SHARC/COBRAMM DOCUMENTATION

19.05.2021

version 1.0

Contents

1 Introduction

This documentation covers the practical aspect of running mixed quantum mechanics/molecular mechanics (QM/MM) surface hopping simulations including arbitrary coupling, using the developed approach based on interfacing SHARC and COBRAMM suites. Theoretical aspects about the SHARC workflow for nonadiabatic dynamics and the COBRAMM workflow about QM/MM partition and energy calculation are reported on the reciprocal manuals.
The official manuals can be found at:

-SHARC: https://SHARC-md.org/?page_id=15
-COBRAMM: https://gitlab.com/cobrammgroup/cobramm/-/wikis/home

2 Environment definition

At least four different software need to be installed to run a QM/MM SHARC/COBRAMM calculation. An individual installation guide for each of the software can be found in the respective manual.

SHARC:
latest version 2.1. The suggested definition in ~/**.bashrc** is

```
export SHARC=/path_to_SHARC/SHARC_2.1/SHARC/bin
```

which includes **sharc.x** and the set of auxiliary scripts for preparation and analysis of the trajectories.

COBRAMM:
latest version 2.0. The suggested definition in ~/**.bashrc** is

```
export COBRAMM_PATH=/path_to_COBRAMM/
```

which includes the folder **cobramm**, which contains the executable Python scripts to run a COBRAMM calculation, and the folder util, which contains auxiliary scripts to setup COBRAMM calculation. COBRAMM reads an additional file with the information about the environments called **cobramm_profile**, hidden in the user's home. It is mandatory to specify in this file the AMBER environment for calculation of the MM energy and gradient.

AMBER:

The energy and gradient of the part of the system treated at MM level are obtained via AMBER using the COBRAMM interface.

The latest released version of AMBER is 20, tested versions are also 12 and 18. The path for AMBER needs to be specified in the **~/.cobramm_profile** . The suggested path is:

```
export AMBER_PREFIX=/path_to_amber/amberXX
export AMBERHOME=/path_to_amber/amberXX
```

QM software:

Currently, it is possible to run QM/MM nonadiabatic dynamics within the frame of SHARC/COBRAMM approach at the following level of theory: i) CASSCF, CASPT2 via the SHARC interface to MOLCAS (see section 6.2 of SHARC manual); ii) TD-DFT via the SHARC interface with ORCA (see section 611 of SHARC manual); iii) ADC(2) via the SHARC interface with TURBOMOLE (see section 6.8 of SHARC manual).

MOLCAS:

working version is OpenMolcas. Python 3 is required. The suggested definition in **~/.bashrc** is:

```
export MOLCAS=/path_to_molcas/openmolcas
```

ORCA:

Working version at least 4.2 The suggested definition in **~/.bashrc** is:

```
export ORCADIR=/path_to_orca/latest
```

TURBOMOLE:

Tested versions 6.6, 7.0, 7.1, 7.2, 7.3, 7.4, and 7.5. The suggested definition in **~/.bashrc** is:

```
export TURBODIR=/path_to_turbomole/ TURBOMOLE
```

3 The SHARC/COBRAMM approach

The SHARC/COBRAMM approach consists in combining the two software in order to simulate nonadiabatic dynamics in complex system through trajectory surface hopping, where the electronic energy is calculated at QM/MM level. SHARC suite performs the dynamics, propagating wavefunctions and nuclei, while COBRAMM, through SHARC interfaces to electronic structure calculation codes and its interface to AMBER, calculates the energies at the QM/MM level on the fly. In particular, the detailed tasks partition is the following:

QM energy:	QM software, *called by* SHARC_QM.py, *called by* sharcQMcalculator.py
MM energy:	AMBER, *called by* amberCalculator.py
QM/MM energy:	cobram.py, *called by* SHARC_COBRAMM.py
QM gradient:	QM software, *called by* SHARC_QM.py, *called by* sharcQMcalculator.py
MM gradient:	AMBER, *called by* amberCalculator.py
Point charges gradient:	QM software, *called by* SHARC_QM.py, *called by* sharcQMcalculator.py
QM/MM gradient:	cobram.py, *called by* SHARC_COBRAMM.py
Propagation:	sharc.x

