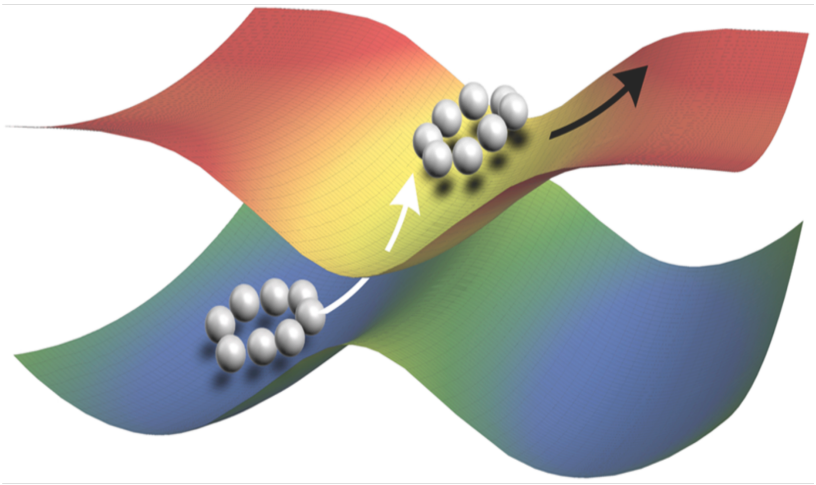
SHARP Pack

Version 2.0

Sep 05, 2025

CONTENTS:



SH|a⟩RP

CHAPTER

ONE

INTRODUCTION

Non-adiabatic dynamics—quantum transitions between electronic and/or vibronic states—is a critical process in electrochemical and photoelectrochemical reactions, including those involved in natural and artificial energy conversion systems. [1, 2]

Simulating these processes accurately in condensed phases with thousands of degrees of freedom (DOFs), especially when nuclear quantum effects (NQEs) like proton tunneling and zero-point energy are significant, is computationally demanding. Mixed quantum-classical dynamics (MQCD) methods address this by treating only light particles quantum-mechanically and treating the environment classically.

One of the most widely used MQCD methods is Fewest-Switches Surface Hopping (FSSH) [3], where the quantum wavepacket is represented by classical trajectories traveling on a single adiabatic surface interrupted by hops to other surfaces. However, independent classical trajectories used in FSSH are not able to capture decoherence or NQEs accurately.

Approaches like Centroid Molecular Dynamics (CMD) [4] and Ring Polymer Molecular Dynamics (RPMD) [5] offer a way to model NQEs, but have limitations for multi-electron transfer or real-time electronic coherence effects. Ring Polymer Surface Hopping (RPSH) [6, 7] is a method that combines RPMD and FSSH to overcome these challenges.

SHARP Pack [8] —short for *Surface Hopping And Ring Polymer* package—is a modular, parallelized, and user-friendly software that implements RPSH, enabling simulations from model systems to more realistic applications.

Features of SHARP Pack:

- Supports both FSSH and RPSH simulations.
- Provides a variety of model Hamiltonians including 1D Tully to spin-boson model with Debye/Ohmic spectral density.
- Handles flexible initial conditions and user-defined control parameters.
- Implements different schemes for velocity rescaling within the surface hopping algorithm.
- Propagates dynamics using both velocity-Verlet and Langevin dynamics schemes.
- Enables serial as well as parallel simulations with trajectory distribution across multiple cores.
- Offers PIMD sampling with PILE thermostat for initial sampling.
- Option for Diagonal Born-Oppenheimer correction (DBOC) for accurate quantum dynamics.
- Presents option for decoherence correction to improve population and coherence evolution.
- Produces a wide range of outputs: energies, NACs, populations, branching probablities, etc.
- Provides ready-to-use job scripts for local runs, HPC cluster submissions, and branching probability analysis.
- Supplies comprehensive tutorials and example cases with scripts for visualization.

CHAPTER

TWO

INSTALLATION

The source code is compiled using the provided `Makefile` in the `source` directory. The package has been tested under **Linux** with the **Intel Fortran compiler**.

2.1 Required Libraries

To compile and run SHARP Pack, the following external libraries must be installed: **BLAS**, **LAPACK**, and **FFTW3**.

2.2 Compilation Steps

- Clone the **SHARP Pack 2.0** source code from the GitHub repository:

```
$ git clone https://github.com/fashakib/SHARP_pack.git
```

- Once the package has been downloaded, navigate to the directory containing the source codes:

```
$ cd SHARP_pack/source
```

- Open the `Makefile` and update the library paths to point to your installed libraries. For example, edit the lines:

```
LIBPATH = -L/path/to/blas \
        -L/path/to/lapack \
        -L/path/to/fftw3
```

> **Note**
>
> Make sure to replace `/path/to/...` with the correct installation paths on your system.

- Compile the code by running:

```
$ make
```

- Upon successful compilation, the `bin/` directory will contain the main executable, `sharp.x`, and the auxiliary executables, `average.x`, and `average2db.x`. To verify, run:

```
$ ls ../bin/sharp.x
```

- Add the `bin` directory to your `PATH` for convenience:

```
export PATH=$PATH:/path/to/SHARP_pack/bin
```

Now the **SHARP Pack** is ready to run RPSH simulations.

Attention

When using SHARP Pack, please cite the following papers:

- D. K. Limbu and F. A. Shakib, Software Impacts 19, 100604 (2024).
- D. K. Limbu and F. A. Shakib, J. Phys. Chem. Lett. 14, 8658–8666 (2023).

CHAPTER

THREE

MODELS

There are several model systems implemented in **SHARP Pack**, which are summarized in the following table.

Table 3.1: Models

Keyword	Description
tully1	Tully Model-I: Simple avoided crossing model [3]
tully2	Tully Model-II: Dual avoided crossing model [3]
tully3	Tully Model-III: Extended coupling with reflection model [3]
morse1	Morse Model-I: photo-dissociation model [9]
morse2	Morse Model-II: photo-dissociation model [9]
morse3	Morse Model-III: photo-dissociation model [9]
db2lchain	2-state model coupled with linear chain model [10, 11]
db3lchain	3-state super-exchange model with linear chain model [12]
superexchange	3-state super-exchange model [13]
spinboson	Spin-Boson model coupled to N-harmonic bath [14]
dboc1	Flat Born-Oppenheimer PES model [15, 16]

3.1 Tully Models

Diabatic Hamiltonians and parameters for Tully models [3] are:

Model I

$$V_{11}(R) = \begin{cases} A(1 - e^{-BR}) & \text{if } R > 0 \\ -A(1 - e^{BR}) & \text{if } R < 0 \end{cases}$$

$$V_{22}(R) = -V_{11}(R)$$

$$V_{12}(R) = V_{21}(R) = Ce^{-DR^2}$$

Model II

$$V_{11}(R) = 0$$

$$V_{22}(R) = -Ae^{-BR^2} + E_0$$

$$V_{12}(R) = V_{21}(R) = Ce^{-DR^2}$$

Model III

$$V_{11}(R) = A, \quad V_{22}(R) = -A$$

$$V_{12}(R) = V_{21}(R) = \begin{cases} Be^{CR}, & \text{if } R < 0 \\ B(2 - e^{-CR}), & \text{if } R > 0 \end{cases}$$

Table 3.2: Tully Model Parameters (in a.u.)

Parameter	Model I	Model II	Model III
A	0.01	0.1	6×10^{-4}
B	1.6	0.28	0.1
C	0.005	0.015	0.9
D	1.0	0.06	–
E_0	–	0.05	–

3.2 Morse Potential Models

Diabatic Hamiltonians and parameters for three Morse models [9] are:

$$V_{ii}(R) = De_i \left(1 - e^{-\beta_i (R - Re_i)}\right)^2 + c_i$$

$$V_{ij}(R) = A_{ij} e^{-a_{ij}(R - R_{ij})^2}, \quad V_{ji}(R) = V_{ij}(R)$$

Table 3.3: Numerical values of the parameters related to the three Morse potential models (in a.u.)

