch.csh` is located. Similarly, you must also open `nexmd_batch.csh` and change the job specifications accordingly. An example of submitting an ensemble of trajectories is shown below.

```
[sifain@wc-fe1 NEXMD1]$ ls
0001 0067 0191 0244 0324 0408 0491 0574 0702 0837 0919 1000
0015 0072 0196 0259 0326 0411 0509 0580 0742 0841 0931 dirlist
0031 0074 0209 0272 0377 0430 0523 0586 0750 0843 0934 dirlist1
0034 0078 0220 0291 0380 0435 0527 0595 0769 0861 0936
0045 0083 0227 0292 0381 0439 0534 0596 0786 0863 0945
0051 0125 0229 0293 0395 0440 0552 0647 0805 0884 0957
0052 0127 0234 0306 0400 0441 0556 0664 0808 0896 0965
0053 0162 0235 0311 0402 0467 0561 0689 0818 0916 0981
0057 0186 0236 0322 0405 0485 0562 0697 0823 0917 0989
[sifain@wc-fe1 NEXMD1]$ /lustre/scratch1/turquoise/sifain/bin/launch_batch.csh
Submitted batch job 213991
Submitted batch job 213992
Submitted batch job 213993
Submitted batch job 213994
Submitted batch job 213995
Submitted batch job 213996
Submitted batch job 213997
```

In the example shown above, there are 100 trajectories in `NEXMD1`. The trajectories are sent to a total of 7 nodes. The first 6 nodes contain $6 \times 16 = 96$ trajectories, while the 7th node contains 4 trajectories.

10 Restarting Simulations

Non-adiabatic trajectories may not finish within the user-defined number of classical steps for several reasons such as (1) the computing system may have a time-limit that is less than the time to complete a trajectory, (2) the defined wall-time may not be long enough to complete the trajectory, or (3) a problem in the computing system may cause jobs to stop before completion. In any case, trajectories may be restarted from the last time-step, as long as the last excited-state, nuclear coordinates and velocities, and quantum coefficients are known. These quantities are available in `restart.out`. To prepare restart input files, type `python getexcited.py`. The contents of `restart.out` are inserted into a new input file, which will be called `input.ceon`. Within each `NEXMD` folder is a file called `dirlist`, which lists the trajectories that are to be restarted. During every iteration of generating restart input files a new random seeds list, called `rseedslist#`, will be generated in the current working directory, where `#` refers to the iteration. Note: It is important that the input file and list of directories for the current job submission be given the names `input.ceon` and `dirlist`, respectively, as these are how they are defined in the job submission script. If you wish to append results of the restarted trajectory to the previous standard output (e.g. `md.out`), it is important to change the output command in the submission script (e.g. `nexmd_batch.csh`) from `> md.out` to `>> md.out`. *Before restarting trajectories, it is important to check the end of their standard output files, as there may be error messages giving reasons as to why they failed to complete. In these cases, restarting trajectories may be irrelavant, as the same error message may reappear.* An example of preparing restart input files is shown below.

```
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd pop.out singlepoint
[sifain@wc-fe1 realistic]$ cd nexmd/
[sifain@wc-fe1 nexmd]$ ls
header NEXMD1 rseedslist
[sifain@wc-fe1 realistic]$ cd ..
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 7
Preparing restart input files for NEXMD.
NEXMD directory: nexmd
```

```
Currently, trajectories are set up to run for 50
classical steps with a time-step of 0.10 fs.
This is a total of 5.00 fs.
Keep this trajectory length? Answer YES [1] or NO [0]: 0
Enter new number of classical time-steps: 100
New random seeds? Answer YES [1] or NO [0]: 1
nexmd/NEXMD1/0002
nexmd/NEXMD1/0009
nexmd/NEXMD1/0016
⋮
nexmd/NEXMD1/0987
nexmd/NEXMD1/0989
nexmd/NEXMD1/0996
Deleting extraneous data in output files. Please wait ...
[sifain@wc-fe1 realistic]$ cd nexmd/
[sifain@wc-fe1 nexmd]$ ls
header NEXMD1 rseedslist rseedslist1
```

An example of preparing restart input files for NEXMD with a pre-generated list of random seeds is shown below.

```
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd rseedslist singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools

Enter the number corresponding to the desired task: 7
Preparing restart input files for NEXMD.
NEXMD directory: nexmd
Currently, trajectories are set to run for 100
classical steps with a time-step of 0.10 fs.
This is a total of 10.00 fs.
Keep this trajectory length? Answer YES [1] or NO [0]: 1
```

```
New random seeds? Answer YES [1] or NO [0]: 0
Path to random-seeds list (** must be different from past random seeds **): rseedslist
nexmd/NEXMD1/0002
nexmd/NEXMD1/0015
nexmd/NEXMD1/0020
⋮
nexmd/NEXMD1/0984
nexmd/NEXMD1/0988
nexmd/NEXMD1/1000
Deleting extraneous data in output files. Please wait ...
[sifain@wc-fe1 realistic]$ cd nexmd/
[sifain@wc-fe1 nexmd]$ ls
header NEXMD1 rseedslist rseedslist1
```

11 Complete Restart of Unfinished Trajectories

At times, you may experience strange behavior on your computing system. Depending on the scenario, it may be best to restart trajectories from the initial time. This is different from starting a trajectory from the last-generated time-step, as that explained in Section 10. To remove all the files for each trajectory that did not finish, use `getexcited.py`. The `input.ceon` file is the only file that is not removed. If the trajectory is complete, no files are removed. A new `dirlist` is generated with a list of trajectories. The job submission script reads `dirlist` and runs the trajectories listed therein. *You must be certain that a complete restart of unfinished trajectories is desired before accessing this option!* An example of deleting unfinished trajectories is shown below.

```
[sifain@wc-fe1 realistic]$ ls
ceo_gauss.out getexcited.py gsdynamics nexmd singlepoint
[sifain@wc-fe1 realistic]$ python getexcited.py

Select a task from the following list:

[1] Prepare input files for single-point calculations
[2] Generate an optical spectrum from single-point calculations
[3] Prepare input files for NEXMD
[4] Prepare input files for adiabatic dynamics with geometries from NEXMD
[5] Collect populations from NEXMD
[6] Collect PESs and NACTs from NEXMD
[7] Prepare restart input files for NEXMD
[8] Clean out the directories of NEXMD trajectories that are incomplete
[9] Access options for geometry analysis
[10] Access options for dipole analysis
[11] Access options for transition density analysis
[12] Access options for pump-push-probe spectroscopy (*** UNDER DEVELOPMENT, DO NOT USE ***)
[13] Access code testing tools
```

```
Enter the number corresponding to the desired task: 8
Cleaning directories of unfinished trajectories.
Are you sure you want to delete all unfinished trajectories?
Answer YES [1] or NO [0]: 1
NEXMD directory: nexmd
Trajectories less than 200 classical time-steps will be deleted.
Continue? Answer YES [1] or NO [0]: 1
nexmd/NEXMD1/0001
nexmd/NEXMD1/0015
nexmd/NEXMD1/0031
⋮
nexmd/NEXMD1/0981
nexmd/NEXMD1/0989
nexmd/NEXMD1/1000
The contents of 2 trajectories have been deleted.
```

12 Output Files

- `coords.xyz`, `velocity.out`, and `coefficient.out` contain the coordinates in angstroms (Å), velocities in angstroms per femtosecond (Å/fs), and coefficients of the system as a function of time, respectively. These data are written to their respective files at a rate which was specified in `out_data_steps` and `out_coords_steps` under `Output & Log Parameters`. The `coords.xyz` file can be read by several molecular visualization tools.

- `restart.out` contains the coordinates, velocities, coefficients, and residing PES at the last-generated time-step. This file is used to generate restart input files.

- `pes.out` contains PESs of all states being propagated as a function of time. The first two columns are time in femtoseconds and ground-state energy in eV, respectively. The remaining columns are excited-state energies in eV from $|1\rangle$ to $|N\rangle$, respectively, where N is the number of states being propagated.

- `nact.out` contains the non-adiabatic coupling terms between all pairs of states. The non-adiabatic coupling between states $|\alpha\rangle$ and $|\beta\rangle$ is defined as $\dot{\mathbf{R}} \cdot \mathbf{d}_{\alpha\beta}$, where $\dot{\mathbf{R}}$ is velocity and $\mathbf{d}_{\alpha\beta}$ is the non-adiabatic coupling vector between states $|\alpha\rangle$ and $|\beta\rangle$. The first column is time in femtoseconds. The remaining columns are consecutive rows of the non-adiabatic coupling matrix. For example, if 2 states were being propagated, the output would be $\dot{\mathbf{R}} \cdot \mathbf{d}_{\alpha\beta}$ for $\alpha\beta = 11$, 12, 21, and 22, respectively. The diagonal terms of the non-adiabatic coupling are zero and the non-adiabatic coupling vector matrix is anti-Hermitian such that $\mathbf{d} = -\mathbf{d}^\dagger$ or $\mathbf{d}_{\alpha\beta} = -\mathbf{d}^*_{\beta\alpha}$.

- `temperature.out` contains the temperature of the system as a function of time. The temperature is calculated from the total nuclear kinetic energy. The first column is time in femtoseconds, the second column is the temperature of the system in Kelvin, and the third column is the set temperature of the thermostat in Kelvin.