3.1 SHARC/COBRAMM workflow

The SHARC/COBRAMM workflow is based on a three calls process. When the user submits the calculation, **sharc.x** calls the interface between SHARC and COBRAMM, called SHARC_COBRAMM.py. This interface connects the two codes and prepares the QM/MM calculation. In particular, it writes the **cobram.command**, input file for COBRAMM calculation and calls cobram.py, which starts the COBRAMM calculation. In a first stage, COBRAMM creates the system partition and calls AMBER for the MM part calculation. Regarding the QM part, COBRAMM i) writes a new **QM.in** file, including the atoms of only the QM part; ii) via **sharcQMcalculator.py** writes the point charges file in the format required by the QM code; iii) calls the **SHARC_QM.py** interface. **amber.py** and **SHARC_QM.py** interfaces regularly perform the calculations. **cobram.py** incorporates the results obtained from those and give back a **QM.out** file in SHARC format. **sharc.x** is then able to read this file and proceed with the wavefunction and nuclear propagation as regular in its surface hopping scheme.

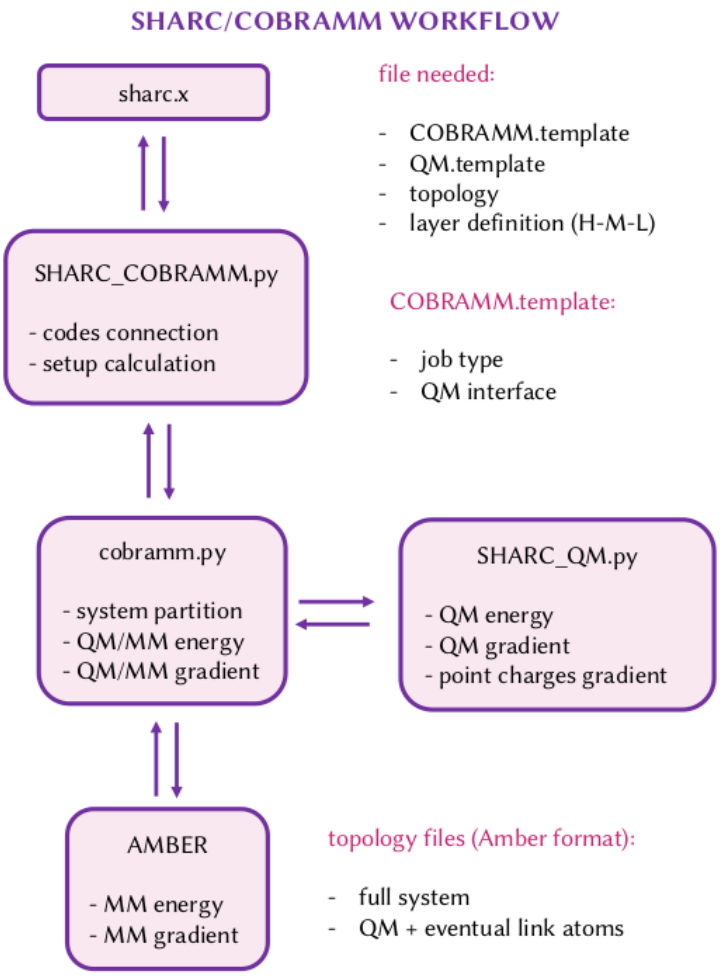

Figure 1: Schematic representation of SHARC/COBRAMM workflow

3.2 SHARC_COBRAMM.py interface

The basic structure of **SHARC_COBRAMM.py** interface and the logic behind the code are the same as other SHARC interfaces. In particular, the interface reads the **COBRAMM.template**, the **COBRAMM.resources** and the **QM.in** files. It creates the **SCRATCH** directory and a save directory (**SAVEDIRQMMM**), where a coordinate file in AMBER format for the previous (**real.crd.old**) and current (**real.crd**) time step is written each time step. The interface writes the input file for COBRAMM, **cobram.command**, and run cobram.py. It writes the **QMMM.err** and **QMMM.log** files during the dynamics.