Model-I	De_i	β_i	Re_i	c_i	A_{ij}	R_{ij}	a_{ij}
V_{11}	0.003	0.65	5.0	0.00			
V_{22}	0.004	0.60	4.0	0.01			
V_{33}	0.003	0.65	6.0	0.006			
V_{12}					0.002	3.40	16.0
V_{23}					0.002	4.80	16.0
Model-II	De_i	β_i	Re_i	c_i	A_{ij}	R_{ij}	a_{ij}
V_{11}	0.02	0.65	4.5	0.00			
V_{22}	0.01	0.40	4.0	0.01			
V_{33}	0.003	0.65	4.4	0.02			
V_{12}					0.005	3.66	32.0
V_{13}					0.005	3.34	32.0
Model-III	De_i	β_i	Re_i	c_i	A_{ij}	R_{ij}	a_{ij}
V_{11}	0.02	0.40	4.0	0.02			
V_{22}	0.02	0.65	4.5	0.00			
V_{33}	0.003	0.65	6.0	0.02			
V_{12}					0.005	3.40	32.0
V_{13}					0.005	4.97	32.0

3.3 Linear Chain Model

Linear Chain Model is **two-level/three-level** system copuled to a signle atom in N-atom linear chain, with the following anharmonic, nearest-neighbor potential energy function:

$$V(\mathbf{R}) = \sum_{i=1}^{N} V_M(R_i - R_{i+1})$$

where

$$V_M(R) = V_0 \left(a^2R^2 - a^3R^3 + 0.58a^4R^4\right)$$

and R_{N+1} is a fixed position and the atom farthest from the quantum systerm (R_N) is connected to Langevin dynamics with friction constant γ. [10, 11, 12]

Table 3.4: Simulation Parameters for **two-level** Linear Chain Model

Parameter	Value	Unit
N	20	
m	12.0	amu
V_0	175.0	kJ/mol
a	4.0	Å^{-1}
γ	10^{14}	s^{-1}
$\Delta = \epsilon_2 - \epsilon_1$	8.0	kJ/mol
d_{12}	-6.0	Å^{-1}

Table 3.5: Simulation Parameters for **three-level** Superexchange Linear Chain Model

Parameter	Value	Unit
N	20	
m	12.0	amu
V_0	175.0	kJ/mol
a	4.0	Å^{-1}
γ	10^{14}	s^{-1}
ϵ_1	0	kJ/mol
ϵ_2	39.0	kJ/mol
ϵ_3	13.0	kJ/mol
d_{12}	-6.0	Å^{-1}
d_{23}	8.0	Å^{-1}
d_{13}	0	Å^{-1}

3.4 Super Exchange Model

Diabatic Hamiltonians and parameters for 3-state super exchange model [13] are defined as:

$$V_{ii}(R) = A_i$$

$$V_{ij}(R) = V_{ji}(R) = B_{ij}\, e^{-R^2/2}$$

Table 3.6: Super Exchange model parameter (in a.u.)

	$i=1$	$i=2$	$i=3$	$ij=12$	$ij=23$	$ij=13$
A_i	0	0.01	0.005			
B_{ij}				0.001	0.01	0

3.5 Spin–Boson Model

The **spin-boson model** provides a theoretical framework for studying non-adiabatic dynamics in a condensed-phase environment. The corresponding Hamiltonian [14] is given by:

$$H = H_s + H_b + H_{sb}.$$

Two–level system Hamiltonian:

$$H_s = \epsilon\sigma_z + \Delta\sigma_x = \begin{pmatrix} \epsilon & \Delta \\ \Delta & -\epsilon \end{pmatrix},$$

with bias ϵ and coupling Δ is interacting with a bath of harmonic oscillators defined with,

Bath Hamiltonian:

$$H_b = \sum_j \left(\frac{P_j^2}{2M_j} + \frac{1}{2} M_j \omega_j^2 R_j^2 \right).$$

Two-level system bilaterally interacting with a harmonic bath, defined as System–bath coupling Hamiltonian:

$$H_{sb} = \sigma_z \sum_j c_j R_j = \begin{pmatrix} \sum_j c_j R_j & 0 \\ 0 & -\sum_j c_j R_j \end{pmatrix}.$$

3.6 Spectral Density

General definition for N bath modes [17, 18]:

$$J(\omega) = \frac{\pi}{2} \sum_{j=1}^{N} \frac{c_j^2}{M_j \omega_j} \, \delta(\omega - \omega_j),$$

where

- M_j = mass of j-th oscillator,
- c_j = coupling constant between the system and the j,-th oscillator,
- ω_j = oscillator frequency.

3.6.1 Debye form

$$J_D(\omega) = \frac{E_r}{2} \frac{\omega \, \omega_c}{\omega^2 + \omega_c^2},$$

where

- E_r = reorganization energy,
- ω_c = characteristic bath frequency.

Debye Bath Discretization

Frequencies and couplings are discretizeda as [19, 20]:

$$\omega_j = \omega_c \cdot \tan\left(\frac{j}{N} \tan^{-1}\left(\frac{\omega_{\max}}{\omega_c} \right) \right),$$

$$c_j = \omega_j \sqrt{\frac{M_j E_r}{\pi N} \tan^{-1}\left(\frac{\omega_{\max}}{\omega_c} \right)},$$

with $\omega_{\max} = 20\omega_c$, a high-frequency cutoff.

3.6.2 Ohmic form

$$J_O(\omega) = \frac{\pi}{2}\,\alpha\,\omega\,e^{\omega/\omega_c},$$

where

- α = Kondo parameter that controls the strength of the coupling,
- ω_c = characteristic bath frequency.

Ohmic Bath Discretization

Frequencies and couplings are discretizeda as:

$$\omega_j = \omega_c \cdot \log\left(1 - \frac{j}{1+N}\right),$$

$$c_j = \omega_j \sqrt{\frac{M_j \alpha}{1+N}},$$

CHAPTER

FOUR

SIMULATIONS

To run a simulation with **SHARP Pack**, only a single input file named `param.in` is required. This file contains all simulation details specified through predefined keywords.

> **Note**
>
> The file must be located in the same directory where the executable `sharp.x` is launched, unless a path is explicitly provided.

4.1 Keywords

Table *Input Keywords* lists all available input parameter keywords and their usage.

Table 4.1: Input Keywords

Keywords	Arguments	Descriptions
acckval	*yes*	If TRUE, writes accumulated result with k-value; useful for branching probability calculation(s) of **Tully model**
	no	No accumulated result(s)
approximation	CA	Centroid approximation RPSH (RPSH-CA)
	BA	Bead approximation RPSH (RPSH-BA)
dboc	BF	DBOC corrected force(s) at bead level
	CF	DBOC corrected force(s) at centroid level
decoherence	none	No decoherence correction
	damp	Expontial damping decoherence
dynamics	rpmd	RPMD dynamics with surface hopping
	trpmd	T-RPMD dynamics with surface hopping
iprint	n	Print data every n timesteps
iseed	n	Random number generator seed
nbead	n	Number of bead(s)
ncore	n	Number of core(s): (1-Serial), (n-Parallel)
nequil	n	equailibrate steps **if pimd sampling**
nmode	fft	FFT tranform for normal mode
	matrix	use normal mode tranformation matrix for normal mode
nparticle	n	Number of simulating particle(s)
nsample	n	sample run steps **if pimd sampling**
nstate	n	Number of electronic states; Default based on Model
nstep	n	Number of simulation steps
ntraj	n	Number of trajectories

continues on next page

Table 4.1 – continued from previous page

Keywords	Arguments	Descriptions
model	s	Model name string from Table *Models*
pimd	pile τ	PIMD sampling of postion and velocity with PILE thermostat
pinit	f	Initial momentum (a.u.)
rsamp	fraction	Sample centroid bead from Gaussian distribution and other bead(s) position based on fraction of de-Broglie length
	gaussian	Sample initial bead(s) position from Gaussian distribution
	wigner	Sample initial bead(s) position from Wigner distribution
rmap	no	No mapping ring polymer centroid to dividing surface after sampling **rsamp**
	r0	Mapping ring polymer centroid to dividing surface after sampling **rsamp**
rmode	direct	Direct ring polymer sampling of **rsamp**
	norm	Normal mode ring polymer sampling of **rsamp** for **gaussian**
rundtail	*yes*	Print run-time details of the simulation
	no	No details of simulation
temperature	f	Temperature (K)
tstep	f	Simulation time-step (a.u.)
vsamp	fixed	Deterministic initial momentum
	gaussian	Sample initial momenta from Gaussian distribution
	wigner	Sample initial momenta from Wigner distribution
vrescale	CL	Centroid-approximation level of velocity rescaling of ring-polymer for energy conservation
	BL	Bead-approximation level of velocity rescaling of ring-polymer for energy conservation
vreverse	never	Never reversal (NR) velocity for frustrated hop(s)
	always	Always reversal (VR) of velocity for frustrated hop(s)
	delV1	Velocity reversal based on Truhlar's ΔV scheme
	delV2	Velocity reversal based on Subotnik's ΔV^2 scheme
finish	***	End of the input file

Legend:

n – integer, f – real number, s – string, blue – denotes default value.