- `hops-trial.out` contains attempted and successful hops throughout the trajectory. Hops may be rejected due to energy conservation. This occurs when dispensable nuclear kinetic energy does not exceed the energy barrier between the residing and target surfaces. Therefore, rejected hops can only occur when the target surface lies above the residing surface. The first column is the time at which attempted hops occur in femtoseconds, the second column is the target surface, and the third column labels type of hops, where `0` is a successful hop, `1` is a rejected hop with no decoherence event, and `2` is a rejected hop with a decoherence event. The latter two depend on whether the `decoher_type` was set to `1` or `2` under `Non-Adiabatic Parameters` of the `input.ceon` file. The wavefunction also decoheres at hops of type `0` if `decoher_type=1` or `2`.

- `coeff-n.out` contains the residing surface as a function of time, as well as populations of all excited-states being propagated. The first two columns are residing surface and time in femtoseconds, respectively. The remaining columns are excited-state populations from $|1\rangle$ to $|N\rangle$, respectively, where N is the number of states being propagated. The last column is the sum of all populations, which should be approximately 1.0.

- `force.out` contains the forces along the x, y, and z directions for each atom in the system as a function of time. In order for force data to be outputted to `force.out`, `verbosity` under `Output and Log Parameters` must be set to `3`. The unit of force is eV/Å.

- `muab.out` contains the excited-to-excited dipoles in arbitrary units. This file is generated when `calcxdens=.true.` during single-point calculations. First and second columns label states $|i\rangle$ and $|j\rangle$, respectively. The third column is the energy difference, $E_{ji} = E_j - E_i$ in eV. The following three columns are excited-to-excited dipoles along the x, y, and z axes, respectively. The last column is the total dipole moment.

- `nacr.out` contains the nonadiabatic coupling vector. The x, y and z components of the nonadiabatic coupling vector are shown for each atom, at all attempted hops. This includes successful and unsuccessful hops. The latter, which are also known as rejected hops, are due to lack of energy conservation. The columns from left to right show the time at which a successful hops occurs, the state before the hop, the state after the hop, the atom number, followed by the nonadiabatic coupling in the x, y, and z directions, respectively.

- `transition-densities.out` contains diagonal elements of the transition density as a function of time for each atom in the current excited state. This file is important for determining where the excitation is localized/delocalized throughout the molecule. The number of atomic orbitals for each atom is 1 for hydrogen and at most 4 for heavier atoms. The sequence of the atoms corresponds to the same sequence used in `input.ceon` between `&coord` and `&endcoord`. As an example, the columns from left to right, in `transition-density.out` for H_2O (where atoms are listed in this order) would be: time in femtoseconds, 1 column showing occupation of H, 1 column showing occupation of H, and 4 columns showing occupation of O. It is important to sum the contributions of all atomic orbitals for a single atom.

References

[1] T. Nelson, S. Fernandez-Alberti, V. Chernyak, A. E. Roitberg, and S. Tretiak, J. Phys. Chem. B **115**, 5402 (2011).

[2] T. Nelson, S. Fernandez-Alberti, A. E. Roitberg, and S. Tretiak, Acc. Chem. Res. **47**, 1155 (2014).

[3] T. Nelson, S. Fernandez-Alberti, V. Chernyak, A. E. Roitberg, and S. Tretiak, J. Chem. Phys. **136**, 054108 (2012).

[4] S. Fernandez-Alberti, A. E. Roitberg, T. Nelson, and S. Tretiak, J. Chem. Phys. **137**, 014512 (2012).

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

[6] T. Nelson, S. Fernandez-Alberti, A. E. Roitberg, and S. Tretiak, Chem. Phys. Lett. **590**, 208 (2013).

[7] M. A. Soler, T. Nelson, A. E. Roitberg, S. Tretiak, and S. Fernandez-Alberti, J. Phys. Chem. A **118**, 10372 (2014).

[8] J. A. Bjorgaard, V. Kuzmenko, K. Velizhanin, and S. Tretiak, J. Chem. Phys. **142**, 044103 (2015).

[9] J. A. Bjorgaard, K. A. Velizhanin, and S. Tretiak, J. Chem. Phys. **143**, 054305 (2015).

[10] T. Nelson, A. Naumov, S. Fernandez-Alberti, and S. Tretiak, Chem. Phys. **481**, 84 (2016).

[11] J. A. Bjorgaard, K. A. Velizhanin, and S. Tretiak, J. Chem. Phys. **144**, 154104 (2016).

[12] S. Fernandez-Alberti, V. D. Kleiman, S. Tretiak, and A. E. Roitberg, J. Phys. Chem. A **113**, 7535 (2009).

[13] S. Fernandez-Alberti, V. D. Kleiman, S. Tretiak, and A. E. Roitberg, J. Phys. Chem. Lett. **1**, 2699 (2010).

[14] M. A. Soler, A. E. Roitberg, T. Nelson, S. Tretiak, and S. Fernandez-Alberti, J. Phys. Chem. A **116**, 9802 (2012).

[15] S. Fernandez-Alberti, A. E. Roitberg, V. D. Kleiman, T. Nelson, and S. Tretiak, J. Chem. Phys. **137**, 22A526 (2012).

[16] T. Nelson, S. Fernandez-Alberti, A. E. Roitberg, and S. Tretiak, Phys. Chem. Chem. Phys. **15**, 9245 (2013).

[17] N. Oldani, S. Tretiak, G. Bazan, and S. Fernandez-Alberti, Energy Environ. Sci. **7**, 1175 (2014).

[18] D. Ondarse-Alvarez, N. Oldani, S. Tretiak, and S. Fernandez-Alberti, J. Phys. Chem. A **118**, 10742 (2014).

[19] J. F. Galindo et al., J. Am. Chem. Soc. **137**, 11637 (2015).

[20] W. P. Bricker et al., Sci. Rep. **5** (2015).

[21] P. M. Shenai, S. Fernandez-Alberti, W. P. Bricker, S. Tretiak, and Y. Zhao, J. Phys. Chem. B **120**, 49 (2015).

[22] M. T. Greenfield et al., J. Phys. Chem. A **119**, 4846 (2015).

[23] L. Alfonso Hernandez, T. Nelson, S. Tretiak, and S. Fernandez-Alberti, J. Phys. Chem. B **119**, 7242 (2015).

[24] J. A. Bjorgaard et al., Chem. Phys. Lett. **631**, 66 (2015).

[25] R. Franklin-Mergarejo, D. O. Alvarez, S. Tretiak, and S. Fernandez-Alberti, Sci. Rep. **6** (2016).

[26] D. Ondarse-Alvarez et al., Phys. Chem. Chem. Phys. **18**, 25080 (2016).

[27] T. Nelson et al., J. Phys. Chem. A **120**, 519 (2016).

[28] L. Alfonso Hernandez et al., J. Phys. Chem. Lett. **7**, 4936 (2016).

[29] T. R. Nelson, S. Fernandez-Alberti, A. E. Roitberg, and S. Tretiak, J. Phys. Chem. Lett. (2017).

[30] R. Franklin-Mergarejo, T. Nelson, S. Tretiak, and S. Fernandez-Alberti, Phys. Chem. Chem. Phys. **19**, 9478 (2017).

[31] F. Zheng, S. Fernandez-Alberti, S. Tretiak, and Y. Zhao, J. Phys. Chem. B (2017).

[32] Amber Reference Manual: http://ambermd.org/doc12/Amber17.pdf (2017).

[33] W. M. Haynes, *CRC Handbook of Chemistry and Physics*, CRC press, 2014.

[34] J. C. Tully, J. Chem. Phys. **93**, 1061 (1990).

[35] G. Granucci, M. Persico, and A. Zoccante, J. Chem. Phys. **133**, 134111 (2010).

[36] G. Granucci and M. Persico, J. Chem. Phys. **126**, 134114 (2007).

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

[38] M. D. Hack and D. G. Truhlar, J. Chem. Phys. **114**, 9305 (2001).

[39] C. Wu, S. V. Malinin, S. Tretiak, and V. Y. Chernyak, Nat. Phys. **2**, 631 (2006).

[40] M. Baer, *Beyond Born-Oppenheimer: Electronic Nonadiabatic Coupling Terms and Conical Intersections*, John Wiley & Sons, 2006.

