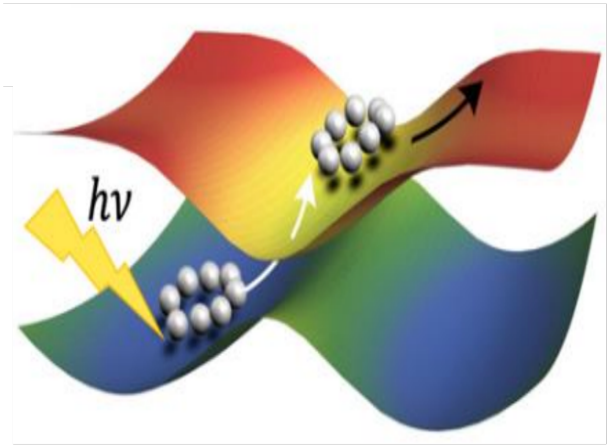

Surface Hopping And Ring Polymer (SHARP Pack) Manual

Version 1.0

Dil K. Limbu

Farnaz A. Shakib

November 2023

Contents

1 Introduction

1.1 SHARP pack

SHARP pack is a molecular dynamics (MD) simulation package that has been developed to utilize the potentials of combined path integral and surface hopping formalisms [1] for incorporating nuclear quantum effects (NQEs) into non-adiabatic dynamics simulations in condensed phases. It is a modular-based package written in Fortran 90 and distributed under the MIT license.

SHARP pack version 1.0 is produced in the "Method Development and Materials Simulation Lab" at the New Jersey Institute of Technology by Dil K. Limbu and Farnaz A. Shakib. It will be constantly evolved to provide (i) a methodology assessment and model testing platform and (ii) a large-scale MD simulation package subject to both electronic and nuclear quantum effects.

We request future users to respect the copyright of SHARP pack and do not alter any authorship or copyright notice within.

1.2 Functionality

The SHARP pack version 1.0 handles:

(a) Fewest-switches surface-hopping (FSSH) simulations, Section 2.

(b) Ring-polymer surface-hopping (RPSH) simulations, Section 2.

(c) A variety of model Hamiltonians, Section 3.

(d) A variety of initial conditions and control parameters, Section 5.

(e) Different schemes for velocity rescaling in SH algorithm, Section 5.

(f) Velocity-verlet as well as Langevine dynamics for MD propagation, Section 2.

(g) A variety of output data, Section 6.

(h) Serial and parallel simulations.

2 Methodology

2.1 RPSH and FSSH

In RPSH [1–3], nuclear quantum effects (NQEs) such as proton tunneling and zero-point energy are introduced into the non-adiabatic quantum dynamics via the extended phase-space

of a classical ring polymer. Essentially, every nuclear degree of freedom (DOFs) is represented by a ring polymer comprising n copies, known as beads, of the same nuclear DOF connected by harmonic springs. The corresponding extended Hamiltonian is described as:

$$H_n = \sum_{i=1}^{n} \left[\frac{\mathbf{P}_i^2}{2M} + \frac{M\omega_n^2}{2}(\mathbf{R}_i - \mathbf{R}_{i-1})^2 + V_\alpha(\mathbf{R}_i) \right].$$

Here, $\mathbf{P}_i$ and $\mathbf{R}_i$ represent the momentum and position of each bead of the ring polymer which moves on a single adiabatic surface $|\alpha; \mathbf{R}_i\rangle$ corresponding to the potential energy $V_\alpha(\mathbf{R}_i) = \langle \alpha; \mathbf{R}_i | \hat{V} | \alpha; \mathbf{R}_i \rangle$. At every time step of the nuclear dynamics, the position and momentum of the centroid of the ring polymer are calculated. According to the surface hopping algorithm, the time-dependent Shrödinger's equation is numerically integrated along the motion of the centroid to confer the electronic coefficients c_α associated with each adiabatic surface. The probability of the non-adiabatic transition, i.e., switching between surfaces during each time step, is defined based on these electronic amplitudes. If a transition occurs, the entire ring polymer hops to the next adiabatic surface while the velocity of each bead in the ring polymer is re-scaled to conserve the total energy of the quantum plus classical subsystems.