3.3 sharcQMcalculator.py

sharcQMcalculator.py is called by cobram.py to run energy and gradient calculation of the QM part using a **SHARC_QM.py** interface. First of all, it writes the point charges relative to the MM part to be included in the QM calculation. Charges are written in a file called **charges.dat**, already

correctly formatted for the specific QM software, specified in **COBRAMM.template**. **sharcQMcalculator.py** calls the **SHARC_QM.py** interface. Once the calculation is done, it reads the **QM.out**, given by the **SHARC_QM.py** interface only for the QM part, the file **gradient_charges**, including the gradient of the point charges induced by the QM region, reads the MM output and writes a complete **QM.out** for the whole system to be used by **sharc.x**.

3.4 SHARC_QM.py modifications

Small modifications to each of the interfaces were required about reading the point charges file and returning the point charges gradient induced by the QM density. Each interface is adapted to the specific software but reads and writes the same files. All the interfaces currently available read the **charges.dat** file and print the **grad_charges** file. In case of ORCA and TURBOMOLE interfaces, the point charges gradient is explicitly written by the QM software, while in case of MOLCAS it is calculated from the electric field in the position of the point charges induced by the QM atoms, given by MOLCAS.

4 File required to run a QM/MM SHARC/COBRAMM calculation

Several files are needed in order to run a QM/MM SHARC/COBRAMM calculation, as required by one or the other suites. In particular, templates file read by SHARC, system partition file read by COBRAMM, topology files read by AMBER.

4.1 List of files required

4.1.1 COBRAMM.template

Contains the information about COBRAMM calculations. Mandatory entries are i) “jobtype”, which indicates the type of job (single point, optimization, frequency, see section 2.1.1.1 of COBRAMM manual); ii) “interface”, which indicates the name of the software will be called to calculate QM energy and gradient (“MOLCAS, “ORCA, “RICC2”).
For excited state calculations and nonadiabatic dynamics in the frame of SHARC/COBRAMM approach, COBRAMM only needs to run a single point calculation, indicated by the option “sp”.
An example of a **COBRAMM.template** is:

```
user@local:~> cat COBRAMM.template
```

```
interface ORCA
jobtype sp
```

This file must be generated manually.

4.1.2 QM.template

Contains the information about QM calculation run by **SHARC_QM.py** interfaces. MOLCAS, TURBOMOLE and ORCA interfaces are currently available, which correspond to **SHARC_MOLCAS.py**, **SHARC_RICC2.py** and **SHARC_ORCA.py** interface. Exemplary ORCA.template is:

```
user@local:~> cat ORCA.template
basis def2-SVP
functional B3LYP
charge 0
cobramm
```

which indicates the basis set, the functional, the charge of the system and a COBRAMM calculation. See SHARC manual (see chapter 6) for all the specifications and requirement of any of the template file. The entry "cobramm" is mandatory for all of the interfaces in order to perform SHARC/COBRAMM calculation.
This file must be generated manually, with exception of **MOLCAS.template** that can be generated by **$SHARC/molcas_input.py** (see section 6.4.3 of SHARC manual). It is suggested that the appropriate template is taken from **$SHARC/../examples** and adapted as necessary.

4.1.3 real.top

Topology file in AMBER format (see AMBER manual), containing the topologies for all the atoms of the full system. This file can be generated manually, or using **$COBRAM_PATH/cobramm-droplet.py**

4.1.4 model-H.top

Topology file in AMBER format (see AMBER manual), containing the topologies for the atoms of the QM region and the eventual link atoms (see section 1.2 of COBRAMM manual). This file can be generated manually, or using **$COBRAM_PATH/cobramm-droplet.py**