Table 4.2: Spin-boson model specific keywords

Keywords	Arguments	Descriptions
model	spinboson ϵ Δ enu	Spin-boson model with parameters with scale factor `enu`
spectra	debye E_r ω_c T	Debye spectral density parameters and temperature
	ohmic E_r ω_c T	Ohmic spectral density parameters and temperature

Important

- All input parameters are specified in **atomic units (a.u.)**, except for **temperature**, which must be given in **Kelvin**.
- For **spin-boson parameters**, values are expressed in energy unit of cm^{-1} including temperature, and scaled by the factor `enu`.
- If `enu = 1`, the spin-boson parameters are interpreted directly as in **atomic units**.

4.2 Input File

A sample input file (param.in) for Spin-Boson model.

Listing 4.1: SAMPLE INPUT FILE (param.in)

```
#SHARP Pack INPUT PARAMETERS
#model(spinboson eps delta, enu)
model  spinboson  0.0  1.0  104.25

#bathspectrum (lambda wc temp)
spectra  debye  1  1  2

nbeads  8

approx  CA

nmode  matrix

iseed  12345

nParticle  100

#number steps
nsteps  41340

#trajectories
ntraj  1000

ncore  20

#timestep
tstep  1.0

#postion sampling
rsamp  gaussian
rmode  direct
rmap   no

#velocity sampling
vsamp  gaussian

#velocity reversal scheme for frustrated hop
vreverse  always

#decoherence
decoherence  none

rundtail  yes

finish
#end of Input File
```

4.3 Output Files

After a successful simulation run, SHARP Pack generates the following output files:

Table 4.3: Output file(s)

File(s)	Descriptions
`param.out`	Lists all the model parameters used in the simulation.
`dcoupling.out`	Detailed trajectory data including positions, momenta, energies, NACs, etc.
`hoppinghist.out`	Records surface-hopping events between different potential energy surfaces.
`pop_adiabat1.out`	Adiabatic population computed using Method I.
`pop_adiabat2.out`	Adiabatic population computed using Method II.
`pop_diabat1.out`	Diabatic population computed using Method I.
`pop_diabat2.out`	Diabatic population computed using Method II.
`pop_diabat3.out`	Diabatic population computed using Method III.
`pop_branch.out`	Branching probablity of **Tully model** if `accKval yes` .
`sampling_pos.out`	Initial sampling distribution of positions.
`sampling_vel.out`	Initial sampling distribution of velocities.
`sampling_dis.out`	Initial sampling Boltzmann distribution function of speeds.
`pimd_sample.out`	PIMD sampling output (generated only if PIMD sampling is enabled).
`energy_surface.out`	Diabatic and adiabatic energy surfaces, including NAC vectors.
`energy_dboc.out`	Adiabatic and DBOC energy surfaces (if dboc is requested)
`psi.out`	Wavefunctions, adiabatic coefficients, and PES.
`bathfrequency.out`	Spectral density information (specific to the spin-boson model).

Note

The output files `dcoupling.out`, `hoppinghist.out`, and `psi.out` are produced **only if** `rundtail` is `yes` in the input file.

4.4 Population Calculation

SHARP Pack runs simulations in an **adiabatic formalism**; hence, the adiabatic state populations are the direct results. But it also provides multiple approaches for computing adiabatic and diabatic populations.

4.4.1 Adiabatic Population

There are two methods:

Method-I: Direct counting trajectory

The percentage of trajectories propagating on each adiabatic PES represents the population of that state.

$$P_\alpha = \langle \delta_{\alpha,\lambda} \rangle$$

where λ is the adiabatic active surface.

output file name: `pop_adiabat1.out`

Method-II: Density matrix method

Populations are computed from the complex-valued electronic coefficients.

$$P_\alpha = \langle \rho_{\alpha\alpha} \rangle$$

where $\rho_{ij} = c_i\, c_j^*$ is the electronic density matrix.

output file name: `pop_adiabat2.out`

4.4.2 Diabatic Population

There are three methods, [21] using U_{ij} which are elements of the adiabatic-to-diabatic transformation matrix, and λ denotes the active surface.

Method-I: Projection Method

The diabatic population is determined by projecting the active adiabatic state onto the diabatic basis.

$$P_\alpha = \left\langle \sum_i |U_{\alpha i}\, \delta_{i,\lambda}|^2 \right\rangle,$$

output file name: `pop_diabat1.out`

Method-II: Electronic Wavefunction Method

Diabatic populations are calculated by transforming the electronic coefficients c_i in the adiabatic basis to the diabatic representation.

$$P_\alpha = \left\langle \sum_i |U_{\alpha i}\, c_i|^2 \right\rangle,$$

output file name: `pop_diabat2.out`

Method-III: Mixed quantum-classical density matrix

The diabatic populations are extracted from the electronic density matrix in the diabatic representation.

$$P_\alpha = \left\langle \sum_i |U_{\alpha i}|^2 \delta_{i,\lambda} + \sum_{i<j} 2\,\mathrm{Re}\left[U_{\alpha i}, \rho_{ij}, U_{\alpha j}^*\right] \right\rangle,$$

output file name: `pop_diabat3.out`

4.5 Running Simulation

To run **SHARP Pack**, a single input file `param.in` is required, which defines the system model parameters (see *Input Keywords* for details). Simulations are executed using the provided bash scripts located in the `utility/` directory.

Available job submission scripts:

- `job-script-local.sh` Run simulations on a **local machine**.
- `job-script-hpc.sh` Submit jobs on an **HPC cluster** using the **Slurm** workload manager.
- `job-script-branching.sh` Used **only** for computing branching probabilities on **Tully model** over multiple *k*-values on an HPC system.
- `job-script-avg.sh` Submit job that computes average of *output* of the completed parallel jobs simulation.

Each script automatically generates the appropriate job submission file (serial or parallel, depending on the `ncore` value in `param.in`) and submits the job.

Usage

Run any of the scripts as follows:

```
# Run on a local machine
$ sh job-script-local.sh

# Run on an HPC cluster (Slurm)
$ sh job-script-hpc.sh

# Compute branching probability on HPC
$ sh job-script-branching.sh
```

Example script

An example of `job_script_hpc.sh` is shown below which is designed for running simulations on an **HPC cluster** using the Slurm workload manager. It automatically detects the number of cores (`ncore`) requested in the `param.in` file and prepares the submission accordingly:

- If `ncore 1`; A **serial job** is created and submitted.
- If `ncore n` (`n > 1`); The script creates n **run* directories** and distributes the number of trajectories across them for **parallel simulation**. After all simulations are complete, the results are averaged to produce the final output.

Listing 4.2: `Bash script for job submission`

```
#!/bin/bash
#============================================================
#     JOB SCRIPT TO RUN SHARP PACK v2 ON HPC SERIAL/PARALLEL
#     WHILE RUNNING PARALLEL, BASED ON n-CPU, IT CREATES
#     n-DIRECTORIES AND RUN JOBS PARALLELLY, AND FINALLY
#     COMPUTES AVERAGE OF THE DISIRED POPULATATION (NEED TO
#     SPECIFY FOR INPUT, SEE EXAMPLE IN, WRITE_AVERAGE_INPUT)
#
#     authors    - D.K. Limbu & F.A. Shakib
#
#     Method Development and Materials Simulation Laboratory
#     New Jersey Institute of Technology
#
#     USAGE:: bash job_script_hpc.sh
#           - creates and run job(s)
#
#============================================================

# Path to your executable
#============================================================
root2bin=~/Softwares/SHARP_pack2/bin
exe=${root2bin}/sharp.x

#============================================================
```

(continues on next page)

(continued from previous page)

```
maxcore=1
ncore=$(awk '/ncore/{print $2}' param.in)
maxcore=$((ncore > maxcore ? ncore : maxcore))
echo 'ncore(s):' $maxcore

# ==== CONFIGURATION ====
#============================================================
root_dir="run"            # Base directory name
ncore=$maxcore            # Number of parallel jobs/directories
param_file="param.in"     # Parameter file to copy to each directory

# Determine padding width based on ncore
if [ "$ncore" -lt 10 ]; then
   pad=1
elif [ "$ncore" -lt 100 ]; then
   pad=2
else
   pad=3
fi

# ==== FUNCTION TO CREATE AVERAGE INPUT ====
write_average_input() {
  #fname :: File name of specific population data for averaging
  #df    :: No. of data column(s)(except first column)?

  fname="pop_diabat3.out"
  df=2

  echo "Calculating average results from $ncore parallel jobs."

  rm -f input

  echo ${root_dir} >> input
  echo ${fname} >> input
  echo ${ncore} >> input
  echo ${df} >> input
  echo ${pad} >> input

}

# ==== FUNCTION TO CREATE PARALLEL DIRECTORIES ====
paralleldir() {
  # Check if the first directory already exists
  first_dir=$(printf "${root_dir}%0${pad}g" 1)
  if [ -d "$first_dir" ]; then
    echo '***********************************'
    echo "$first_dir exists. Remove it first!"
    echo '***********************************'
    exit 1
  fi

  # Create directories and copy files
```