A Appendix

A.1 Example Input File: Geometry Optimization

Shown below is an example input file for geometry optimization. Important inputs include:

- `grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]`
- `maxcyc=300, ! Number of cycles for geometry optimization [0]`
- `n_class_steps=0, ! Number of classical steps [1]`
- `exc_state_init=0, ! Initial excited state (0 - ground state) [0]`

You must set the convergence criteria with `grms_tol` and the number of cycles for optimization with `maxcyc` > 0. The latter is normally set to zero for all other calculations within `NEXMD`. Whenever `maxcyc` > 0, set `n_class_steps` $= 0$. Attempting to run geometry optimization and dynamics simultaneously will generate an error. You may optimize in a particular state using `exc_state_init`, where `0` labels the ground state. Other important inputs include the initial geometry contained within `&coord` and `&endcoord`.

```
&qmmm
   !***** Geometry Optimization
   maxcyc=300, ! Number of cycles for geometry optimization [0]
   ntpr=1, ! Print results every ntpr cycles [1]
   grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]

   !***** Ground-State and Output Parameters
   qm_theory='AM1', ! Integral type, check Amber's SQM for more options [AM1]
   scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]
   verbosity=1, ! QM/MM output verbosity (0-minimum, 5-maximum)
   ! [1 for dynamics and optimization, 5 for others]
   printdipole=2, ! (0) Unrelaxed transitions, (1) Unrelaxed transitions plus
   ! total molecular, or (2) Unrelaxed/relaxed transitions plus
   ! total molecular [1 for dynamics, 2 for optimization and single-point]
   printbondorders=0, ! (0) No or (1) Yes [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   density_predict=0, ! (0) None, (1) Reversible MD,
   ! or (2) XL-BOMD [0] *** ALL ARE UNDER DEVELOPMENT, DO NOT USE ***
   itrmax=300, ! Max SCF iterations for ground state
   ! (negative to ignore convergence) [300]

   !***** Excited-State Parameters
   exst_method=1, ! CIS (1) or RPA (2) [1]
   dav_guess=1, ! Restart Davidson from (0) Scratch, (1) Previous,
   ! or (2) XL-BOMD [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]
   ftol1=1.0d-8, ! Acceptance tolerance for residual norm [1.0d-5]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   dav_maxcyc=200, ! Max cycles for Davidson diagonalization
   ! (negative to ignore convergence) [100]
   printcharges=0, ! Print (1) or do not print (0) Mulliken charges of QM atoms [0]
   calcxdens=.false., ! Print (.true.) or do not print (.false.)
   ! excited-to-excited transition dipole moments [.false.]

   !***** Solvent Models and External Electric Fields
   solvent_model=0, ! (0) None, (1) Linear response, (2) Vertical excitation,
   ! or (3) State-specific [0]
   potential_type=1, ! (1) COSMO or (2) Onsager [1]
   onsager_radius=2, ! Onsager radius, A (system dependent) [2]
   ceps=10, ! Dielectric constant, unitless [10]
   linmixparam=1, ! Linear mixing parameter for vertical excitation
   ! or state-specific SCF calculation [1]
   cosmo_scf_ftol=1.0d-5, ! Vertical excitation or state-specific
   ! SCF tolerance, eV [1.0d-5]
   doZ=.false., ! Use relaxed (.true.) or unrelaxed (.false) density for
   ! vertical excitation or state-specific COSMO or Onsager [.false.]
   index_of_refraction=100, ! Dielectric constant for linear response
   ! solvent in excited-state, unitless [100] *** UNDER DEVELOPMENT, DO NOT USE ***
   EF=0, ! (0) None or (1) Electric field in ground- and excited-state [0]
   Ex=0, ! Electric field vector X, eV/A [0]
   Ey=0, ! Electric field vector Y, eV/A [0]
   Ez=0, ! Electric field vector Z, eV/A [0]
&endqmmm

&moldyn
   !***** General Parameters
   natoms=12, ! Number of atoms
   ! (must be equal to the number of atoms in system)
   rnd_seed=19345, ! Seed for the random number generator
   bo_dynamics_flag=0, ! (0) Non-BO or (1) BO [1]
   exc_state_init=0, ! Initial excited state (0 - ground state) [0]
   n_exc_states_propagate=0, ! Number of excited states [0]

   !***** Dynamics Parameters
   time_init=0.0, ! Initial time, fs [0.0]
   time_step=0.1, ! Time step, fs [0.1]
   n_class_steps=0, ! Number of classical steps [1]
   n_quant_steps=4, ! Number of quantum steps for each classical step [4]
   moldyn_deriv_flag=1, ! (0) None, (1) Analytical, or (2) Numerical [1]
   num_deriv_step=1.0d-3, ! Displacement for numerical derivatives, A [1.0d-3]
   rk_tolerance=1.0d-7, ! Tolerance for the Runge-Kutta propagator [1.0d-7]

   !***** Non-Adiabatic Parameters
   decoher_type=2, ! Type of decoherence: Reinitialize (0) Never,
   ! (1) At successful hops, (2) At successful plus frustrated hops...
   ! (3) Persico/Granucci, or (4) Truhlar [2]
   ! *** (3) AND (4) ARE UNDER DEVELOPMENT, DO NOT USE ***
   decoher_e0=0.0, ! Decoherence parameter E0, Hartrees [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   decoher_c=0.0, ! Decoherence parameter C, unitless [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   dotrivial=1, ! Do unavoided (trivial) crossing routine (1) or not (0) [1]
   quant_step_reduction_factor=2.5d-2, ! Quantum step reduction factor [2.5d-2]

   !***** Thermostat Parameters
   therm_type=1, ! Thermostat type: (0) Newtonian, (1) Langevin,
   ! or (2) Berendsen [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   therm_temperature=300, ! Thermostat temperature, K [300]
   therm_friction=20, ! Thermostat friction coefficient, 1/ps [20]
   berendsen_relax_const=0.4, ! Bath relaxation constant for Berendsen
   ! thermostat, ps [0.4] *** UNDER DEVELOPMENT, DO NOT USE ***
   heating=0, ! Equilibrated (0) or heating (1) [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   heating_steps_per_degree=100, ! Number of steps per degree
   ! during heating [100] *** UNDER DEVELOPMENT, DO NOT USE ***

   !***** Output & Log Parameters
   verbosity=3, ! NEXMD output verbosity (0-minimum, 3-maximum)
   ! [2 for dynamics, 3 for optimization and single-point]
   out_data_steps=1, ! Number of steps to write data [1]
   out_coords_steps=10, ! Number of steps to write the restart file [10]
   out_data_cube=0, ! Write (1) or do not write (0) view files to generate cubes [0]
   out_count_init=0, ! Initial count for view files [0]
&endmoldyn

&coord
  6      -7.9798271101     0.6776918081    -0.0532285388
  6      -7.0849928010     1.7602597759     0.0294961792
  6      -5.7058415294     1.5490364812     0.0312760931
  6      -5.2231419594     0.2195333448    -0.0446043010
  6      -6.1050960756    -0.8685564920     0.0220869421
  6      -7.5344099241    -0.6444487634     0.0248126135
  1      -9.0268081830     0.8587716724    -0.0794794940
  1      -7.4774606514     2.7566353436     0.1635393862
  1      -5.0939335779     2.4479885163     0.1481876938
  1      -4.1292016456     0.0999373674    -0.1580811639
  1      -5.6916991654    -1.8878557992     0.1151090966
  1      -8.2838388636    -1.4313798704     0.1051927200
&endcoord

&veloc
    3.3718248255   -5.6032885851   -1.1970845430
```

```
     2.5106648755   2.0978837936  -1.0696411897
    -5.9135180273  -3.7505826950   1.1689299883
     7.7194332369   4.8702351843   0.6576546539
    -7.1851218597  -2.0113572464  -0.6329683366
    -1.7276579899   0.3919019235  -0.0257452789
   -17.0279163131   9.9875659542   5.3513734186
    -4.7222747943  18.9640275032  11.9601977632
    10.9539809532  17.0164104392  -9.7113209726
    25.7548696749   2.2116651958  -0.5444198125
   -16.5303708308  -2.3313274630  -3.2147489925
    16.2776787026   2.2582071549   9.3572624705
&endveloc

&coeff
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  1.00  0.00
  0.00  0.00
  0.00  0.00
&endcoeff
```

A.2 Example Input File: Ground-State Trajectory

Shown below is an example input file for a ground-state trajectory. Important inputs include:

- `scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]`
- `itrmax=300, ! Max SCF iterations for ground state`

  `! (negative to ignore convergence) [300]`
- `bo_dynamics_flag=1, ! (0) Non-BO or (1) BO [1]`
- `exc_state_init=0, ! Initial excited state (0 - ground state) [0]`
- `n_exc_states_propagate=0, ! Number of excited states [0]`
- `time_step=0.1, ! Time step, fs [0.1]`
- `n_class_steps=10000, ! Number of classical steps [1]`
- `out_data_steps=1, ! Number of steps to write data [1]`
- `out_coords_steps=10, ! Number of steps to write the restart file [10]`

