Description of the Reactive Flux method implemented in PCET 5.4 code

Christine Schwerdtfeger

Department of Chemistry, University of Illinois at Urbana-Champaign, Urbana, IL 61801

I. INTRODUCTION

Capabilities to perform reactive flux calculations have been added to the PCET code. This allows the user to start the trajectory at a dividing surface (chosen by the user). The trajectory begins and propagates "backwards" in time until either reactants or products are reached, using approximate hopping probabilities to check for hops in the reverse time propagation. If reactants or products are reached, the trajectory is then propagated forward in time to its initial coordinates at the dividing surface, and the time dependent wavefunction coefficients are updated normally. If the trajectory reached reactants in the reverse trajectory, it is then propagated from the dividing surface forward in time to determine if it ultimately reaches products. The probability that the trajectory would occur is also computed as the trajectory moves forward in time. When the trajectory reaches reactants or products in the forward direction, the trajectory is stopped. The values of F_n, F_d (F_d is the total number of forward crossings sampled and appropriately weighted to obtain a MaxwellBoltzmann flux distribution, and F_n is the number of these forward crossings that started in reactant well and ended in the product region, accounting for multiple crossings), and trajectory weightings W are printed at the end of each trajectory. The value of the transmission coefficient $\kappa = F_n/F_d$, and κ is multiplied by the transition state theory (TST) rate constant, $k = \kappa k_{\mathrm{TST}}$. Please see Ref. 1 for more details about the theory of reactive flux calculations.

II. IMPLEMENTATION

To perform this calculation, the user should choose a dividing surface by setting the `ZE0=` keyword (in the `DYNAMICSET2` group). The dividing surface is the energy gap between the reactant and product states. The value set for the `ZE0` keyword corresponds to the energy gap coordinate where the trajectory should begin. The `REACTIVE_FLUX` keyword must also be specified in the `DYNAMICSET2` group to perform this calculation also.

Execution of code:

1. Turn on `REACTIVE_FLUX`. Ensure `MDQT` is on. Ensure `DECOHERENCE` is off. Ensure Onodera-1 solvent model is being used. Onodera-1 is chosen using `EPSMODEL=ONODERA`
2. Allocate arrays for storage of energy gap, velocity, hopping probability, hop attempts
3. Begin trajectory at the energy gap chosen by the user (using the `ZE0=` keyword)
4. Choose initial velocity according to a Boltzmann distribution; reverse velocity if it is negative (start trajectory moving towards reactants).
5. Choose initial adiabat using the cumulative distribution function
6. Initialize reactive flux storage arrays to 0
7. Call the routine that propagates the trajectory backwards in time. This routine computes an approximate hopping probability and tracks the energy gap, velocity, approximate hopping probability, and hop attempts. The backwards propagation stops when the energy gap of the trajectory passes the energy gap at the minimum of the reactant or product well.
8. If trajectory reached reactants or products in the reverse trajectory, set `successfulreverse` variable to `True`.
9. Initialize time-dependent wavefunction coefficients to the pure state occupied in the last step of the reverse trajectory (at reactant or product minimum). On occupied adiabat, `coefficient=1.0` and on unoccupied adiabats, `coefficient=0.0`
10. Reverse sign of velocities stored, and call the routine that retraces the backwards trajectory from the reactant or product minimum towards the dividing surface (`ZE0` value). In this propagation, true time-dependent wavefunctions and hopping probabilities are computed. The running value of the weights w_μ is computing using the true and approximated hopping probabilitites.

11. If trajectory reached reactants in backwards trajectory, propagate the trajectory forward in time from the dividing surface (`ZE0`) normally.