(continues on next page)

(continued from previous page)

```
  for i in $(seq -f "%0${pad}g" 1 $ncore); do
    dir="${root_dir}${i}"
    mkdir "$dir"
    cp param.in "$dir/param.in"
    echo "$i" > "$dir/icpu.in"
  done

  echo "Created $ncore directories: ${root_dir}$(printf "%0${pad}d" 1) to ${root_dir}$
↪{ncore}"
}

# ==== FUNCTION TO WRITE SLURM ARRAY SCRIPT ====
write_array_job() {

  cat <<EOF > submit_array.sh
#!/bin/bash
#SBATCH --job-name=array_run
#SBATCH --array=1-${ncore}
#SBATCH --output=${root_dir}%0${pad}a/myoutput_%A_%a.out
#SBATCH --error=${root_dir}%0${pad}a/jobError_%A_%a.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH -p RM-shared
#SBATCH --mem-per-cpu=2000M
#SBATCH --mail-type=end,fail
#SBATCH --time=01:00:00

cd ${root_dir}\$(printf "%0${pad}d" \${SLURM_ARRAY_TASK_ID})

 $exe

EOF

  chmod +x submit_array.sh
}

# ==== FUNCTION TO WRITE SERIAL SLURM  SCRIPT ====
write_serial_job() {

  cat <<EOF > submit_serial.sh
#!/bin/bash
#SBATCH --job-name=serial_run
#SBATCH --output=myoutput%j.out
#SBATCH --error=jobError.out
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH -p RM-shared
#SBATCH --mem-per-cpu=2000M
#SBATCH --mail-type=end,fail
#SBATCH --time=10:00:00
```

(continues on next page)

(continued from previous page)

```
 $exe

EOF

  chmod +x submit_serial.sh
}

# ==== FUNCTION TO WRITE ANALYSIS JOB SCRIPT ====
write_average_job() {
  cat <<EOF > average_job.sh
#!/bin/bash
#SBATCH --job-name=average
#SBATCH --output=myoutput%j.out
#SBATCH --error=jobError.out
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH -p RM-shared
#SBATCH --mem-per-cpu=2000M
#SBATCH --mail-type=end,fail
#SBATCH --time=00:10:00

  ${root2bin}/average.x

EOF

  chmod +x average_job.sh
}

#============================================================
## RUN SERIAL JOB ##
#============================================================
if [ $ncore -eq 1 ] ; then

  echo "Serial job running in a single core!!"

  write_serial_job

  sbatch submit_serial.sh

#============================================================
## RUN PARALLEL JOB(S) ON HPC ##
#============================================================
elif [ $ncore -gt 1 ] ; then
  #create parallel jobs

  echo "Parallel job running on " $ncore "core(s)."

  # ==== MAIN ====
  paralleldir
  write_array_job
  write_average_input
```

(continues on next page)

(continued from previous page)

```
  write_average_job

  # Submit array job
  array_jobid=$(sbatch --parsable submit_array.sh)

  echo "Submitting post-processing job dependent on array job ${array_jobid}..."

  # Submit average job to run only after array job completes
  sbatch --dependency=afterok:$array_jobid average_job.sh

  echo "All jobs submitted successfully!"

fi
```

Note

To compute the average results from parallel runs, must to specify `fname`, and `df` inside the **write_average_input()** fucntion in `job_script_hcp.sh` or `job_script_local.sh` as:

```
# ==== FUNCTION TO CREATE AVERAGE INPUT ====
write_average_input() {
#fname :: File name of specific population data for averaging
#df    :: No. of data column(s)(except first column)

fname="pop_diabat3.out"
df=2
.
.
```

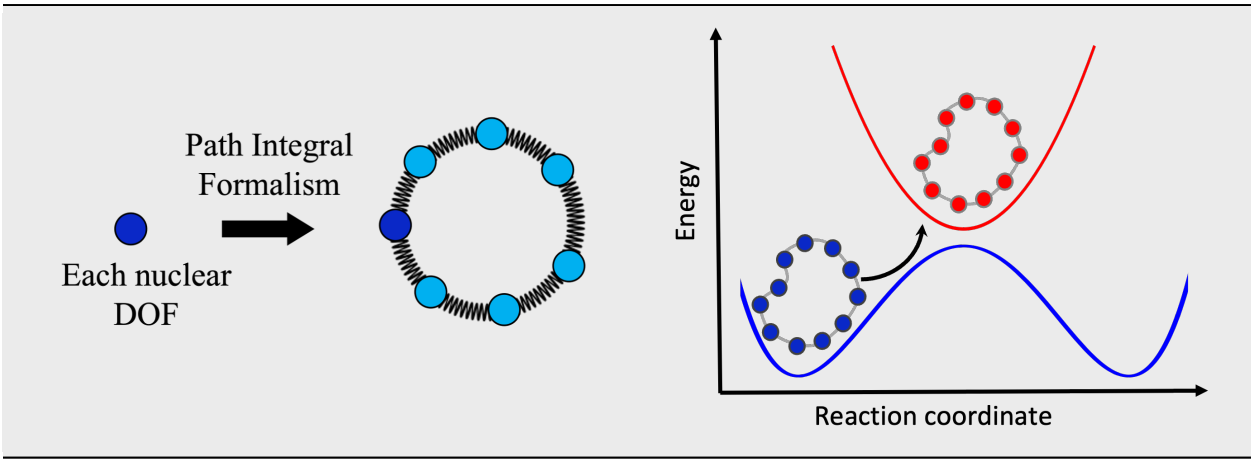
The final average output file is `fname_ave.out`. e.g. `pop_diabat3.out` ==> `pop_diabat3_ave.out`

CHAPTER

FIVE

RING POLYMER SURFACE HOPPING (RPSH) ALGORITHM

The RPSH method combines the nuclear path-integral representation of nuclei [5] with Tully's fewest-switches surface hopping [3] scheme.



For a system of N particles in the electronic ground state, the general Hamiltonian is:

$$\hat{H} = \sum_{I=1}^{N} \frac{\hat{\mathbf{P}}_I^2}{2M_I} + \hat{V}(\mathbf{R}), \tag{5.1}$$

where

- $\mathbf{R}$, $\mathbf{P}$ are nuclear position and momentum vectors,
- M_I is the mass of the I-th nuclear degree of freedom (DOF),
- $\hat{V}(\mathbf{R})$ is the potential energy surface.

The extended Hamiltonian H_n for a ring polymer of **n** beads with positions $\{\mathbf{R}_1, \ldots, \mathbf{R}_n\}$ and momenta $\{\mathbf{P}_1, \ldots, \mathbf{P}_n\}$ is:

$$\begin{aligned} H_n(\mathbf{R}, \mathbf{P}) = &\sum_{I=1}^{N} \sum_{k=1}^{n} \left[\frac{P_{I,k}^2}{2M_{I,k}} + \frac{1}{2} M_{I,k} \omega_n^2 \left(R_{I,k} - R_{I,k-1} \right)^2 \right] \\ &+ \sum_{k=1}^{n} V\left(R_{1,k}, \ldots, R_{N,k} \right), \end{aligned} \tag{5.2}$$

The surface hopping approach can use the extended Hamiltonian (Eq. (5.2)) to propagate the entire ring polymer on a single adiabatic surface $|\alpha; \mathbf{R}\rangle$, with non-adiabatic transitions between surfaces determined by the fewest-switches

ansatz. The Hamiltonian is modified by introducing state-dependent potentials:

$$V_\alpha(\mathbf{R}) = \langle \alpha; \mathbf{R} | \hat{V} | \alpha; \mathbf{R} \rangle,$$

leading to the **Ring Polymer Surface Hopping (RPSH)** algorithm. This requires numerical integration of the time-dependent Schrödinger equation (TDSE) along classical trajectories.

Two approximations for TDSE propagation are used in RPSH [6]:

1. **Bead Approximation (RPSH-BA)** The TDSE is averaged over beads:

$$i\hbar \dot{c}_\alpha(t) = \frac{1}{n}\sum_{k=1}^{n} V_\alpha(\mathbf{R}_k) c_\alpha(t) - i\hbar \sum_\gamma \frac{1}{n}\sum_{k=1}^{n} \dot{\mathbf{R}}_k \cdot \mathbf{d}_{\alpha\gamma}(\mathbf{R}_k) c_\gamma(t),$$

   where $\mathbf{d}_{\alpha\gamma}(\mathbf{R}_k)$ is the NACV for bead k.