Besides the ground-state convergence criteria, you must set `bo_dynamics_flag=1` for Born-Oppenheimer dynamics and `exc_state_init=0` to propagate on the ground state. To reduce computational cost, you should set `n_exc_states_propagate=0`. The product of `time_step` and `n_class_steps` defines the length of the trajectory. Also, `time_step` × `out_data_steps` × `out_coords_steps` defines how often coordinates and velocities are outputted to `coords.xyz`. It is important to be mindful of the latter in order to obtain a good ground-state sampling for excited-state dynamics. In this example, the trajectory is 1000.0 fs in length, and coordinates and velocities are outputted every 1.0 fs to `coords.xyz`.

```
&qmmm
   !***** Geometry Optimization
   maxcyc=0, ! Number of cycles for geometry optimization [0]
   ntpr=1, ! Print results every ntpr cycles [1]
   grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]

   !***** Ground-State and Output Parameters
   qm_theory='AM1', ! Integral type, check Amber's SQM for more options [AM1]
   scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]
   verbosity=1, ! QM/MM output verbosity (0-minimum, 5-maximum)
   ! [1 for dynamics and optimization, 5 for others]
   printdipole=1, ! (0) Unrelaxed transitions, (1) Unrelaxed transitions plus
   ! total molecular, or (2) Unrelaxed/relaxed transitions plus
   ! total molecular [1 for dynamics, 2 for optimization and single-point]
   printbondorders=0, ! (0) No or (1) Yes [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   density_predict=0, ! (0) None, (1) Reversible MD,
   ! or (2) XL-BOMD [0] *** ALL ARE UNDER DEVELOPMENT, DO NOT USE ***
   itrmax=300, ! Max SCF iterations for ground state
   ! (negative to ignore convergence) [300]

   !***** Excited-State Parameters
   exst_method=1, ! CIS (1) or RPA (2) [1]
   dav_guess=1, ! Restart Davidson from (0) Scratch, (1) Previous,
   ! or (2) XL-BOMD [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]
   ftol1=1.0d-8, ! Acceptance tolerance for residual norm [1.0d-5]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   dav_maxcyc=200, ! Max cycles for Davidson diagonalization
   ! (negative to ignore convergence) [100]
   printcharges=0, ! Print (1) or do not print (0) Mulliken charges of QM atoms [0]
   calcxdens=.false., ! Print (.true.) or do not print (.false.)
   ! excited-to-excited transition dipole moments [.false.]

   !***** Solvent Models and External Electric Fields
   solvent_model=0, ! (0) None, (1) Linear response, (2) Vertical excitation,
   ! or (3) State-specific [0]
   potential_type=1, ! (1) COSMO or (2) Onsager [1]
   onsager_radius=2, ! Onsager radius, A (system dependent) [2]
   ceps=10, ! Dielectric constant, unitless [10]
   linmixparam=1, ! Linear mixing parameter for vertical excitation
   ! or state-specific SCF calculation [1]
   cosmo_scf_ftol=1.0d-5, ! Vertical excitation or state-specific
   ! SCF tolerance, eV [1.0d-5]
   doZ=.false., ! Use relaxed (.true.) or unrelaxed (.false) density for
   ! vertical excitation or state-specific COSMO or Onsager [.false.]
   index_of_refraction=100, ! Dielectric constant for linear response
   ! solvent in excited-state, unitless [100] *** UNDER DEVELOPMENT, DO NOT USE ***
   EF=0, ! (0) None or (1) Electric field in ground- and excited-state [0]
   Ex=0, ! Electric field vector X, eV/A [0]
   Ey=0, ! Electric field vector Y, eV/A [0]
   Ez=0, ! Electric field vector Z, eV/A [0]
&endqmmm

&moldyn
   !***** General Parameters
   natoms=12, ! Number of atoms
   ! (must be equal to the number of atoms in system)
   rnd_seed=19345, ! Seed for the random number generator
   bo_dynamics_flag=1, ! (0) Non-BO or (1) BO [1]
   exc_state_init=0, ! Initial excited state (0 - ground state) [0]
   n_exc_states_propagate=0, ! Number of excited states [0]

   !***** Dynamics Parameters
   time_init=0.0, ! Initial time, fs [0.0]
   time_step=0.1, ! Time step, fs [0.1]
   n_class_steps=10000, ! Number of classical steps [1]
   n_quant_steps=4, ! Number of quantum steps for each classical step [4]
   moldyn_deriv_flag=1, ! (0) None, (1) Analytical, or (2) Numerical [1]
   num_deriv_step=1.0d-3, ! Displacement for numerical derivatives, A [1.0d-3]
   rk_tolerance=1.0d-7, ! Tolerance for the Runge-Kutta propagator [1.0d-7]

   !***** Non-Adiabatic Parameters
   decoher_type=2, ! Type of decoherence: Reinitialize (0) Never,
   ! (1) At successful hops, (2) At successful plus frustrated hops...
   ! (3) Persico/Granucci, or (4) Truhlar [2]
   ! *** (3) AND (4) ARE UNDER DEVELOPMENT, DO NOT USE ***
   decoher_e0=0.0, ! Decoherence parameter E0, Hartrees [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   decoher_c=0.0, ! Decoherence parameter C, unitless [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   dotrivial=1, ! Do unavoided (trivial) crossing routine (1) or not (0) [1]
   quant_step_reduction_factor=2.5d-2, ! Quantum step reduction factor [2.5d-2]

   !***** Thermostat Parameters
   therm_type=1, ! Thermostat type: (0) Newtonian, (1) Langevin,
   ! or (2) Berendsen [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   therm_temperature=300, ! Thermostat temperature, K [300]
   therm_friction=20, ! Thermostat friction coefficient, 1/ps [20]
   berendsen_relax_const=0.4, ! Bath relaxation constant for Berendsen
   ! thermostat, ps [0.4] *** UNDER DEVELOPMENT, DO NOT USE ***
   heating=0, ! Equilibrated (0) or heating (1) [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   heating_steps_per_degree=100, ! Number of steps per degree
   ! during heating [100] *** UNDER DEVELOPMENT, DO NOT USE ***

   !***** Output & Log Parameters
   verbosity=2, ! NEXMD output verbosity (0-minimum, 3-maximum)
   ! [2 for dynamics, 3 for optimization and single-point]
   out_data_steps=1, ! Number of steps to write data [1]
   out_coords_steps=10, ! Number of steps to write the restart file [10]
   out_data_cube=0, ! Write (1) or do not write (0) view files to generate cubes [0]
   out_count_init=0, ! Initial count for view files [0]
&endmoldyn

&coord
  6      -7.9798271101     0.6776918081    -0.0532285388
  6      -7.0849928010     1.7602597759     0.0294961792
  6      -5.7058415294     1.5490364812     0.0312760931
  6      -5.2231419594     0.2195333448    -0.0446043010
  6      -6.1050960756    -0.8685564920     0.0220869421
  6      -7.5344099241    -0.6444487634     0.0248126135
  1      -9.0268081830     0.8587716724    -0.0794794940
  1      -7.4774606514     2.7566353436     0.1635393862
  1      -5.0939335779     2.4479885163     0.1481876938
  1      -4.1292016456     0.0999373674    -0.1580811639
  1      -5.6916991654    -1.8878557992     0.1151090966
  1      -8.2838388636    -1.4313798704     0.1051927200
&endcoord

&veloc
     3.3718248255   -5.6032885851   -1.1970845430
     2.5106648755    2.0978837936   -1.0696411897
    -5.9135180273   -3.7505826950    1.1689299883
     7.7194332369    4.8702351843    0.6576546539
    -7.1851218597   -2.0113572464   -0.6329683366
    -1.7276579899    0.3919019235   -0.0257452789
   -17.0279163131    9.9875659542    5.3513734186
    -4.7222747943   18.9640275032   11.9601977632
    10.9539809532   17.0164104392   -9.7113209726
    25.7548696749    2.2116651958   -0.5444198125
   -16.5303708308   -2.3313274630   -3.2147489925
    16.2776787026    2.2582071549    9.3572624705
&endveloc

&coeff
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  1.00  0.00
  0.00  0.00
  0.00  0.00
&endcoeff
```

A.3 Example Input File: Single-Point Calculation

Shown below is an example input file for a single-point calculation. Important inputs include:

- `scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]`
- `verbosity=5, ! QM/MM output verbosity (0-minimum, 5-maximum)`
- `itrmax=300, ! Max SCF iterations for ground state`

  `! (negative to ignore convergence) [300]`
- `ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]`
- `dav_maxcyc=200, ! Max cycles for Davidson diagonalization`

  `! (negative to ignore convergence) [100]`
- `exc_state_init=0, ! Initial excited state (0 - ground state) [0]`
- `n_exc_states_propagate=30, ! Number of excited states [0]`
- `n_class_steps=0, ! Number of classical steps [1]`