12. Print out values of F_n, F_d, and W needed to weight the trajectory.

III. PROGRAMMING NOTES

- `REACTIVE_FLUX` keyword in `DYNAMICSET2` group that requests a reactive flux calculation. In the code, the variable `reactive_flux` is set to true.
- Reactive flux can **only** be turned on when the `MDQT` keyword is also turned on.
- Decoherence **cannot** be used with the reactive flux algorithm.
- Reactive flux can currently **only** be used with the Onodera-1 solvent model. The Onodera-2 model has a memory term that is needed to propagate, and we are not sure how this should work going backwards in time.
- The keywords `ISTATE` and `DSTATE` will be ignored if this is a reactive flux calculation. This is because the initial state is chosen at the beginning of each trajectory. The routine which chooses the initial state is called `reactiveflux_choose_initial_state`.
- With reactive flux, there is no distribution in initial energy gaps around the center (ZE0) chosen. The initial energy gap is *always* the energy gap specified using the ZE0 keyword. However, the initial velocity is chosen from a Boltzmann distribution. If the velocity is negative, the sign of the velocity is reversed so that it is going towards the reactants.

Arrays allocated for reactive flux are:

```
z1_storage(1:nsteps)
vz1_storage(1:nsteps)
vz1_storage_beforehop(1:nsteps)
fnj_storage(1:nsteps,1:nstates_dyn,1:nstates_dyn)
switch_attempt(1:nsteps)
occupied_adiabat(1:nsteps)
occupied_adiabat_beforehop(1:nsteps)
switch_attempt_state(1:nsteps)
```

These arrays are needed because we must store the information about energy gap (`z1_storage`), velocity (`vz1_storage`), velocity before a hop (`vz1_storage_beforehop`), hopping probability (`fnj_storage`), switch attempts (`switch_attempt`), currently occupied adiabat (`occupied_adiabat`), occupied adiabat before a hop (`occupied_adiabat_beforehop`), and if a switch is attempted at a particular timestep (`switch_attempt_state`). We need to store this info so that we can retract the trajectory forward in time and compute the probability that this trajectory would have occurred (if it had started at the reactant rather than the dividing surface).

subroutine reactiveflux_choose_initial_state is in `module_propagators_et2.f90`. This routine takes in the number of states in the dynamics, the temperature, and variables for the initial state and currently occupied state. The cumulative distribution function is used to select the initial state.

subroutine initialize_reactiveflux_variables is in `module_propagators_et2.f90`. This routine is called at the beginning of a reactive flux trajectory. It zeros all of the arrays that store data for the reverse trajectory. It also initializes the values of W (1.0), F_n (0.0), F_d (0.0), and α (0.0). It sets the limit of alpha to 100, which is the maximum number of times a trajectory can cross the dividing surface. The value of `successfulreverse`, which is a logical indicating whether the trajectory reached the reactant or product state in the reverse trajectory, is initialized to `false`. The logical `normalforward` is initialized to false and is set to true if the trajectory reaches the reactant state in the reverse trajectory. It is important to know if the trajectory reaches reactants or products (`successfulreverse=True`) in the reverse trajectory because only those trajectories go into computing the values of F_n and F_d. If `successfulreverse` is true, the trajectory is propagated from its final point (in negative time at the reactant or product minimum) to the $t = 0$ starting point. If the reverse trajectory made it to reactants, `normalforward` is also set to `true`. This indicates that after the trajectory gets from negative time back to 0, the the trajectory should be propagated normally forward in time from the dividing surface.

subroutine reverse_time_propagation is in `dynamicset2.f90`. This routine runs the trajectory from the dividing surface back in time. If the trajectory reaches the reactant or product state before the maximum number of timesteps is reached, then the value of the logical `successfulreverse` changes to `True` and the reverse trajectory stops. Throughout the reverse trajectory, values of energy gap, velocity, occupied adiabat, and the approximated hopping probabilities are stored. This routine calls the `calculate_switch_prob_function` routine, which approximates the hopping probability according to a function that depends on nonadiabatic coupling. Because we don't have real TDSE coefficients, we cannot compute the real hopping probability in the reverse trajectory. The classical propagation backwards in time is also performed in this routine. If there is a hop or hop attempt, that is recorded, and the occupied adiabat is also recorded. After each timestep, the energy gap is checked and compared with that of the reactant and product states. If it is determined that the trajectory crossed the energy gap at the reactant or product minimum while it is occupying adiabat 1, then the values of successfulreverse and normalforward and the index of the final timestep in the reverse trajectory are set accordingly. If the number of timesteps in the reverse trajectory exceeds the maximum, the reverse propagation is also stopped.