4.1.5 real_layers.xyz

File including some atom specifications, divided in 7 columns: i) atomic number; ii) frozen (0=not frozen); iii-v) initial coordinates, vi) layer of belonging (H: high layer=QM atom; M: medium layer=MM atom, mobile; L: lower layer= MM atom, frozen atom, see section 1.1 of COBRAMM manual); vii) eventual H link to atom indexed. This file does not contain blank lines. Exemplary **real_layers.xyz** file is:

```
user@local:~> cat real_layers.xyz
C     0      8.836000     8.252000     8.903000   H
O     0      8.702000     8.313000    10.108000   H
H     0      9.690000     7.612000     9.172000   H
C     0      7.880000     8.898000     7.932000   M H1
H     0      7.844000     9.977000     8.097000   M
H     0      6.877000     8.482000     8.054000   M
H     0      8.210000     8.714000     6.908000   M
O     0      4.060000    10.153000     5.923000   L
```

This file can be generated manually, or using **$COBRAM_PATH/cobramm-droplet.py**

5 Setup SHARC/COBRAMM QM/MM surface hopping simulation

5.1 How to get distribution of geometries and momenta

Getting an accurate and proper ensemble of initial condition is in the user's responsibility. In this chapter an overview on how to get the initial conditions for the surface hopping trajectories, within the SHARC frame, is given. First of all, a distribution of geometries and momenta is needed. The script **$SHARC/wigner.py** reads a frequency calculation in MOLDEN format and writes an **initconds** file, including a number of geometries specified by the user. Alternatively, AMBER and SHARC trajectories can be converted into an **initconds** file using the respective **$SHARC/amber_to_initconds.py** (see section 7.2 of SHARC manual) and **$SHARC/sharctraj_to_initconds.py** (see section 7.3 of SHARC manual) auxiliary scripts. Whether a Wigner distribution, classical MM, COBRAMM equilibration (see Tutorial 1 of COBRAMM manual) or SHARC trajectories are used to sample the initial conditions, it's important to end the procedure with an **initconds** file in SHARC format (see section 7.5.5 of SHARC manual). Specifically, no additional information needs to be added in the file in the **initconds** file in case of

SHARC/COBRAMM QM/MM calculation, but all the atoms of the system need obviously to be listed after every index.
In the Appendix, an exemplary combination of SHARC and COBRAMM scripts to obtain an **initconds** file is reported. This includes how to obtain an ensemble of **R** and **v,** for the QM part, from a Wigner distribution, how to equilibrate the solvent, and how to obtain the file required for the dynamics.

5.2 SHARC/COBRAMM initial condition calculation

Once the **initconds** file is written, the information about the excited states must be added, in order to decide which state will be considerate for the excited states dynamics for each trajectory initialized with a specific initial condition. This operation consists in two steps, specifically i) run an excited state calculation for each of the initial conditions; ii) analyze the results of these calculation and select the valid state to initialize the dynamics.

5.2.1 setup_cobramm_init.py

The excited state calculations can be setup with the auxiliary script **$SHARC/setup_cobramm_init.py**. This script follows the logic of the standard **$SHARC setup_init.py** script (see section 7.4 of SHARC manual) but includes at the beginning the information about the QM/MM calculation with COBRAMM. This script is interactive, and it will first ask the user to define the file required (**COBRAMM.template**, **real.top**, **model-H.top** and **real-layers.xyz**), to set-up the paths to COBRAMM and AMBER, memory and processor will be used by COBRAMM. Then it will ask information to setup the QM calculation, as in the **$SHARC/setup_init.py** script.

5.2.3 ICOND_xxxxx folders

Additionally, the script creates individual folders for each of the initial condition called, **ICOND_xxxxx** and a submission script for the excited states calculation:

```
user@local:/path_to_directory> ls
all_qsub_init.sh    all_run_init.sh    ICOND_00000    ICOND_00001    ICOND_xxxxx
```

The script copies all the file required for the dynamics (see chapter 4), creates a QM.in file, writes files called **COBRAMM.resources** and **QM.resource** (with QM=interface name), containing all

the specifics for the calculation, and a submission script called **run.sh**, in each of the folder, which looks like:

```
user@local:/path_to_directory> ls ICOND_00000
COBRAMM.resources    QM.resources model-h.top    real_layers.xyz    QM.in
COBRAMM.template     QM.template  real.top       run.sh
```

The calculation can be simply run by executing **run.sh**.