You must set the ground-state and excited-state convergence criteria in `scfconv` and `ftol0`, respectively. The number of cycles for these calculations must also be set with `itrmax` and `dav_maxcyc`, respectively, but typically, these inputs will not be changed. The initial excited-state must be set with `exc_state_init`. For determining the optical absorption from the ground state, as that discussed in Subsections 6.4 and 6.5, `exc_state_init=0`. The number of excited states must be set with `n_exc_states_propagate`. For single-point calculations, `n_exc_states_propagate` can be relatively large, and which may be needed to compute a *full* absorption spectrum. During excited-state dynamics, however, this number should only be as large as a few states above the highest-excited trajectory. Lastly, `n_class_steps=0` for single-point calculations. The `verbosity` of single-point calculations should be set to at least 1 in order to obtain energies and oscillators strengths, which are needed to generate the absorption spectrum in Subsection 6.5.

```
&qmmm
   !***** Geometry Optimization
   maxcyc=0, ! Number of cycles for geometry optimization [0]
   ntpr=1, ! Print results every ntpr cycles [1]
   grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]

   !***** Ground-State and Output Parameters
   qm_theory='AM1', ! Integral type, check Amber's SQM for more options [AM1]
   scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]
   verbosity=5, ! QM/MM output verbosity (0-minimum, 5-maximum)
   ! [1 for dynamics and optimization, 5 for others]
   printdipole=2, ! (0) Unrelaxed transitions, (1) Unrelaxed transitions plus
   ! total molecular, or (2) Unrelaxed/relaxed transitions plus
   ! total molecular [1 for dynamics, 2 for optimization and single-point]
   printbondorders=0, ! (0) No or (1) Yes [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   density_predict=0, ! (0) None, (1) Reversible MD,
   ! or (2) XL-BOMD [0] *** ALL ARE UNDER DEVELOPMENT, DO NOT USE ***
   itrmax=300, ! Max SCF iterations for ground state
   ! (negative to ignore convergence) [300]

   !***** Excited-State Parameters
   exst_method=1, ! CIS (1) or RPA (2) [1]
   dav_guess=1, ! Restart Davidson from (0) Scratch, (1) Previous,
   ! or (2) XL-BOMD [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]
   ftol1=1.0d-8, ! Acceptance tolerance for residual norm [1.0d-5]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   dav_maxcyc=200, ! Max cycles for Davidson diagonalization
   ! (negative to ignore convergence) [100]
   printcharges=0, ! Print (1) or do not print (0) Mulliken charges of QM atoms [0]
   calcxdens=.false., ! Print (.true.) or do not print (.false.)
   ! excited-to-excited transition dipole moments [.false.]

   !***** Solvent Models and External Electric Fields
   solvent_model=0, ! (0) None, (1) Linear response, (2) Vertical excitation,
   ! or (3) State-specific [0]
   potential_type=1, ! (1) COSMO or (2) Onsager [1]
   onsager_radius=2, ! Onsager radius, A (system dependent) [2]
   ceps=10, ! Dielectric constant, unitless [10]
   linmixparam=1, ! Linear mixing parameter for vertical excitation
   ! or state-specific SCF calculation [1]
   cosmo_scf_ftol=1.0d-5, ! Vertical excitation or state-specific
   ! SCF tolerance, eV [1.0d-5]
   doZ=.false., ! Use relaxed (.true.) or unrelaxed (.false) density for
   ! vertical excitation or state-specific COSMO or Onsager [.false.]
   index_of_refraction=100, ! Dielectric constant for linear response
   ! solvent in excited-state, unitless [100] *** UNDER DEVELOPMENT, DO NOT USE ***
   EF=0, ! (0) None or (1) Electric field in ground- and excited-state [0]
   Ex=0, ! Electric field vector X, eV/A [0]
   Ey=0, ! Electric field vector Y, eV/A [0]
   Ez=0, ! Electric field vector Z, eV/A [0]
&endqmmm

&moldyn
   !***** General Parameters
   natoms=12, ! Number of atoms
   ! (must be equal to the number of atoms in system)
   rnd_seed=19345, ! Seed for the random number generator
   bo_dynamics_flag=1, ! (0) Non-BO or (1) BO [1]
   exc_state_init=0, ! Initial excited state (0 - ground state) [0]
   n_exc_states_propagate=30, ! Number of excited states [0]

   !***** Dynamics Parameters
   time_init=0.0, ! Initial time, fs [0.0]
   time_step=0.1, ! Time step, fs [0.1]
   n_class_steps=0, ! Number of classical steps [1]
   n_quant_steps=4, ! Number of quantum steps for each classical step [4]
   moldyn_deriv_flag=1, ! (0) None, (1) Analytical, or (2) Numerical [1]
   num_deriv_step=1.0d-3, ! Displacement for numerical derivatives, A [1.0d-3]
   rk_tolerance=1.0d-7, ! Tolerance for the Runge-Kutta propagator [1.0d-7]

   !***** Non-Adiabatic Parameters
   decoher_type=2, ! Type of decoherence: Reinitialize (0) Never,
   ! (1) At successful hops, (2) At successful plus frustrated hops...
   ! (3) Persico/Granucci, or (4) Truhlar [2]
   ! *** (3) AND (4) ARE UNDER DEVELOPMENT, DO NOT USE ***
   decoher_e0=0.0, ! Decoherence parameter E0, Hartrees [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   decoher_c=0.0, ! Decoherence parameter C, unitless [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   dotrivial=1, ! Do unavoided (trivial) crossing routine (1) or not (0) [1]
   quant_step_reduction_factor=2.5d-2, ! Quantum step reduction factor [2.5d-2]

   !***** Thermostat Parameters
   therm_type=1, ! Thermostat type: (0) Newtonian, (1) Langevin,
   ! or (2) Berendsen [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   therm_temperature=300, ! Thermostat temperature, K [300]
   therm_friction=20, ! Thermostat friction coefficient, 1/ps [20]
   berendsen_relax_const=0.4, ! Bath relaxation constant for Berendsen
   ! thermostat, ps [0.4] *** UNDER DEVELOPMENT, DO NOT USE ***
   heating=0, ! Equilibrated (0) or heating (1) [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   heating_steps_per_degree=100, ! Number of steps per degree
   ! during heating [100] *** UNDER DEVELOPMENT, DO NOT USE ***

   !***** Output & Log Parameters
   verbosity=3, ! NEXMD output verbosity (0-minimum, 3-maximum)
   ! [2 for dynamics, 3 for optimization and single-point]
   out_data_steps=1, ! Number of steps to write data [1]
   out_coords_steps=10, ! Number of steps to write the restart file [10]
   out_data_cube=0, ! Write (1) or do not write (0) view files to generate cubes [0]
   out_count_init=0, ! Initial count for view files [0]
&endmoldyn

&coord
  6      -7.9798271101     0.6776918081    -0.0532285388
  6      -7.0849928010     1.7602597759     0.0294961792
  6      -5.7058415294     1.5490364812     0.0312760931
  6      -5.2231419594     0.2195333448    -0.0446043010
  6      -6.1050960756    -0.8685564920     0.0220869421
  6      -7.5344099241    -0.6444487634     0.0248126135
  1      -9.0268081830     0.8587716724    -0.0794794940
  1      -7.4774606514     2.7566353436     0.1635393862
  1      -5.0939335779     2.4479885163     0.1481876938
  1      -4.1292016456     0.0999373674    -0.1580811639
  1      -5.6916991654    -1.8878557992     0.1151090966
  1      -8.2838388636    -1.4313798704     0.1051927200
&endcoord

&veloc
     3.3718248255  -5.6032885851  -1.1970845430
     2.5106648755   2.0978837936  -1.0696411897
    -5.9135180273  -3.7505826950   1.1689299883
     7.7194332369   4.8702351843   0.6576546539
    -7.1851218597  -2.0113572464  -0.6329683366
    -1.7276579899   0.3919019235  -0.0257452789
   -17.0279163131   9.9875659542   5.3513734186
    -4.7222747943  18.9640275032  11.9601977632
    10.9539809532  17.0164104392  -9.7113209726
    25.7548696749   2.2116651958  -0.5444198125
   -16.5303708308  -2.3313274630  -3.2147489925
    16.2776787026   2.2582071549   9.3572624705
&endveloc

&coeff
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  1.00  0.00
  0.00  0.00
  0.00  0.00
&endcoeff
```

A.4 Example Input File: Adiabatic Excited-State Trajectory

Shown below is an example input file for an adiabatic excited-state trajectory. Important inputs include:

- `scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]`
- `ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]`
- `rnd_seed=19345, ! Seed for the random number generator`
- `bo_dynamics_flag=1, ! (0) Non-BO or (1) BO [1]`
- `exc_state_init=4, ! Initial excited state (0 - ground state) [0]`
- `n_exc_states_propagate=4, ! Number of excited states [0]`
- `time_step=0.1, ! Time step, fs [0.1]`
- `n_class_steps=10000, ! Number of classical steps [1]`

The convergence criteria of the ground and excited state are set with `scfconv` and `ftol0`, respectively. Typically, `ftol0` is an order or magnitude greater than `scfconv` to reduce computational cost. The random seed in `rnd_seed` is used for the nuclear Langevin dynamics. Besides setting `bo_dynamics=1`, the excited state that is being propagated is set with `exc_state_init`. You may choose to set `n_exc_states_propagate` equal to `exc_state_init` to reduce computational cost. Lastly, `n_class_steps` > 0 to run dynamics, where the product of `time_step` and `n_class_steps` defines the length of the trajectory. If an ensemble of adiabatic trajectories is desired, each trajectory must have a different random seed in `rnd_seed`. This can be accomplished with `getexcited.py` (see Subsection 6.6).