Since RPSH follows the same propagation and transition ansatz of the original FSSH [4], albeit in the extended phase-space, in the limit of one bead, it will go back to the classical FSSH without any NQEs. Hence, by choice of 1 or more than one bead in the input file, see Section 5, the SHARP pack user can easily switch between FSSH and RPSH to assess their applicability in different systems and regimes.

2.2 Propagation

SHARP pack version 1.0 offers two main integration algorithms for different model Hamiltonians, velocity-Verlet (VV) [5], and Langevine [6] dynamics. The VV algorithm is both time-reversible and simple, proceeding in two stages. In the first, assuming initial time $t_0 = 0$, the momenta and positions of all beads of all ring polymers are evolved at half and full time-steps, respectively:

$$P(\Delta t/2) \leftarrow P(0) + \frac{\Delta t}{2} F(R(0))$$

$$R(\Delta t) \leftarrow R(0) + \Delta t P(\Delta t/2)$$

In the second stage, using the new positions, the new forces $F(R(\Delta t))$ are calculated, and then the momenta are updated to the full time-step:

$$P(\Delta t) \leftarrow P(\Delta t/2) + \frac{\Delta t}{2} F(R(\Delta t)).$$

Langevine dynamics follow similar ansatz, except the common force ($F = -\partial V/\partial R$) is complemented by a friction force and a random force representing the Brownian motion.

3 Models

A few model Hamiltonians are provided in version 1.0 of SHARP pack, as listed in Table 1. More information about these models can be found in the provided references.

Table 1: Model Hamiltonians implemented in the SHARP pack version 1.0

Models	Descriptions
tully1	Tully's single avoided crossing [4]
tully2	Tully's dual avoided crossing [4]
tully3	Tully's extended coupling with reflection [4]
morse1	Morse Model - I, photo-dissociation multiple avoided crossings [7]
morse2	Morse Model - II, photo-dissociation multiple avoided crossings [7]
morse3	Morse Model - III, photo-dissociation multiple avoided crossings [7]
db2lchain	2-state model coupled to a linear chain of particles [8, 9]
db3lchain	3-state super-exchange model to a linear chain of particles [9]

4 Compiling the code

The source code is compiled using a Makefile that is provided with the code in the source directory. The SHARP pack is tested under a Linux environment with an Intel Fortran compiler. The BLAS, LAPACK, and FFTW3 libraries are required to compile and execute the software following these steps:

- Download the SHARP pack software package from the official website or repository.

- Extract the downloaded file to a desired location on your computer.

- Install the necessary dependencies required by SHARP pack.

-Open the terminal and navigate to the directory where you have saved the SHARP pack files.

```
$ cd source/
```

- Type “make” and press Enter. This will run the Makefile and compile the SHARP pack software.

```
$ make
```

Once the compilation process is complete, the SHARP pack software will be ready to use with **sharp.x** executable in the bin directory.

5 Input Files

5.1 Keywords and descriptions

This section presents the format and all input keywords for the main SHARP pack input file called **param.in**. In Table 2, all input keywords and short descriptions for the SHARP pack input file are listed with necessary explanations followed.

Table 2: Input Keywords

Keywords	Arguments	Descriptions
model	***s***	Model name string from Table 1
nparticle	***n***	Number of simulating particle(s)
nbeads	***n***	Number of bead(s)
nsteps	***n***	Number of simulation steps
ntraj	***n***	Number of trajectories
nstates	***n***	Number of electronic states; Default based on Model
ncpu	***n***	Number of CPU(s) (1-Serial), (n-Parallel)
timestep	***f***	Simulation time-step (a.u.)
temp	***f***	Temperature (K)
kinitial	***f***	Initial momentum (a.u.)

Continued on next page

Table 2: Input Keywords (Continued)

Keywords	Arguments	Descriptions
beadpos	**'same'**	Assigning same position for bead(s) of each particle
	'gaussian'	Initial (beads) position from Gaussian distribution
	'wigner'	Initial (beads) position from Wigner distribution
vinitial	**'fixed'**	Deterministic initial momentum
	'gaussian'	Initial momentum from Gaussian distribution
	'wigner'	Initial momentum form Wigner distribution
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
vrescale	**'centroid'**	Centroid-approximation level of velocity rescaling of ring-polymer for energy conservation
	'bead'	Bead-approximation level of velocity rescaling of ring-polymer for energy conservation
usrkval	**'yes'**	If TRUE, read initial momentum (k) value and overwrite 'param.in' k-value
	'no'	Read and use initial k-value from 'param.in'
acckval	**'yes'**	If TRUE, writes accumulated result with k-value; useful for branching probability calculation(s)
	'no'	No accumulated result(s)
rundtail	**'yes'**	Print run-time details of the simulation
	'no'	No details of simulation
iprint	***n***	Print data every n timesteps
finish		End of the input file

