NAD Input Manual

Sapunar, M

February 9, 2021

Three files are required for running a trajectory.

1. The main input file containing all of the options for the program.
2. A file containing the initial geometry.
3. A file containing the initial velocity.

The name of the file is passed to the program through the command line (if no file name is passed, the default input file is dynamics.in). The file is split into sections with options concerning parts of the program. For details on the options, see below. The minimum input is the $method section which gives the path to a driver program/script which communicates between the dynamics and the electronic structure package. It passes new geometries to the electronic structure package, runs a calculation, and reads the energies, gradients and couplings from the output. In the case of QM/MM dynamics, a second script is used to communicate to the MM package. The QM(MM) driver expects a directory which contains all the input required for a QM(MM) calculation. By default this directory is "qmdir"("mmdir"), but can be changed using the QMDIR(MMDIR) environment variable.

The geometry file (called "geom" by default) contains one line for each atom. Each line contains the atom symbol, atom mass, coordinates and type of atom. The type in the final column is either "q" (for QM atoms) or "m" (for MM atoms). The velocity file contains just the initial velocity vector for each atom.

A stopped or finished trajectory can be restarted by adding the $restart keyword to the input file. In this case, the backup file is read before restarting the calculation.

Program options

method Section $method contains paths to the driver programs for running the calculations.
$method

qm	**No default.** Path to driver for QM calculation.
mm	**No default.** Path to driver for MM calculation.
overlap	**No default.** Command line call for code to calculate the overlap matrix. Run "cis-overlap.exe -h" for command line options.

system Section $system contains information about the dimensions of the system and the initial conditions.
$system

nstate #	**1** Number of states in the QM system.
istate #	**nstate** Initially populated state. In a FSSH calculation, the initial wave function coefficient of this state will be 1, while others will be 0.
geometry	**geom** Name of the geometry file, containing the masses and initial positions of the atoms.
velocity	**veloc** Name of the velocity file, containing the initial velocities of the atoms.
ndim #	**3** Number of dimensions per atom.

output Section $output contains options for program output and backup.
$output

results_dir	**Results** Name of the main output file.

(no)print — **1-10**
Select which output files to print.
- [1] energy.dat and mm.dat.
- [2] trajectory.xyz.
- [3] geometry, atomic units.
- [4] velocity, atomic units.
- [5] gradient, atomic units.
- [6] cwf.dat, wave function coefficients.
- [7] overlap, overlap matrix for each step.
- [10] oscill.dat, QM oscillator strengths.
- [11] qm_traj.xyz, XYZ file containing only QM atoms.

bufile — **backup.dat**
Name of backup file. The current state is read from this file when restarting the program using the $restart keyword.

buinterval # — **1**
The backup file is written every # steps.

dynamics Section $dynamics contains options for the propagation of the nuclear coordinates.
$dynamics

time/max_time # — **10**
Total time (in fs) for the propagation of nuclear coordinates.

tstep # — **0.5**
Time step (in fs) for the propagation of nuclear coordinates.

orient # — **0**
Select how to handle translation/rotation.
- [0] do nothing.
- [1] keep molecule in center of mass.
- [2] project rotation and translation during dynamics.

end_state # — **0**
End dynamics after reaching target state.

end_state_time # — **0.0**
Allow dynamics to run for an additional # fs after reaching target state.

max_toten_d # — **1.0**
End dynamics if drift in total energy is larger than # eV.

max_toten_d_step # — **0.2**
End dynamics if change in total energy during a single step is larger than # eV.

constraints Section $constraints contains constraints to be kept while propagating nuclear coordinates. The section contains 1 line defining each constraint. Currently, only bond lengths can be constrained.
$constraints

b #1 #2 #3 #4 — **No default**
Freeze bond between atoms #1 and #2 to value #3. If present, #4 is the tolerance for the convergence of the constraint.

surfhop Section $surfhop contains options for the surface hopping procedure and the propagation of the electronic wave function. If the "$surfhop off" option is given, the rest of the section is not read and dynamics are performed on the initial adiabatic state without hopping.
$surfhop (off)

lz — **off**
Instead of the fewest-switches surface hopping algorithm, use Landau-Zener formula to determine hops between surfaces. Most of the other options in the section are ignored.

fssh	**adiabatic** Use fewest-switches surface hopping algorithm. Possible options: – [adiabatic] Propagate electronic wave function coefficients in the adiabatic basis. – [diabatic] Propagate electronic wave function coefficients in the diabatic basis using local diabatization.
tdse_steps #	**10000** Number of substeps for electronic wave function propagation between the nuclear time step at t and at $t + \Delta t$. Energies and couplings are interpolated during the propagation based on the overlap/nadvec and energy options. A hop can occur in any substep of the propagation. Probabilities printed in the main output file are sums of probabilities from each substep.
decoherence	**nldm** Method used for applying a decoherence correction to the electronic wave function. Possible options: – [off] No decoherence correction. – [nldm] Non-linear decay of mixing algorithm.
overlap	**npi** Use overlaps between wave functions at the start and end of nuclear time step to calculate time-derivative couplings (TDCs) between states. The TDCs can be interpolated during the propagation between nuclear time steps. Possible options: – [constant] Use the finite-differences method to calculate TDCs at $t + \Delta t/2$ and use this value for all substeps. – [linear] Use linear interpolation/extrapolation between the TDCs at $t - \Delta t/2$ and $t + \Delta t/2$. – [npi] Use the norm preserving interpolation method to calculate the average TDCs during the nuclear time step and use this value for all substeps.
phase	**2** Method of handling phase of wave functions during dynamics. – [0] No phase matching. – [1] Match phase of adiabatic states. – [2] Match phase of diabatic states (using assignment problem solution).
nadvec	**linear** Use nonadiabatic coupling vectors to calculate time-derivative couplings between states. The TDCs can be interpolated during the propagation between nuclear time steps. Possible options: – [constant] Use the TDCs at $t + \Delta t$ for all substeps. – [linear] Use linear interpolation between the TDCs at t and $t + \Delta t$.
energy	**linear** Method for interpolation of energies during the propagation between nuclear time steps. – [constant] Use energies at $t + \Delta t$ for all substeps. – [step] Use energies at t until $t + \Delta t/2$ and energies at $t + \Delta t$ for the second half of the substeps – [linear] Use linear interpolation between energies at t and $t + \Delta t$
velocity	**vel** Method of rescaling velocity to conserve total energy after a hop. – [off] No phase matching. – [vel] Uniformly rescale velocity. – [gdif] Rescale velocity along vector given by difference of gradients on initial and final surface. – [nadvec] Rescale velocity along nonadiabatic coupling vector.
frustrated	**reverse** Method of treating frustrated hops (hops rejected due to lack of energy). – [continue] Continue trajectory along previous state. – [reverse] Continue trajectory along previous state, but also invert velocity along the direction along which velocity rescaling was attempted.

skip_state	**none** Give list of electronic states which will not be included in the calculation of the couplings. (Example: skip_state 1-2, 5)
seed #	**-1** Give a seed for the random number generator. If seed is < 0, a seed is generated automatically based on the current time and job ID.