Other important inputs that generally do not need to be changed include: `therm_type=1`, `therm_temperature=300`, and `therm_friction=20`.

```
&qmmm
   !***** Geometry Optimization
   maxcyc=0, ! Number of cycles for geometry optimization [0]
   ntpr=1, ! Print results every ntpr cycles [1]
   grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]

   !***** Ground-State and Output Parameters
   qm_theory='AM1', ! Integral type, check Amber's SQM for more options [AM1]
   scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]
   verbosity=1, ! QM/MM output verbosity (0-minimum, 5-maximum)
   ! [1 for dynamics and optimization, 5 for others]
   printdipole=1, ! (0) Unrelaxed transitions, (1) Unrelaxed transitions plus
   ! total molecular, or (2) Unrelaxed/relaxed transitions plus
   ! total molecular [1 for dynamics, 2 for optimization and single-point]
   printbondorders=0, ! (0) No or (1) Yes [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   density_predict=0, ! (0) None, (1) Reversible MD,
   ! or (2) XL-BOMD [0] *** ALL ARE UNDER DEVELOPMENT, DO NOT USE ***
   itrmax=300, ! Max SCF iterations for ground state
   ! (negative to ignore convergence) [300]

   !***** Excited-State Parameters
   exst_method=1, ! CIS (1) or RPA (2) [1]
   dav_guess=1, ! Restart Davidson from (0) Scratch, (1) Previous,
   ! or (2) XL-BOMD [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]
   ftol1=1.0d-8, ! Acceptance tolerance for residual norm [1.0d-5]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   dav_maxcyc=200, ! Max cycles for Davidson diagonalization
   ! (negative to ignore convergence) [100]
   printcharges=0, ! Print (1) or do not print (0) Mulliken charges of QM atoms [0]
   calcxdens=.false., ! Print (.true.) or do not print (.false.)
   ! excited-to-excited transition dipole moments [.false.]

   !***** Solvent Models and External Electric Fields
   solvent_model=0, ! (0) None, (1) Linear response, (2) Vertical excitation,
   ! or (3) State-specific [0]
   potential_type=1, ! (1) COSMO or (2) Onsager [1]
   onsager_radius=2, ! Onsager radius, A (system dependent) [2]
   ceps=10, ! Dielectric constant, unitless [10]
   linmixparam=1, ! Linear mixing parameter for vertical excitation
   ! or state-specific SCF calculation [1]
   cosmo_scf_ftol=1.0d-5, ! Vertical excitation or state-specific
   ! SCF tolerance, eV [1.0d-5]
   doZ=.false., ! Use relaxed (.true.) or unrelaxed (.false) density for
   ! vertical excitation or state-specific COSMO or Onsager [.false.]
   index_of_refraction=100, ! Dielectric constant for linear response
   ! solvent in excited-state, unitless [100] *** UNDER DEVELOPMENT, DO NOT USE ***
   EF=0, ! (0) None or (1) Electric field in ground- and excited-state [0]
   Ex=0, ! Electric field vector X, eV/A [0]
   Ey=0, ! Electric field vector Y, eV/A [0]
   Ez=0, ! Electric field vector Z, eV/A [0]
&endqmmm

&moldyn
   !***** General Parameters
   natoms=12, ! Number of atoms
   ! (must be equal to the number of atoms in system)
   rnd_seed=19345, ! Seed for the random number generator
   bo_dynamics_flag=1, ! (0) Non-BO or (1) BO [1]
   exc_state_init=4, ! Initial excited state (0 - ground state) [0]
   n_exc_states_propagate=4, ! Number of excited states [0]

   !***** Dynamics Parameters
   time_init=0.0, ! Initial time, fs [0.0]
   time_step=0.1, ! Time step, fs [0.1]
   n_class_steps=10000, ! Number of classical steps [1]
   n_quant_steps=4, ! Number of quantum steps for each classical step [4]
   moldyn_deriv_flag=1, ! (0) None, (1) Analytical, or (2) Numerical [1]
   num_deriv_step=1.0d-3, ! Displacement for numerical derivatives, A [1.0d-3]
   rk_tolerance=1.0d-7, ! Tolerance for the Runge-Kutta propagator [1.0d-7]

   !***** Non-Adiabatic Parameters
   decoher_type=2, ! Type of decoherence: Reinitialize (0) Never,
   ! (1) At successful hops, (2) At successful plus frustrated hops...
   ! (3) Persico/Granucci, or (4) Truhlar [2]
   ! *** (3) AND (4) ARE UNDER DEVELOPMENT, DO NOT USE ***
   decoher_e0=0.0, ! Decoherence parameter E0, Hartrees [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   decoher_c=0.0, ! Decoherence parameter C, unitless [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   dotrivial=1, ! Do unavoided (trivial) crossing routine (1) or not (0) [1]
   quant_step_reduction_factor=2.5d-2, ! Quantum step reduction factor [2.5d-2]

   !***** Thermostat Parameters
   therm_type=1, ! Thermostat type: (0) Newtonian, (1) Langevin,
   ! or (2) Berendsen [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   therm_temperature=300, ! Thermostat temperature, K [300]
   therm_friction=20, ! Thermostat friction coefficient, 1/ps [20]
   berendsen_relax_const=0.4, ! Bath relaxation constant for Berendsen
   ! thermostat, ps [0.4] *** UNDER DEVELOPMENT, DO NOT USE ***
   heating=0, ! Equilibrated (0) or heating (1) [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   heating_steps_per_degree=100, ! Number of steps per degree
   ! during heating [100] *** UNDER DEVELOPMENT, DO NOT USE ***

   !***** Output & Log Parameters
   verbosity=2, ! NEXMD output verbosity (0-minimum, 3-maximum)
   ! [2 for dynamics, 3 for optimization and single-point]
   out_data_steps=1, ! Number of steps to write data [1]
   out_coords_steps=10, ! Number of steps to write the restart file [10]
   out_data_cube=0, ! Write (1) or do not write (0) view files to generate cubes [0]
   out_count_init=0, ! Initial count for view files [0]
&endmoldyn

&coord
  6      -7.9798271101     0.6776918081    -0.0532285388
  6      -7.0849928010     1.7602597759     0.0294961792
  6      -5.7058415294     1.5490364812     0.0312760931
  6      -5.2231419594     0.2195333448    -0.0446043010
  6      -6.1050960756    -0.8685564920     0.0220869421
  6      -7.5344099241    -0.6444487634     0.0248126135
  1      -9.0268081830     0.8587716724    -0.0794794940
  1      -7.4774606514     2.7566353436     0.1635393862
  1      -5.0939335779     2.4479885163     0.1481876938
  1      -4.1292016456     0.0999373674    -0.1580811639
  1      -5.6916991654    -1.8878557992     0.1151090966
  1      -8.2838388636    -1.4313798704     0.1051927200
&endcoord

&veloc
     3.3718248255  -5.6032885851  -1.1970845430
     2.5106648755   2.0978837936  -1.0696411897
    -5.9135180273  -3.7505826950   1.1689299883
     7.7194332369   4.8702351843   0.6576546539
    -7.1851218597  -2.0113572464  -0.6329683366
    -1.7276579899   0.3919019235  -0.0257452789
   -17.0279163131   9.9875659542   5.3513734186
    -4.7222747943  18.9640275032  11.9601977632
    10.9539809532  17.0164104392  -9.7113209726
    25.7548696749   2.2116651958  -0.5444198125
   -16.5303708308  -2.3313274630  -3.2147489925
    16.2776787026   2.2582071549   9.3572624705
&endveloc

&coeff
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  1.00  0.00
  0.00  0.00
  0.00  0.00
&endcoeff
```

A.5 Example Input File: Non-Adiabatic Excited-State Trajectory

Shown below is an example input file for a non-adiabatic excited-state trajectory. Important inputs include:

- `scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]`
- `ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]`
- `rnd_seed=19345, ! Seed for the random number generator`
- `bo_dynamics_flag=0, ! (0) Non-BO or (1) BO [1]`
- `exc_state_init=4, ! Initial excited state (0 - ground state) [0]`
- `n_exc_states_propagate=6, ! Number of excited states [0]`
- `time_step=0.1, ! Time step, fs [0.1]`
- `n_class_steps=10000, ! Number of classical steps [1]`

Similar to adiabatic dynamics, the convergence criteria of the ground and excited state must be set with `scfconv` and `ftol0`, respectively, and dynamics with `n_class_steps` > 0. The `rnd_seed` is important for both Langevin dynamics and stochastic hops between states according to the surface hopping algorithm. [34] If an ensemble of adiabatic trajectories is desired, each trajectory must have a different random seed in `rnd_seed`. Non-adiabatic dynamics are set with `bo_dynamics=0`. If an ensemble of trajectories is requested and `getexcited.py` is being used to generate input files, inputs for random seed, initial excited state, nuclear coordinates and velocities, and quantum coefficients should be replaced with their respective flags (i.e. `rnd_seed`, `exc_state_init_flag`, `nucl_coord_veloc`, and `quant_amp_phase`). See Subsection 6.6 for more details.