2. **Centroid Approximation (RPSH-CA)** TDSE is evaluated at the ring-polymer centroid:

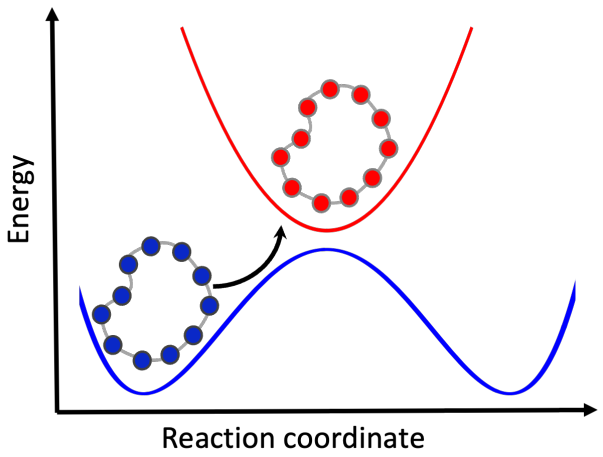
$$\bar{\mathbf{R}} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{R}_k, \quad \bar{\mathbf{P}} = \frac{1}{n}\sum_{k=1}^{n} \mathbf{P}_k,$$

$$i\hbar \dot{c}_\alpha(t) = V_\alpha(\bar{\mathbf{R}}) c_\alpha(t) - i\hbar \sum_\gamma \dot{\bar{\mathbf{R}}} \cdot \mathbf{d}_{\alpha\gamma}(\bar{\mathbf{R}}) c_\gamma(t).$$

The probability of a non-adiabatic transition at each timestep Δt is computed from the density matrix $\rho_{\alpha\gamma} = c_\alpha c_\gamma^*$ and NACVs within the framework of the FSSH algorithm,:cite:*Tully:1990* (needs to modify the corresponding relation appropriately for RPSH-BA or RPSH-CA).

$$g_{\alpha\to\gamma} = \max\left(0, \frac{-2\,\mathrm{Re}\left[\rho_{\gamma\alpha}^* \dot{\mathbf{R}} \cdot \mathbf{d}_{\gamma\alpha}\right] \delta t}{\rho_{\alpha\alpha}}\right).$$

Now attempt a hop between states. If a hop occurs, the ring polymer switches surfaces and nuclear velocities are rescaled to conserve total energy. If the hop is energetically forbidden, remain in the current state.



Continue nuclear and electronic propagation until the end of the simulation time.

CHAPTER

SIX

DIAGONAL BORN–OPPENHEIMER CORRECTION (DBOC)

When the nuclear kinetic energy operator

$$\hat{T}_{\mathrm{n}} = -\sum_{I} \frac{\hbar^2}{2M_I} \nabla^2_{\mathbf{R}_I}$$

acts on the adiabatic electronic wavefunctions both diagonal and off-diagonal non-adiabatic couplings arise. [16] The diagonal term behaves like a potential energy correction and modifies the adiabatic PESs, $V_\alpha(\mathbf{R})$, as

$$\tilde{V}_\alpha(\mathbf{R}) = V_\alpha(\mathbf{R}) + V^{(\alpha)}_{\mathrm{DBOC}}(\mathbf{R}).$$

Here, $\mathbf{R} \equiv \{\mathbf{R}_1, \mathbf{R}_2, \ldots, \mathbf{R}_n\}$ denotes the ring-polymer coordinates containing n beads. The correction term is

$$V^{(\alpha)}_{\mathrm{DBOC}}(\mathbf{R}) = \sum_{I=1}^{N} \sum_{k=1}^{n} \frac{\hbar^2}{2M_{I,k}} \sum_{\gamma \neq \alpha} \left| \mathbf{d}_{I,k,\alpha\gamma}(\mathbf{R}_k) \right|^2 .$$

Total force on bead k for adiabatic state α is

$$\mathbf{F}_{I,k} = -\nabla \tilde{V}_\alpha(\mathbf{R}_k) = -\nabla V_\alpha(\mathbf{R}_k) - \nabla V^{(\alpha)}_{\mathrm{DBOC}}(\mathbf{R}_k) = \mathbf{F}^{(0)}_{I,k} + \Delta\mathbf{F}_{I,k},$$

where the DBOC force contribution is

$$\Delta\mathbf{F}_{I,k} = \sum_{\alpha \neq \gamma} \frac{1}{M_{I,k}} \mathbf{d}_{I,k,\alpha\gamma}(\mathbf{R}_k) \cdot \nabla \mathbf{d}_{I,k,\alpha\gamma}(\mathbf{R}_k).$$

The non-adiabatic coupling vectors (NACVs) are evaluated at each bead k:

$$\mathbf{d}_{I,k,\alpha\gamma} = \frac{\langle \psi_{k,\alpha} | \nabla_{R_{I,k}} H(\mathbf{R}_k) | \psi_{k,\gamma} \rangle}{E_\gamma(\mathbf{R}_k) - E_\alpha(\mathbf{R}_k)}.$$

DBOC-corrected forces are computed numerically via finite differences:

$$\nabla \mathbf{d}_{I,k,\alpha\gamma} = \frac{\mathbf{d}_{I,k,\alpha\gamma}(\mathbf{R}_k + \delta) - \mathbf{d}_{I,k,\alpha\gamma}(\mathbf{R}_k - \delta)}{2\delta}.$$

> **Warning**
>
> The displacement parameter δ must be chosen to ensure numerical stability and convergence of the DBOC forces during dynamics simulations.

CHAPTER

SEVEN

DECOHERENCE

Energy-based decoherence correction [22, 23] applies exponential damping to inactive adiabatic states, while renormalizing the active state to preserve the total wavefunction norm ($\sum |c_i|^2 = 1$).

$$c_i \to c_i \exp\left(-\frac{\Delta t}{\tau_{ia}}\right), \quad \forall\, i \neq a,$$

$$c_a = c_a \left[\frac{1 - \sum_{i \neq a} |c_i|^2}{|c_a|^2}\right]^{1/2},$$

with

$$\tau_{ia} = \frac{\hbar}{|E_i - E_a|}\left(C + \frac{E_0}{T_a}\right),$$

where

- E_i is the energy of inactive state i,
- E_a is the energy of active state a,
- T_a is the kinetic energy on the active surface,
- C and E_0 are empirical constants.

CHAPTER

EIGHT

TUTORIALS

This section provides step-by-step tutorials for running **SHARP Pack**. Each tutorial includes the required `param.in` input template, execution commands, and simple plotting instructions.

Note

- All simulations assume SHARP Pack is compiled (see *Installation*) and the executable is available as `sharp.x`.
- By default, the program reads input parameters from `param.in` in the working directory.
- Units: All input parameters are in **atomic units**, except temperature in Kelvin. Spin-boson model parameters are given in cm^{-1} when `enu` $\neq 1$.

List of tutorials:

- *Tully Model-I (FSSH)*
- *Tully Model-II (RPSH)*
- *Tully Model-III (Branching Probability)*
- *Linear Chain Model*
- *Spin-Boson Model (Debye Spectral Density)*

Note

All the example tutorials can be found in the `example/` directory.

Note

Job submission bash scripts are located in the `utility/` directory for the HPC Slurm system, as well as on the local machine.

CHAPTER

NINE

TULLY MODEL-I (FSSH)

This tutorial demonstrates how to run **SHARP Pack** for the `Tully Model-I` in **serial mode**. Follow the steps below to perform the simulation.