5.2.4 Output of initial conditions calculations

Once the calculations are completed, the folder contains a link to the SCRATCH directory, previously specified by the user in **$SHARC/setup_cobramm_init.py**, called SCRATCH and a save directory called **SAVEDIRQMMM**, where for example, if required, the determinant can be stored for overlap calculation. Additionally, two pairs of log and error files are present. Log files (**QM.log** and **QMMM.log**) contain the details and the progresses of only the QM and of the QM/MM calculations, respectively. Error files (**QM.err** and **QMMM.err**) contain the eventual errors of the only QM and of the QMMM calculations, respectively. An output file (**QM.out**) contains the result of the QM/MM calculations. The files resulting from the COBRAMM calculation are stored in the **SCRATCH** directory and just a summary is reported in **QMMM.log**. The **ICOND_xxxxx** looks as the following at the end of the calculation:

```
user@local:/path_to_directory> ls ICOND_00000
COBRAMM.resources    QM.resources model-h.top    real_layers.xyz    QM.in
COBRAMM.template     QM.template  real.top       run.sh             SAVEDIRQMM  QM.err
QM.log    QM.out.    QMMM.err     QMMM.log       SCRATCH
```

5.4.5 The SCRATCH directory

The **SCRATCH** directory is divided in two sub-folders, called QM and QMMM, where respectively the QM and QM/MM calculations are performed. In the **SCRATCH/QMMM** folder the **cobram.command** (see section 2.1 of COBRAMM manual) is written, as well as a file called charge.dat, containing the information about the point charges to be included in the QM calculation. **cobram.py** is run and it calls AMBER to compute the MM energy. All the files about the setup and the result of such calculation are stored here. cobram.py creates a new **QM.in** file, including only the QM atoms, and calls the **SHARC_QM.py** interface. The QM calculation is run

in the **SCRATCH/QM** directory, which corresponds to the **SCRATCH** directory for a QM calculation as by default in SHARC (see section 7.4.3 of SHARC manual). In the **SHARC/QMMM** folder, the information from the QM and MM calculation are stored and integrated by **cobram.py** that returns the output file.

5.4.6 excite.py

The interactive script **$SHARC/excite.py** (see section 7.5 of SHARC manual) can be used to process the information obtained about the excited state, to update the **initconds** file and to select initial states for the dynamics for each trajectory. The script will write a new file, called **initconds.excited** (see section 7.5.5 of SHARC manual).

5.3 SHARC/COBRAMM trajectories

5.3.1 setup_cobramm_traj.py

The **initconds.excited** file can be read by the interactive script **$SHARC/setup_cobramm_traj.py**. This script, following the same logic of the script **$SHARC/setup_traj.py** (see section 7.6 of SHARC manual) is in charge to setup the dynamics calculation, creating an individual folder for each of the trajectories. This script first asks the information about COBRAMM calculation, in particular the names of the required files (see chapter 4), the paths for COBRAMM and AMBER installation, the memory and number of processors will be used by COBRAMM and AMBER. Additionally, the script asks the user about the possibility of applying RATTLE algorithm to restrain the distance between two atoms and the velocities of those atoms (see section 5.3.5).

5.3.2 Singlet_x/TRAJ_xxxxx folders

$SHARC/setup_cobramm_traj.py creates two submission scripts and a separate folder for each initial excited state where the trajectories will be initialized:

```
user@local:/path_to_directory> ls
all_qsub_traj.sh    all_run_traj.sh    Singlet_1    Singlet_2
```

Inside these directories, the script creates specific folders for each of the trajectories:

```
user@local:/path_to_directory> ls Singlet_1
TRAJ_00001          TRAJ_00002          TRAJ_xxxxx
```

Each of these folders contains the input files required by SHARC (**input**, **geom**, **veloc**, written by **$SHARC/setup_cobramm_traj.py**), the files, if required, **rattle**, to restrain specific bonds, and **atommask** which indicates which atoms are exclude from the kinetic energy used do dampen inactive population in the energy based decoherence correction (recommended for all the MM atoms, see section 4.1.3 of SHARC manual), and a submission script **run.sh**.

```
user@local:/path_to_directory> ls Singlet_1/TRAJ_00001
input          geom          veloc          run.sh  QM
```