Other important inputs that generally do not need to be changed include: `n_quant_steps=4`, `decoher_type=2`, `dotrivial=1`, `quant_step_reduction_factor=2.5d-2`. The latter two reduce the `time_step` by a factor of 40 in the vicinity of trivial crossings. Additionally, the thermostat is set with `therm_type=1`, `therm_temperature=300`, and `therm_friction=20`.

```
&qmmm
   !***** Geometry Optimization
   maxcyc=0, ! Number of cycles for geometry optimization [0]
   ntpr=1, ! Print results every ntpr cycles [1]
   grms_tol=1.0d-2, ! Tolerance in eV/A (derivatives) [1.0d-2]

   !***** Ground-State and Output Parameters
   qm_theory='AM1', ! Integral type, check Amber's SQM for more options [AM1]
   scfconv=1.0d-8, ! Ground-state SCF convergence criteria, eV [1.0d-6]
   verbosity=1, ! QM/MM output verbosity (0-minimum, 5-maximum)
   ! [1 for dynamics and optimization, 5 for others]
   printdipole=1, ! (0) Unrelaxed transitions, (1) Unrelaxed transitions plus
   ! total molecular, or (2) Unrelaxed/relaxed transitions plus
   ! total molecular [1 for dynamics, 2 for optimization and single-point]
   printbondorders=0, ! (0) No or (1) Yes [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   density_predict=0, ! (0) None, (1) Reversible MD,
   ! or (2) XL-BOMD [0] *** ALL ARE UNDER DEVELOPMENT, DO NOT USE ***
   itrmax=300, ! Max SCF iterations for ground state
   ! (negative to ignore convergence) [300]

   !***** Excited-State Parameters
   exst_method=1, ! CIS (1) or RPA (2) [1]
   dav_guess=1, ! Restart Davidson from (0) Scratch, (1) Previous,
   ! or (2) XL-BOMD [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   ftol0=1.0d-7, ! Acceptance tolerance (|emin-eold|) [1.0d-5]
   ftol1=1.0d-8, ! Acceptance tolerance for residual norm [1.0d-5]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   dav_maxcyc=200, ! Max cycles for Davidson diagonalization
   ! (negative to ignore convergence) [100]
   printcharges=0, ! Print (1) or do not print (0) Mulliken charges of QM atoms [0]
   calcxdens=.false., ! Print (.true.) or do not print (.false.)
   ! excited-to-excited transition dipole moments [.false.]

   !***** Solvent Models and External Electric Fields
   solvent_model=0, ! (0) None, (1) Linear response, (2) Vertical excitation,
   ! or (3) State-specific [0]
   potential_type=1, ! (1) COSMO or (2) Onsager [1]
   onsager_radius=2, ! Onsager radius, A (system dependent) [2]
   ceps=10, ! Dielectric constant, unitless [10]
   linmixparam=1, ! Linear mixing parameter for vertical excitation
   ! or state-specific SCF calculation [1]
   cosmo_scf_ftol=1.0d-5, ! Vertical excitation or state-specific
   ! SCF tolerance, eV [1.0d-5]
   doZ=.false., ! Use relaxed (.true.) or unrelaxed (.false) density for
   ! vertical excitation or state-specific COSMO or Onsager [.false.]
   index_of_refraction=100, ! Dielectric constant for linear response
   ! solvent in excited-state, unitless [100] *** UNDER DEVELOPMENT, DO NOT USE ***
   EF=0, ! (0) None or (1) Electric field in ground- and excited-state [0]
   Ex=0, ! Electric field vector X, eV/A [0]
   Ey=0, ! Electric field vector Y, eV/A [0]
   Ez=0, ! Electric field vector Z, eV/A [0]
&endqmmm

&moldyn
   !***** General Parameters
   natoms=12, ! Number of atoms
   ! (must be equal to the number of atoms in system)
   rnd_seed=19345, ! Seed for the random number generator
   bo_dynamics_flag=0, ! (0) Non-BO or (1) BO [1]
   exc_state_init=4, ! Initial excited state (0 - ground state) [0]
   n_exc_states_propagate=6, ! Number of excited states [0]

   !***** Dynamics Parameters
   time_init=0.0, ! Initial time, fs [0.0]
   time_step=0.1, ! Time step, fs [0.1]
   n_class_steps=10000, ! Number of classical steps [1]
   n_quant_steps=4, ! Number of quantum steps for each classical step [4]
   moldyn_deriv_flag=1, ! (0) None, (1) Analytical, or (2) Numerical [1]
   num_deriv_step=1.0d-3, ! Displacement for numerical derivatives, A [1.0d-3]
   rk_tolerance=1.0d-7, ! Tolerance for the Runge-Kutta propagator [1.0d-7]

   !***** Non-Adiabatic Parameters
   decoher_type=2, ! Type of decoherence: Reinitialize (0) Never,
   ! (1) At successful hops, (2) At successful plus frustrated hops...
   ! (3) Persico/Granucci, or (4) Truhlar [2]
   ! *** (3) AND (4) ARE UNDER DEVELOPMENT, DO NOT USE ***
   decoher_e0=0.0, ! Decoherence parameter E0, Hartrees [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   decoher_c=0.0, ! Decoherence parameter C, unitless [0.1]
   ! (only for decoher_type = 3 or 4) *** UNDER DEVELOPMENT, DO NOT USE ***
   dotrivial=1, ! Do unavoided (trivial) crossing routine (1) or not (0) [1]
   quant_step_reduction_factor=2.5d-2, ! Quantum step reduction factor [2.5d-2]

   !***** Thermostat Parameters
   therm_type=1, ! Thermostat type: (0) Newtonian, (1) Langevin,
   ! or (2) Berendsen [1] *** (2) IS UNDER DEVELOPMENT, DO NOT USE ***
   therm_temperature=300, ! Thermostat temperature, K [300]
   therm_friction=20, ! Thermostat friction coefficient, 1/ps [20]
   berendsen_relax_const=0.4, ! Bath relaxation constant for Berendsen
   ! thermostat, ps [0.4] *** UNDER DEVELOPMENT, DO NOT USE ***
   heating=0, ! Equilibrated (0) or heating (1) [0]
   ! *** UNDER DEVELOPMENT, DO NOT USE ***
   heating_steps_per_degree=100, ! Number of steps per degree
   ! during heating [100] *** UNDER DEVELOPMENT, DO NOT USE ***

   !***** Output & Log Parameters
   verbosity=2, ! NEXMD output verbosity (0-minimum, 3-maximum)
   ! [2 for dynamics, 3 for optimization and single-point]
   out_data_steps=1, ! Number of steps to write data [1]
   out_coords_steps=10, ! Number of steps to write the restart file [10]
   out_data_cube=0, ! Write (1) or do not write (0) view files to generate cubes [0]
   out_count_init=0, ! Initial count for view files [0]
&endmoldyn

&coord
  6      -7.9798271101     0.6776918081    -0.0532285388
  6      -7.0849928010     1.7602597759     0.0294961792
  6      -5.7058415294     1.5490364812     0.0312760931
  6      -5.2231419594     0.2195333448    -0.0446043010
  6      -6.1050960756    -0.8685564920     0.0220869421
  6      -7.5344099241    -0.6444487634     0.0248126135
  1      -9.0268081830     0.8587716724    -0.0794794940
  1      -7.4774606514     2.7566353436     0.1635393862
  1      -5.0939335779     2.4479885163     0.1481876938
  1      -4.1292016456     0.0999373674    -0.1580811639
  1      -5.6916991654    -1.8878557992     0.1151090966
  1      -8.2838388636    -1.4313798704     0.1051927200
&endcoord

&veloc
     3.3718248255  -5.6032885851  -1.1970845430
     2.5106648755   2.0978837936  -1.0696411897
    -5.9135180273  -3.7505826950   1.1689299883
     7.7194332369   4.8702351843   0.6576546539
    -7.1851218597  -2.0113572464  -0.6329683366
    -1.7276579899   0.3919019235  -0.0257452789
   -17.0279163131   9.9875659542   5.3513734186
    -4.7222747943  18.9640275032  11.9601977632
    10.9539809532  17.0164104392  -9.7113209726
    25.7548696749   2.2116651958  -0.5444198125
   -16.5303708308  -2.3313274630  -3.2147489925
    16.2776787026   2.2582071549   9.3572624705
&endveloc

&coeff
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  0.00  0.00
  1.00  0.00
  0.00  0.00
  0.00  0.00
&endcoeff
```

A.6 Benchmarks: Excited-State Populations and Timings

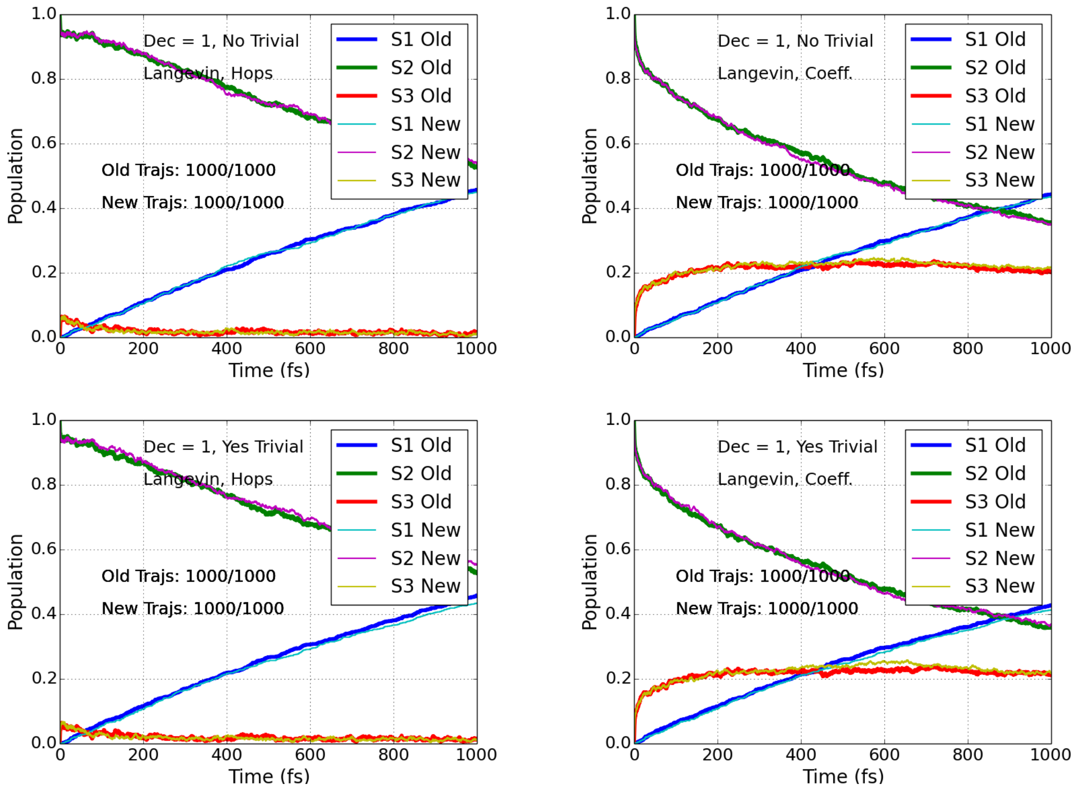


Figure 6: Excited-state populations of a Poly(p-phenylene vinylene) (PPV3) molecule. Each row is separate test, where 3 excited states were propagated. Specification of the tests are labeled in the figures. The **left** figures show populations from surface hopping, while the **right** figures show population from quantum coefficients. Ensembles of 1000 trajectories were used. For each trajectory, the system was initially excited on state $|2\rangle$.