9.1 Prepare `param.in`

First, prepare the input parameter file `param.in` with the required keywords. For example, at a particular `pinit` value and `nbeads 1` (`FSSH` method):

```
#tully model-I input parameters
model         tully1
nParticle     1
nbeads        1
iseed         12345
tstep         1.0
nsteps        3000
ntraj         1000
ncore         1
pinit         20.0
rsamp         gaussian
vsamp         fixed
vreverse      never
rundtail      yes
iprint        10
finish
```

9.2 Run Simulation

Run the simulation using one of the following methods:

1. Direct execution:

```
$ ./sharp.x
```

2. Using a job submission script from the `utility` directory, depending on the computing system:

```
# for running on a local machine
$ sh job-script-local.sh

#for submitting jobs on an HPC cluster (Slurm)
$ sh job-script-hpc.sh
```

9.3 Plot Results with Gnuplot

Plot Energy

- Create a **gnuplot** script file `plot_energy.gnu`:

```
#!/usr/bin/gnuplot
set key right top

set term postscript eps colour enhanced 'Helvetica' 24 size 4,4.5
set border lw 2.5
set tics scale 2

set output 'fig-energy.pdf'
NOXTICS = "set xtics (''-10,''-5,''0,''5,''10); \
      unset xlabel"
XTICS="set xtics -10,5,10;\
   set xlabel '{/Helvetica=22 R (a.u.)}'"

set key  samplen 1.0 spacing 1.3 font "Helvetica, 15"
set multiplot layout 2,1

@NOXTICS
set ylabel "{/Helvetica=22 H_{dia} (a.u.)}"
set yr[-.015:0.02]
set ytics 0.01
plot 'energy_surface.out' u 1:2 w l lw 5 t'V_{11}',\
     'energy_surface.out' u 1:5 w l lw 5 t'V_{22}',\
     'energy_surface.out' u 1:3 w l lw 5 dt 3 t'V_{12}'

@XTICS
set ylabel "{/Helvetica=22 H_{adia} (a.u.)}"
set yr[-.015:0.02]
set ytics 0.01

plot 'energy_surface.out' u 1:6 w l lw 5 t'{/Symbol e}_{1}',\
     'energy_surface.out' u 1:7 w l lw 5 t'{/Symbol e}_{2}',\
     'energy_surface.out' u 1:(colum(10)/100) w l lw 5 dt 3 t'd_{12}/100'
```

- Run the **gnuplot** script to generate the figure as:

```
$ gnuplot plot_energy.gnu
```

Plot Populations

- Similarly, create **gnuplot** script `plot_pop.gnu` to plot `adiabatic` or `diabatic` populations:

```
#!/usr/bin/gnuplot

set xlabel "Time (a.u.)"
set ylabel "Population"
set title "Tully1 Diabatic Populations"
plot "pop_diabat3.out" u 1:2 w l lw 2 title "Diabatic State 1", \
     "pop_diabat3.out" u 1:3 w l lw 2 title "Diabatic State 2"
```

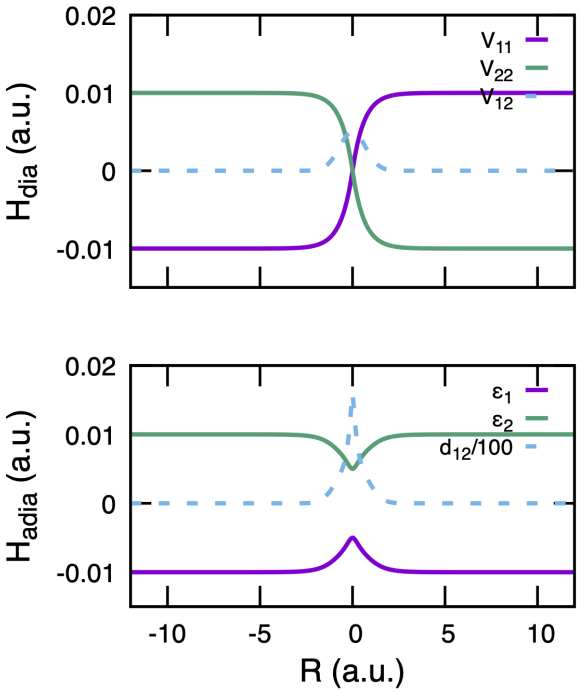


Fig. 9.1: Diabatic and adiabatic energy surfaces of Tully1 model.

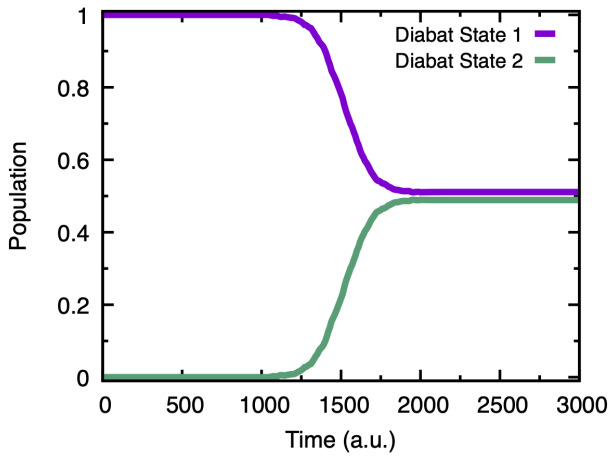


Fig. 9.2: Diabatic population of Tully1 model.

CHAPTER

TEN

TULLY MODEL-II (RPSH)

This tutorial demonstrates how to run RPSH simulation in **parallel mode**. The number of parallel runs is controlled by the value of `ncore n` specified in the `param.in` file.

Follow the steps below to perform the simulation:

10.1 Prepare `param.in`

Set `ncore n` (where `4` is the number of desired parallel runs).

```
#tully model-II input parameters
model        tully2
nParticle    1
nbeads       4
approx       CA
nmode        matrix
iseed        12345
ncore        4
tstep        1.0
nsteps       3000
ntraj        1000
pinit        15.0
rsamp        gaussian
vsamp        fixed
rundtail     Yes
iprint       10
finish
```

10.2 Run Simulation

Run the simulation using the provided job script in `utility/` directory (e.g., `job-script-hpc.sh` for an HPC cluster, or `job-script-local.sh` for local execution).

Based on `ncore 4`, the job script will:

- Create **4** `run*` directories.
- Distribute the trajectories evenly across the runs.
- Execute the simulations in parallel.
- After all parallel simulations are complete, the script automatically averages the results from the **4** `run*` and produces the final population output.

Note

To get average result from parallel runs, must to specify `fname`, and `df` inside the **write_average_input()** fucntion in `job_script_hcp.sh` or `job_script_local.sh` as:

```
# ==== FUNCTION TO CREATE AVERAGE INPUT ====
write_average_input() {
#fname :: File name of specific population data for averaging
#df    :: No. of data column(s)(except first column)

fname="pop_diabat3.out"
df=2
.
.
```

Note

To compute a standalone average of a specific output file after completing the parallel simulation, one can use `job_script_avg.sh` script provided in `utility/` directory.

Warning

Pay attention to modifying the variables `fname` (output filename) and `df` (number of data columns, excluding the first column) inside the `write_average_input()` function of `job_script_avg.sh`.

```
.
.
# ==== FUNCTION TO CREATE AVERAGE INPUT ====
write_average_input() {
#fname :: File name of specific population data for averaging
#df    :: No. of data column(s)(except first column)

fname="pop_diabat2.out"
df=4
.
.
```

10.3 Plot Results

Finally, use any visualization tool (e.g., **gnuplot** or **matplotlib**) to plot the population dynamics from the output files and analyze the results.

```
#!/usr/bin/gnuplot

set xlabel "Time (a.u.)"
set ylabel "Population"
set title "Tully2 Diabatic Populations"
plot "pop_diabat3_ave.out" u 1:2 w l lw 2 title "Diabatic State 1", \
     "pop_diabat3_ave.out" u 1:3 w l lw 2 title "Diabatic State 2"
```

CHAPTER

ELEVEN

TULLY MODEL-III (BRANCHING PROBABILITY)

This tutorial demonstrates how to calculate the **branching probability** in the **Tully model III** with the RPSH method for a range of *k-values* (initial momentum).

Follow the steps below to perform the simulation.

11.1 Prepare Base Input File

Create a base input parameter file `base_param.in` that specifies the system model parameters. This file serves as the template for all `k` values.

```
#tully model-III input parameter
model     tully3
ncore     1
nbeads    8
.
.
pinit     pval
acckval   yes
.
.
.
finish
```

> **Note**
>
> Ensure that `acckval yes` in `base_param.in` to obtain the branching probability results at each `k` value.

11.2 Run Simulation

Run the provided branching job script `job_script_branching.sh`.

```
$ sh job-script-branching.sh
```

This script will:

- Create the required number of **run*** directories based on the `k` values.
- Submit and run the simulations inHPC cluster.

- After all simulations are completed, the script automatically collects the outputs from all directories and combines them.

The final result is `pot_branch_all.out` for the specified `k` values.

> **Note**
>
> Make sure to modify the range and increment of `k` in `job_script_branching.sh` to match the desired initial momentum values, e.g.:

```
kmin=5    #minimum k-value
kmax=20   #maximum k-value
dk=0.5    #increment of k
```

11.3 Plot Branching Probability

Use any visualization tool to plot the **branching probability** results.