$n-$integer, $f-$real number, $s-$strings, *default

5.2 Further explanations

Here, we provide more details for some of the keywords in Table 2.

beadpos

The user has three different options for initializing the positions of every beads of the ring polymer. While "same" is self-explanatory, "gaussian" refers to a Gaussian wavepacket in the form of

$$G(R) = \left(\frac{2\alpha}{\pi}\right)^{\frac{1}{4}} \exp(-\alpha(R - R_0)^2 + ik(R - R_0)) \tag{1}$$

which is manifested in a Gaussian distribution of the nuclear position around R_0 and the width of $\sigma_R = 1/\sqrt{2\alpha}$. On the other hand, "wigner" refers to the Wigner distribution corresponding to the same wavepacket. Note that Wigner transformation gives a wider distribution than the Gaussian.

vinitial

Follows similar explanations as in **beadpos**.

kinitial vs. **vinitial**

In RPSH, each nuclear DOF is represented with a ring polymer comprised of n beads. In the input file, **kinitial** allows the user to assign an initial momentum to each particle, whereas **vinitial** offers different ways to sample the initial velocity of each bead according to the value at **kinitial**.

vreverse

By construction, surface hopping methodologies conserve total quantum plus classical energy by enforcing the hopping trajectories to have enough kinetic energy to compensate for the potential energy difference between surfaces or states. Transition attempts not fulfilling this requirement are deemed as frustrated hops and return to the original states.[4] How to deal with frustrated hops has been an active research area. Here, we provide four different schemes for treating frustrated hops in RPSH (and FSSH as well). Assuming frustrated hops as trajectories hitting a wall and coming back, one can reverse the component of their velocity in the direction of the nonadiabatic coupling vector,[4, 10] specified by the keyword "always" in the input file. On the contrary, one can avoid reversing the velocity[11] by choosing the keyword "never" in the input file. Furthermore, we implemented two different remedies suggested by Truhlar[12] and Subotnik[13] to treat frustrated hops properly and rescale velocities, specified by keywords "delv1" and "delv2", respectively. The interested reader is

referred to the corresponding publications for more details and to our recent publication on the effect of proper treatment of frustrated hops on preserving detailed balance and retrieving correct quantum dynamics using RPSH.[3]

5.3 Sample Input File

Table 3: A sample input file (param.in) for Tully's single avoided crossing model.

param.in (Sample Input File)

model	Tully1
nparticle	1
nbeads	4
nsteps	3000
ntraj	1000
ncpu	1
timestep	1.0
kinitial	30.0
beadpos	Gaussian
vinitial	Fixed
vreverse	Never
rundtail	Yes
iprint	10
finish	

6 Output Files

6.1 List of output files

After completing the simulation, the following output files are produced

- *param.out* - list of all the model parameter(s) used in the simulation
- *energy_surface.out* - diabatic and adiabatic energy surace and non-adiabatic coupling vector strength

- *pop_adiabat1.out* - Adiabatic population by Method-I
- *pop_adiabat2.out* - Adiabatic population by Method-II
- *pop_diabat1.out* - Diabatic population by Method-I
- *pop_diabat2.out* - Diabatic population by Method-II
- *pop_diabat3.out* - Diabatic population by Method-III
- *pop_branch.out* - Brancing probability (if **usrKval** is **Yes**)
- *dcoupling.out* - detailed simulation output including position, momentum, energy, NAC, active energy surface, etc. (if **rundtail** is **Yes**)
- *hoppinghist.out* - history of hopping between different potential energy surfaces. (if **rundtail** is **Yes**)

6.2 Adiabatic vs. diabatic populations

SHARP pack runs quantum simulations in an adiabatic formalism; hence, the adiabatic state populations are the direct results. Two different schemes are implemented to calculate adiabatic state populations. In one way, the percentage of trajectories propagating on the adiabatic PES can represent the population of that state; these results appear in a file called *pop_adiabat1.out*. Alternatively, the adiabatic populations can be obtained from the complex-value electronic coefficients; these results appear in an output file called *pop_adiabat2.out*. Furthermore, three different schemes are implemented to obtain diabatic populations based on adiabatic results according to procedures suggested by Landry and Subotnik.[14] The results of the three schemes appear in files *pop_diabat1.out*, *pop_diabat2.out* and *pop_diabat3.out*.