subroutine calculate_switch_prob_function is in `module_propagators_et2.f90`. This routine approximates a hopping probability using the function `switch_probability(adiabat j)` $= 0.5 * (1 - e^{|\mathbf{v}\,\mathbf{d}_{ij}\delta t|})$. It then stores the hopping probability in the `fnj_storage` array in case it is needed for the forward trajectory.

subroutine forward_time_propagation_quantum_only is in `dynamicset2.f90`. This routine is called if `successfulreverse` is set to `True`, meaning that the reactant or product state was reached in the reverse trajectory. The purpose of this routine is to start at the last timestep in the reverse trajectory and propagate forward towards the $t = 0$ starting point. The coefficients of the TDSE are set to a pure state (on the adiabat occupied at the end of the reverse trajectory) before entering this routine. The trajectory is propagated forward in time using the values of energy gap and velocity that were stored during the reverse trajectory. Hops also occur in this trajectory in the same timesteps where they occurred in the reverse trajectory. In this routine, the value of $W_m u$, which relates the true hopping probability to the approximated one used in the reverse trajectory, is also computed.

subroutine calculate_w_mu in `module_propagators_et2.f90`. The quantity w_μ, relates the true hopping probability to the one that is approximated in the `calculate_switch_prob_function` routine which is used in the reverse trajectory. The value of w_μ at all timesteps are multiplied to give the overall value of W, used in weighting the probability that the trajectory actually would occur if it had began at the reactant state rather than the dividing surface (`ZE0` value).

IV. EXAMPLE SET OF INPUT FILES

```
====== File: const-gas.d ==========================================
XYZ
2-state ET model

De  0.000  0.000  -2.50000
Ae  0.000  0.000   2.50000

ATOMIC CHARGES
 -0.250  -0.250   0.250   0.250
  0.250   0.250  -0.250  -0.250

PT interface: donor-hydrogen-acceptor
 1 1 1
ET interface: donor-acceptor
 1  2

====== Filename: const.ini ========================================
C==== CONSTANT POTENTIAL PARAMETERS (kcal/mole, Angstroms)

      V12    0.0
      V34    0.0
```

```
      V13    0.1
      V24    0.1
      V14    0.0
      V23    0.0

C     Energy biases
      CORR   0.0  0.0  0.0  0.0


====== Filename: const-sol.d ========================================
XYZ RADIUS MODFE=20
2-state ET model

De  0.000  0.000  -2.50000
Ae  0.000  0.000   2.50000

RADIUS
De 2.5
Ae 2.5

ATOMIC CHARGES
 -0.250  -0.250   0.250   0.250
  0.250   0.250  -0.250  -0.250

PT interface: donor-hydrogen-acceptor
 1 1 1
ET interface: donor-acceptor
 1  2


====== Filename: et-o2c-V01-dG-top+5.inp ==============================
JOBNAME=et-o2c-er23-V01-dG-top+5
2-state ET model test
CHARGE=0
HGAS(constant,pars=const.ini,GEOM=const-gas.d)
SOLV(TSET,EPS0=85.748308,EPS8=1.0,GSOLV1=0.0,GSOLV2=0.0,
ER=23.4656,GEOM=const-sol.d)
DYNAMICSET2(MDQT,ADIAB=1/3,ZE0=0,NSTATES=2,DSTATE=1
INTERPOLATION=LINEAR,T=298.0,SEED=30184,
EPSMODEL=ONODERA,EPSPARS,TAU0=0.00200631,TAUD=6.8867,
EPS0=85.74308,EPS8=1.0000,
TSTEP=0.0000025,NSTEPS=100000000,NTRAJ=20000,NQSTEPS=40,
MAXNQSTEPS=4000,NDUMP=400,NDUMP6=40000,REACTIVE_FLUX)

END
```

---

[1] S. Hammes-Schiffer and J. C. Tully, J. Chem. Phys. 103, 8528 (1995); https://doi.org/10.1063/1.470162