The output file `pop_branch_all.out` has the following structure:

- **First column**: `k` values (initial momentum)
- **Second column**: Transmission probability in state 1
- **Third column**: Transmission probability in state 2
- **Fourth column**: Reflection probability in state 1
- **Fifth column**: Reflection probability in state 2

Gnuplot script `plot-branch.gnu` looks like:

```
#!/usr/bin/gnuplot
set terminal pdf size 4in,3in enhanced color font 'Helvetica,16'
set border lw 2.5
set tics scale 1.2
set output 'fig-tully3-branch.pdf'
set key  samplen 1.0 spacing 1.3 font "Helvetica, 14"
set multiplot layout 1,1
set xlabel 'k (a.u.)'
set ylabel 'P(k)'
set mytics 2
set ytics 0.2
set xtics 5
set yr[0:1]
ps=0.7
plot 'pop_branch_all.out' u 1:2 w lp lc 2 lw 2 dt 2 pt 7 ps ps t'T1',\
     'pop_branch_all.out' u 1:3 w lp lc 6 lw 2 dt 2 pt 7 ps ps t'T2',\
     'pop_branch_all.out' u 1:4 w lp lc 7 lw 2 dt 2 pt 7 ps ps t'R1',\
     'pop_branch_all.out' u 1:5 w lp lc 4 lw 2 dt 2 pt 7 ps ps t'R2'
```

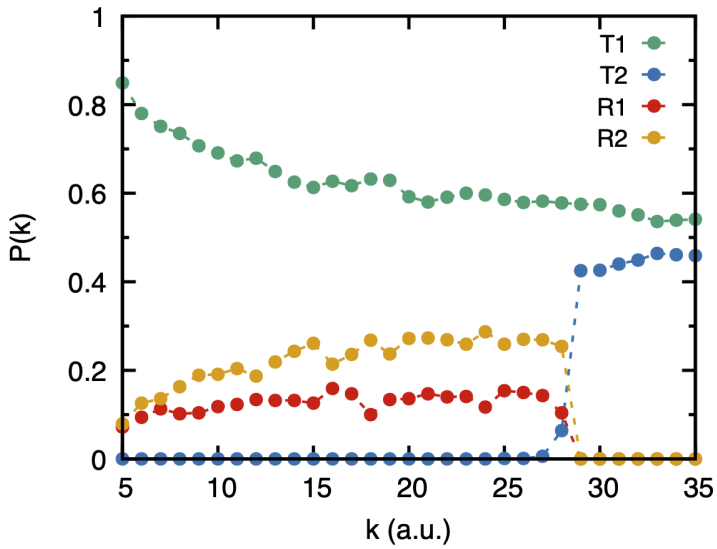


Fig. 11.1: Branching probabilities as a function of the initial **k** for Tully Model III using RPSH method.

CHAPTER

TWELVE

LINEAR CHAIN MODEL

This tutorial demonstrates running the **Linear Chain Model** at a temperature of **500 K** for a simulation time of **50 ps** and compare the simulation result obtained from SHARP Pack with the **exact analytical result**.

Follow the steps below:

12.1 Prepare `param.in`

Prepare input file `param.in` with the model and simulation parameters setting `temperatrue 500`.

```
#linear chain model input parameter
model        db2lchain
nParticle    20
nbeads       1
iseed        12345
ncore        1
tstep        10.0
nsteps       206700
ntraj        200
temperature  500.0
rsamp        gaussian
vsamp        gaussian
iprint       100
finish
```

12.2 Run Simulation

Run the simulation using one of the following methods:

```
# direct execution
$ ./sharp.x

# for running on a local machine
$ sh job-script-local.sh

#for submitting jobs on an HPC cluster (Slurm)
$ sh job-script-hpc.sh
```

12.3 Plot Result

Use **gnuplot** script `plot-pop.gnu` to plot the convergence of population to exact Boltzmann distribution.

```
#!/usr/bin/gnuplot
set encoding iso_8859_1
set key right center
set terminal pdf size 4in,2.8in enhanced color font 'Helvetica,16'
set border lw 2.5
set tics scale 1.2
set ylabel "{/Helvetica=22 Population}"
set xlabel "{/Helvetica=22 time (ps)}"
set mytics 2
set mxtics 2
set grid xtics ytics

e1=0;e2=8.0;
temp=500.0  #in Kevlin
kbT=0.0083*temp

#exact Boltzmann distribution
p1(x)=exp(-e1/kbT)/(exp(-e1/kbT)+exp(-e2/kbT));
p2(x)=exp(-e2/kbT)/(exp(-e1/kbT)+exp(-e2/kbT));

outfile = 'fig-population.pdf'
set output outfile
set key opaque samplen 1.0 spacing 1.3 font "Helvetica, 14"
plot 'pop_adiabat1.out' u (column(1)/41340):2 w l lc 3 t'FSSH: P_1',\
     'pop_adiabat1.out' u (column(1))/41340):3 w l lc 4 t'FSSH: P_2',\
      p1(x) w l lc 6 lw 5 t'Exact: P_1',\
      p2(x) w l lc 7 lw 5 t'Exact: P_2'
```

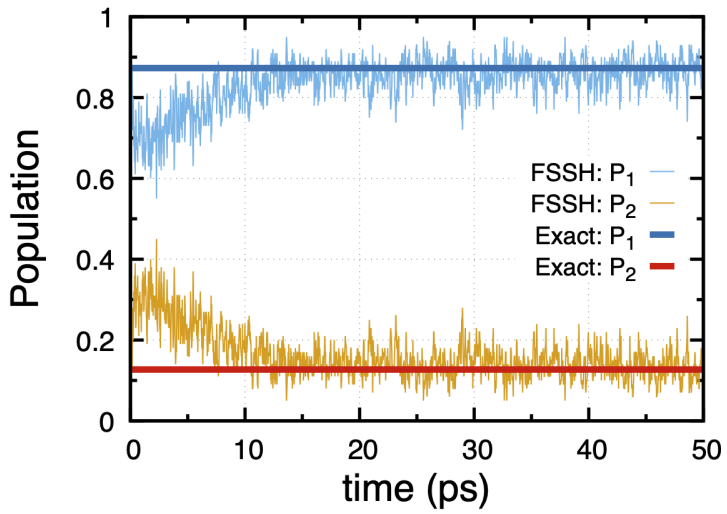


Fig. 12.1: Population convergence of linear chain model to exact Boltzmann distribution at 500 K by FSSH method.

CHAPTER

THIRTEEN

SPIN-BOSON MODEL (DEBYE SPECTRAL DENSITY)

This section demonstrates the simulation of the **spin-boson model**.

The **model parameters** must be specified in the following order: **model** spinboson ϵ Δ enu

The **bath parameters** are specified as: **spectra** **bath-type** E_r ω_c T

> **Note**
>
> - All **spin-boson** parameters are given in energy unit of cm^{-1} and scale by the factor `enu`.
> - If `enu = 1`, then all parameters are expressed in **atomic units**.

13.1 Prepare `param.in`

Prepare input file `param.in` for spin-boson model with appropriate model parameters.

```
#SHARP Pack INPUT PARAMETERS
model      spinboson 0.0 1.0 104.25
spectra    debye 1.0 1.0 2.0
nParticle  100
nbeads     4
approx     CA
nmode      matrix
nsteps     20670
ntraj      1000
ncore      1
tstep      1.0
rsamp      gaussian
rmode      norm
rmap       no
vsamp      gaussian
VReverse   always
Vrescale   BL
rundtail   Yes
iprint     10
finish
```

13.2 Run Simulation

Run the simulation using one of the following methods:

```
# direct execution
$ ./sharp.x

# for running on a local machine
$ sh job-script-local.sh

#for submitting jobs on an HPC cluster (Slurm)
$ sh job-script-hpc.sh
```

13.3 Plot Result

Use **gnuplot** script `plot-pop.gnu` to plot population dynamics.

```
#!/usr/bin/gnuplot
set terminal pdf size 4in,3in enhanced color font 'Helvetica,16'
set border lw 2.5
set tics scale 1.2
set multiplot layout 1,1
set xlabel 'Time (a.u.)'
set ylabel 'P_1'
set yr[0:1]
set xr[0:0.5]

heom = 'heom-result/exact-qutip-lambda1-d1-w1-t300.dat'
set output 'fig-spinboson-pop.pdf'
set key  samplen 1.0 spacing 1.3 font "Helvetica, 14"
plot heom every 25 u 1:2 w p lc 0 pt 7 ps 0.6 t'Exact',\
     'pop_diabat3.out' u (column(1)/41340):2 w l ls 6 lw 5 t'RPSH'
```