The first column in all these files demonstrates time (a.u.); the rest are adiabatic or diabatic populations on corresponding potential energy surface(s).

7 Running Simulations

After compiling, **sharp.x** can be called upon to run the jobs in serial mode (ncpu 1). An input file, *'param.in'*, is required to run any SHARP pack simulation (see sample input in Table 3).

Alternatively, the SHARP pack can be run using the job submission bash script (see ***submist.sh*** script in bin/ directory). Based on ncpu in *'param.in'*, the code will run as a serial on a single node or parallel (trajectories are parallelized in this case) on different nodes of the cluster.

7.1 Example-1: ex1_tully1

To run FSSH/RPSH simulations for Tully's single avoided crossing model, change the directory to tully1, for example.

```
$ cd example/tully1/
```

Create an input parameter file ***'param.in'*** as shown in Table 3. The file has already been created in this case.

Copy the job submission script to the current directory and run the code with the job submission bash script.

```
$ sh submit.sh
```

Based on the **nbeads** used and the surface hopping methods, populations are calculated, which can be plotted by any graphing utility tool like GNUPLOT.

Use GNUPLOT script provided to generate a figure of potential energy surface using *energy_surface.out* data.

7.2 Example-2: ex2_tully2_parallel

This example shows the RPSH simulation of Tully's dual avoided crossing model with parallel trajectories. Set corresponding keywords to tully2. To run parallel simulations at eight cpu(s), set **ncpu** to **8** and **nbeads** to **4** for **RPSH** method simulation with four beads.

param.in (Tully Model-II: Parallel simulation)

```
⋮
model       tully2
nbeads      4
ncpu        8
⋮
```

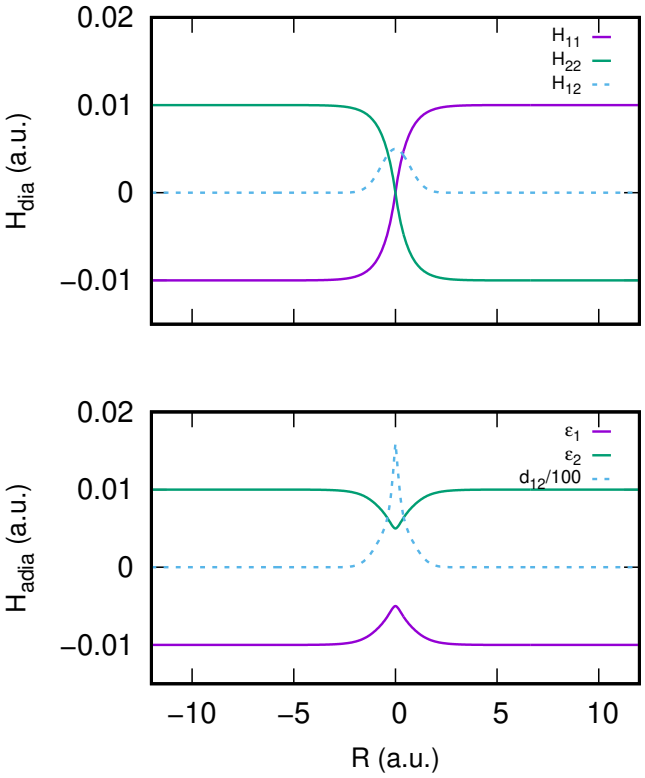

Figure 1: Diabatic and adiabatic potential energy surface for Tully1 model with nonadiabatic coupling strength as a function of position R

Submit your job using the bash script with **jobID=1**

```
$ sh submit.sh
```

It runs parallel jobs using 8 CPU(s), producing eight different run directories. When the jobs are completed, calculate the average population(s) by running *submit.sh* bash script with **jobID=2**.