A folder called QM is created in each of the **TRAJ_xxxxx** folder, containing all the files required for the dynamics (see chapter 4), the files called **COBRAMM.resources** and **QM.resource** (with QM=interface name), containing all the specifics for the calculation, and a submission script called **runQM.sh**:

```
user@local:/path_to_directory> ls Singlet_1/TRAJ_00001/QM
COBRAMM.resources       QM.resources        model-h.top    real_layers.xyz   QM.in
COBRAMM.template        QM.template         real.top       run.sh
```

5.3.3 Output of dynamics calculation

Once the trajectory is submitted, **sharc.x** will create a series of output files (**output.lis**, **output.dat**, **output.log**) in the trajectory folder **TRAJ_xxxxx**. These files remain as a standard SHARC dynamics (see chapter 5 of SHARC manual). Analogously, a restart folder and files called **restart.ctrl** and **restart.traj** are created in this directory, in order to be able to restart the trajectory if the calculation is interrupted.

```
user@local:/path_to_directory> ls Singlet_1/TRAJ_00001
input          geom          veloc          output.dat          output.lis          output.log
run.sh         QM            restart
```

The QM folder will contain a link to the SCRATCH directory, previously specified by the user in **$SHARC/setup_cobramm_traj.py**, called SCRATCH and a save directory called **SAVEDIRQMMM**. Additionally, two pairs of log and error file are present. Log files (**QM.log** and **QMMM.log**) contain the detail and the progresses of the only QM and of the QM/MM calculations, respectively. Error files (**QM.err** and **QMMM.err**) contain the eventual errors of the only QM and of the QMMM calculations, respectively. An output file (**QM.out**) contains the result of the QM/MM calculations required by **sharc.x** to propagate the wavefunction and nuclei. The files resulting from the COBRAMM calculation are stored in the **SCRATCH** directory and just a summary is reported in **QMMM.log**. The QM at the end of the calculations looks like in the following:

```
user@local:/path_to_directory> ls Singlet_1/TRAJ_00001/QM
COBRAMM.resources   QM.resources    model-h.top   real_layers.xyz   QM.in
COBRAMM.template    QM.template     real.top      runQM.sh          SAVEDIRQMM
SCRATCH             QM.err          QM.log        QM.out            QMMM.err
QMMM.log
```

5.3.4 The SCRATCH directory

The **SCRATCH** directory is divided in two sub-folders, called QM and QMMM, where respectively the QM and QMMM calculations are performed. The QM/MM calculation is performed in the **SCRATCH/QMMM** folder. Here, the **cobram.command** (see section 2.1 of COBRAMM manual) is written, as well as a file called charge.dat, containing the information about the point charges to be included in the QM calculation. cobram.py is run and it calls AMBER to compute the MM energy and gradient. All the files about the setup and the result of such calculation are stored here. cobram.py creates a new QM.in file, including only the QM atoms, and calls the **SHARC_QM.py** interface. The QM calculation is run in the **SCRATCH/QM** directory, correspond to the SCRATCH directory for a QM calculation as by default in SHARC. In the **SHARC/QMMM** folder, the information from the QM and MM calculation are stored and integrated by cobram.py that returns the output file.

5.3.5 rattle file

In case the user wants to use the RATTLE algorithm to constrain X-H bond distances, the name of a file containing information about which atoms to restrain and to which value, needs to be given.

This file requires three columns per lines: the first two are the atom index of the atoms to constrain, the third column (optional) the distance (Bohr) at which the bond between the atoms is fixed. In case the third option is not given, the value of the constrain is calculated based on the distance between the same two atoms at the initial time step of the dynamics. An example of **rattle** file is the following:

```
user@local:/path_to_directory> cat rattlefile
4       5       2.4
5       6
```

5.3.6 atommask file

An **atommask** indicates which atoms are not excluded from kinetic energy rescale after a hop and from the kinetic energy used do dampen inactive population in the energy based decoherence correction. The **atommask** consist of only one column and each line indicates the option for atom indexed in **QM.in** with the correspondent line number. The options are "F", if the atom is excluded, "T" if the atom is not excluded. The information about all the atoms is required. An example of **atommask** is the following:

```
user@local:/path_to_directory> cat atommask
T
T
F
F
```

which excludes only the atom number 3 and 4. The atommask file is generated automatically by **$SHARC/setup_cobramm_traj.py**, if required by the user, who has to specify the index of the atoms to be excluded.