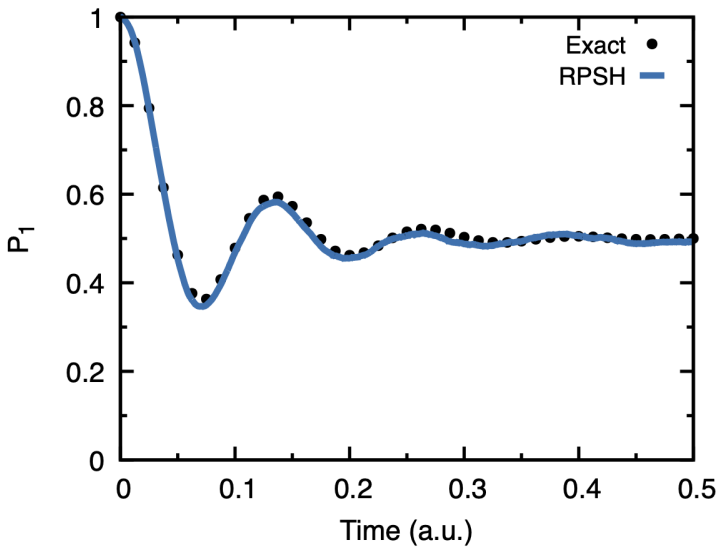


Fig. 13.1: Population dynamics of spin-boson model at temperature 300 K and reorganization energy, $E_r = 0.02$.

CHAPTER

FOURTEEN

CHANGE LOG

Notable changes to **SHAPRP Pack** will be documented here.

14.1 Version 2.0 (2025-09-05)

14.1.1 New Features

- Extended method to bead-approximation ring polymer surface hopping (RPSH-BA).
- Diagonal Born-Oppenheimer Correction (DBOC) implementation.
- Option for decoherence correction.
- Random seed handling updated for reproducibility.
- Added PILE thermostat for PIMD sampling simulations.
- New PIMD and runtime outputs: *bead*- and *centroid*-level system temperatures.
- New option for constrained centroid to dividing surface (`R=0`) for initial sampling, shifting the whole ring polymer accordingly.
- Optput histogram distribution of initial sampling positions and momenta.
- Extented model system to *spin-boson* and *dboc1* models.
- Added *Debye/Ohmic* spectral density options for *spin-boson model*.

14.1.2 Bug Fixes

- Initial bead position sampling from de Broglie length for SB model
- Initial force calculation at t=0 carried out before running dynamics.

14.1.3 Documentation

- Online documentation lauch (docs in Sphinx via Readthedocs)

14.2 Version 1.0 (2023-11-28)

14.2.1 New Features

- SHARP Pack software introduce based on Ring Polymer Surface Hopping method.

BIBLIOGRAPHY

[1] M. H. V. Huynh and T. J. Meyer. Proton-coupled electron transfer. *Chem. Rev.*, 107:5004–5064, 2007. doi:10.1021/cr0500030.

[2] S. Hammes-Schiffer and A. A. Stuchebrukhov. Theory of coupled electron and proton transfer reactions. *Chem. Rev.*, 110:6939–6960, 2010. doi:10.1021/cr1001436.

[3] John C. Tully. Molecular dynamics with electronic transitions. *J. Chem. Phys.*, 93:1061–1071, 1990. doi:10.1063/1.459170.

[4] J. Cao and G. A. Voth. The formulation of quantum statistical mechanics based on the Feynman path centroid density. IV. Algorithms for centroid molecular dynamics. *J. Chem. Phys.*, 101:6168–6183, 1994. doi:10.1063/1.467176.

[5] Ian R. Craig and David E. Manolopoulos. Quantum Statistics and Classical Mechanics: Real Time Correlation Functions from Ring Polymer Molecular Dynamics. *J. Chem. Phys.*, 121:3368–3373, 2004. doi:10.1063/1.1777575.

[6] Philip Shushkov, Richard Li, and John C. Tully. Ring polymer molecular dynamics with surface hopping. *J. Chem. Phys.*, 137:22A549, 2012. doi:10.1063/1.4766449.

[7] Farnaz A. Shakib and Pengfei Huo. Ring Polymer Surface Hopping: Incorporating Nuclear Quantum Effects into Nonadiabatic Molecular Dynamics Simulations. *J. Phys. Chem. Lett.*, 8:3073–3080, 2017. doi:10.1021/acs.jpclett.7b01343.

[8] Dil K. Limbu and Farnaz A. Shakib. SHARP pack: a modular software for incorporating nuclear quantum effects into non-adiabatic quantum dynamic simulations in condensed phases. *Software Impacts*, 19:100604, 2024. doi:10.1016/j.simpa.2023.100604.

[9] Eduardo A. Coronado, Jianhua Xing, and William H. Miller. Ultrafast non-adiabatic dynamics of systems with multiple surface crossings: a test of the meyer–miller hamiltonian with semiclassical initial value representation methods. *Chem. Phys. Lett.*, 349:521–529, 2001. doi:10.1016/S0009-2614(01)01242-8.

[10] Dil K. Limbu and Farnaz A. Shakib. Real-Time Dynamics and Detailed Balance in Ring Polymer Surface Hopping: The Impact of Frustrated Hops. *J. Phys. Chem. Lett.*, 14:8658–8666, 2023. doi:10.1021/acs.jpclett.3c02085.

[11] Priya V. Parandekar and John C. Tully. Mixed quantum-classical equilibrium. *J. Chem. Phys.*, 122:094102, 2005. doi:10.1063/1.1856460.

[12] Andrew E. Sifain, Linjun Wang, and Oleg V. Prezhdo. Communication: proper treatment of classically forbidden electronic transitions significantly improves detailed balance in surface hopping. *J. Chem. Phys.*, 144:211102, 2016. doi:10.1063/1.4953444.

[13] Linjun Wang, Dhara Trivedi, and Oleg V. Prezhdo. Global flux surface hopping approach for mixed quantum-classical dynamics. *J. Chem. Theory Comput.*, 10:3598–3605, 2014. doi:10.1021/ct5003835.

[14] A. J. Leggett, S. Chakravarty, A. T. Dorsey, Matthew P. A. Fisher, Anupam Garg, and W. Zwerger. Dynamics of the dissipative two-state system. *Rev. Mod. Phys.*, 59:1–85, 1987. doi:10.1103/RevModPhys.59.1.

[15] Neil Shenvi. Phase-space surface hopping: nonadiabatic dynamics in a superadiabatic basis. *J. Chem. Phys.*, 130:124117, 2009. doi:10.1063/1.3098321.

[16] Rami Gherib, Liyuan Ye, Ilya G. Ryabinkin, and Artur F. Izmaylov. On the inclusion of the diagonal born-oppenheimer correction in surface hopping methods. *J. Chem. Phys.*, 144:154103, 2016. doi:10.1063/1.4945817.

[17] Haobin Wang, Xueyu Song, David Chandler, and William H. Miller. Semiclassical study of electronically nonadiabatic dynamics in the condensed-phase: spin-boson problem with debye spectral density. *J. Chem. Phys.*, 110:4828–4840, 1999. doi:10.1063/1.478388.

[18] Roel Templaar and David R. Reichman. Generalization of fewest-switches surface hopping for coherence. *J. Chem. Phys.*, 148:102309, 2018. doi:10.1063/1.5000843.

[19] Haobin Wang, Michael Thoss, and William H. Miller. Systematic convergence in the dynamical hybrid approach for complex systems: a numerically exact methodology. *J. Chem. Phys.*, 115:2979–2990, 2001. doi:10.1063/1.1385561.

[20] Najeh Rekik, Chang-Yu Hsieh, Holly Freedman, and Gabriel Hanna. A mixed quantum-classical liouville study of the population dynamics in a model photo-induced condensed phase electron transfer reaction. *J. Chem. Phys.*, 138:144106, 2013. doi:10.1063/1.4799272.

[21] Brian R. Landry, Martin J. Falk, and Joseph E. Subotnik. Communication: the correct interpretation of surface hopping trajectories: how to calculate electronic properties. *J. Chem. Phys.*, 139:211101, 2013. doi:10.1063/1.4837795.

[22] Chaoyuan Zhu, Shikha Nangia, Ahren W. Jasper, and Donald G. Truhlar. Coherent switching with decay of mixing: an improved treatment of electronic coherence for non-born–oppenheimer trajectories. *J. Chem. Phys.*, 121:7658–7670, 2004. doi:10.1063/1.1793991.

[23] Giovanni Granucci and Maurizio Persico. Critical appraisal of the fewest switches algorithm for surface hopping. *J. Chem. Phys.*, 126:134114, 2007. doi:10.1063/1.2715585.