7.3 Example 3: ex3_tully3

This example shows the calculation of the branching probabilities of Tully's extended coupling with the reflection model. As the branching probability determines the probability of transferring between different adiabatic states, we calculate these probabilities over a range of the initial momenta, k. So, we can simulate it manually for the different values of k at a time, or we can write a small bash job script for it. *job.sh* is a bash script to calculate the branching probability for momenta, k_1 to k_2 at the interval of dk. Choose these values based on your need and prepare the input file as

param.in (Branching Probability: Parallel simulation)

```
⋮
model       tully3
usrkval     Yes
acckval     Yes
⋮
```

usrKval and **acckval** both set to **Yes**, as the simulation reads $k-$value from the file and writes branching probability in *pop_branch.out*. Finally, calculate the average of the branching probabilities if running parallel simulations.

7.4 Example 4: ex4_db2linearchain

This example shows the detailed balance of a 2-state system coupled to an N-particle linear chain. Modify the Input File accordingly,

param.in (2-state N-Linear Chain for Detailed Balance)

```
⋮
model       db2lchain
nbeads      1
temp        500
⋮
```

Submit the job script for a serial/parallel run based on the **ncpu**. When the job is completed, use *pop_adiabat1.out* population to compare with exact results.

Use the plot script to generate the figure, as shown below.

```
$ gnuplot plot4.in
```

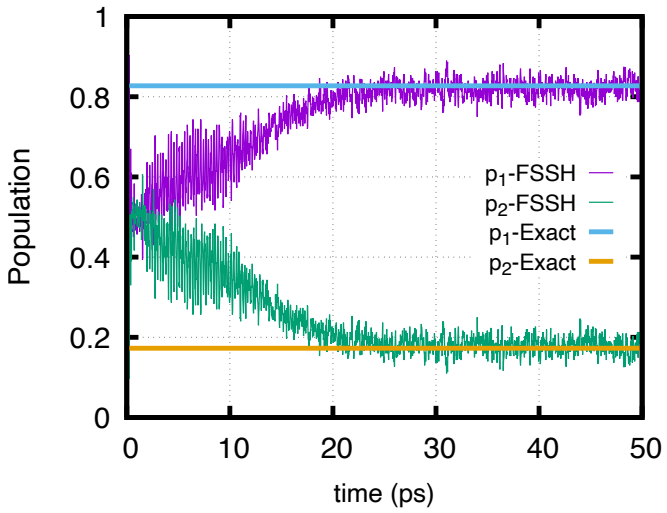

Figure 2: State populations at 1000 K, from a 300-trajectory ensemble, using the FSSH method. Solid horizontal lines are exact Boltzmann populations at thermal equilibrium.

References

(1) Shushkov, P.; Li, R.; Tully, J. C. *J. Chem. Phys.* **2012**, *137*, 22A549.

(2) Shakib, F. A.; Huo, P. *J. Phys. Chem. Lett.* **2017**, *8*, 3073–3080.

(3) Libmbu, D. K.; Shakib, F. A. *J. Phys. Chem. Lett.* **2023**, *14*, 8658–8666.

(4) Tully, J. C. *J. Chem. Phys.* **1990**, *93*, 1061–1071.

(5) Allen, M. P.; Tildesley, D. J., *Computer Simulation of Liquids*; Oxford: Clarendon Press: 1989.

(6) Bussi, G.; Parrinello, M. *Phys. Rev. E* **2007**, *75*, 056707.

(7) Coronado, E. A.; Xing, J.; Miller, W. H. *Chem. Phys. Lett.* **2001**, *349*, 521–529.

(8) Parandekar, P. V.; Tully, J. C. *J. Chem. Phys.* **2005**, *122*, 094102.

(9) Sifain, A. E.; Wang, L.; Prezhdo, O. V. *J. Chem. Phys.* **2016**, *144*, 211102.

(10) Hammes-Schiffer, S.; Tully, J. C. *J. Chem. Phys.* **1994**, *101*, 4657–4667.

(11) Müller, U.; Stock, G. *J. Chem. Phys.* **1997**, *107*, 6230–6245.

(12) Jasper, A. W.; Truhlar, D. G. *Chem. Phys. Lett.* **2003**, *369*, 60–67.

(13) Jain, A.; Subotnik, J. E. *J. Chem. Phys.* **2015**, *143*, 134107.

(14) Landry, B. R.; Subotnik, J. E. *J. Chem. Phys.* **2013**, *139*, 211101.