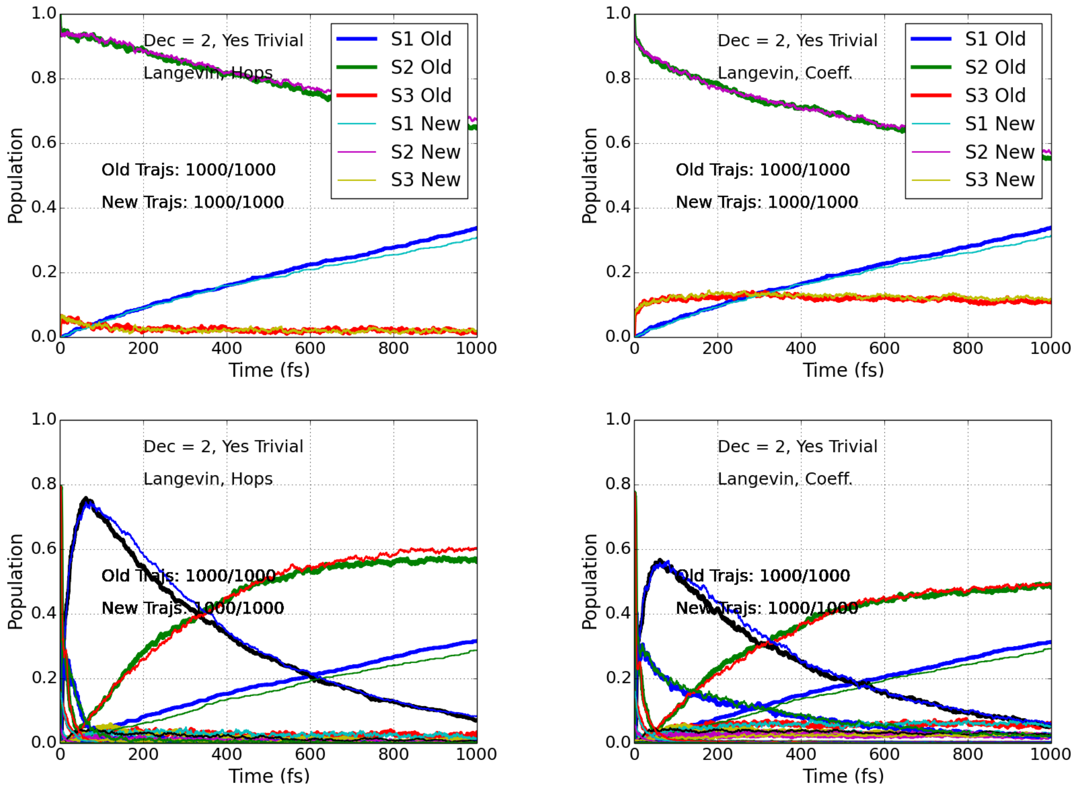


Figure 7: Excited-state populations of a Poly(p-phenylene vinylene) (PPV3) molecule. For **top** and **bottom** tests, 3 and 15 excited states were propagated, respectively. Specification of the tests are labeled in the figures. The **left** figures show populations from surface hopping, while the **right** figures show populations from quantum coefficients. Ensembles of 1000 trajectories were used. **Top**: All trajectories began on state $|2\rangle$. **Bottom**: A distribution of initial excited states were chosen according to the system's theoretical optical absorption spectrum.

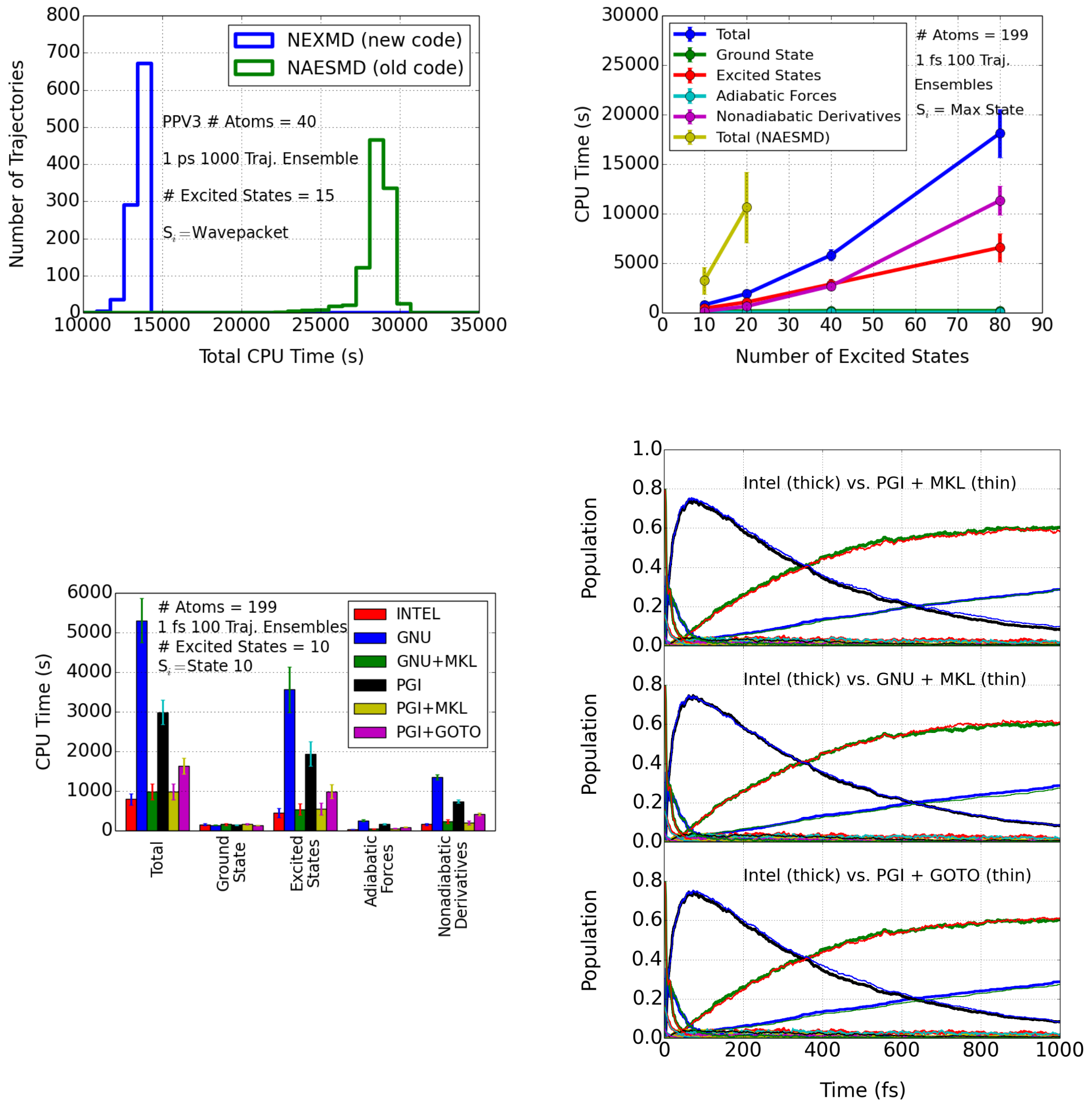


Figure 8: **Upper Left**: Total CPU time histogrammed over all trajectories used in the 1000-trajectory ensembles simulating the non-adiabatic dynamics of the Poly(p-phenylene vinylene) (PPV3) molecule with old and new codes. **Upper Right**: New code (NEXMD) timing performance as a function of the number of excited states propagated using a $\sim$ 200 atom system. The mean and error (standard deviation) in CPU time of the 100-trajectory ensembles are shown. Results of four separate simulations using 10, 20, 40, and 80 excited states are shown. All trajectories began in the highest excited state. Trajectories are 1 fs in length, with 0.10 fs classical step and 4 quantum steps per classical step. All typical non-adiabatic inputs were used. **Lower Left**: New code (NEXMD) timing performance as a a function of different compilers using a $\sim$ 200 atom system. The mean and error (standard deviation) in CPU time of 100-trajectory ensembles are shown. A total of 10 excited states were propagated and all trajectories began in the 10^{th} excited state. All typical non-adiabatic inputs were used. **Lower Right**: New Code (NEXMD) consistency performance test against several compilers using 1000-trajectory ensembles simulating the non-adiabatic dynamics of the Poly(p-phenylene vinylene) (PPV3) molecule. Version used are: GCC 5.3.0, PGI 16.10, GOTO, goto2r1.13, MKL/11.4.1