5.3.7 Analysis of the trajectories

Postproduction analysis represents an important step for understanding the result of SHARC dynamics. Populations analysis, populations fit, geometries analysis and many more possibilities are available via auxiliary script included in SHARC. See SHARC manual for a detail description of them (see chapter 7 of SHARC manual).

Appendix

Example of initial conditions generation

A possible approach to generate initial condition consists in the following steps, result of a combination of auxiliary scripts available in SHARC and COBRAMM suites.

In this example a single chromophore (QM) is solvated in water (MM) is considered.

1) The first step consists in obtaining a geometry optimization and frequency calculation. It would be consistent to have it at the ground state correspondence of the level of theory the excited states dynamics will be performed. It would be optimal if this calculation can be obtained including the solvent effect with implicit solvation methods.
2) A Wigner distribution is calculated with **$SHARC/wigner.py** (see section 7.1 of SHARC manual) and an initial condition file is generated with a number of geometries indicated by the user. The generation of a file containing the coordinates of the sampled geometries can be required, in xyz format, with the option -x and be generated (default **initconds.xyz**). From the **initconds.xyz** file is easy to extract single geometries. The following steps will be explained for a single geometry but can easily be applied to all the geometries with simple self-written scripts. An auxiliary script automatizing this step will be available soon, but has not been implemented yet.
3) The geometry can be solvated by **$COBRAM_PATH/util/cobramm-solvatedchromo.py** script (see section 6.4.1 of COBRAM manual), which creates topology and coordinate files in AMBER format for a solvated system of solvent and size indicated by the user.
4) The file generated can be read by **$COBRAM_PATH/util/cobramm-equilibration.py** script (see section 6.4.2 of COBRAM manual) that sets up a minimization, heating and equilibration run with AMBER. In order to preserve the Wigner geometries and momenta, and just equilibrate the solvent around these geometries, the QM part must be frozen. In order to freeze the QM part, the input files for these calculations need to be modified. In particular adding the following lines:

```
 ibelly = 1,
bellymask = '@O,H1,H2'
```

to the minimization, heating and equilibration input files, only O, H1 and H2 atoms, atom belonging to the water molecules, are allowed to move. Any other atoms, the QM atoms, will be frozen.

5) At the end of the equilibration, the analysis tool of AMBER, **cpptraj** (see chapter 32 of AMBER 20 manual), reads the restart file (.rst) of the equilibration, and centers the chromophore in a shell with a number of water molecules around it. These number of water molecules around the chromophore can be manually selected. An example of commands for **cpptraj** are:

```
parm molecule_solv.top
reference molecule_solv.crd
trajin eq.rst
autoimage
rms reference :1
solvent byres :WAT
closestwaters :1 500
trajout center.rst7
go
quit
```

which selects the closest 500 water molecules around the chromophore and write the coordinates of the system in rst7 AMBER format. An auxiliary script automatizing this step will be available soon, but has not been implemented yet.

6) The script **$COBRAM_PATH/util/cobramm-droplet.py** (see section 6.4.3 of COBRAM manual) can be now used to generate a droplet of specified size, with a first radius including mobile solvent molecules (M layer) and an eventual second radius including the frozen molecules during the dynamics (L layer). This script generates the file **real.top**, **model-H.top** and **real_layers.xyz** required to run the dynamics.

7) The script **$SHARC/amber_to_initconds.py** (see section 7.2 of SHARC manual) can write an **initconds** files reading information from AMBER trajectories in rst7 format. The topology file of the system in AMBER format and the **center.rst7** trajectory files for all the geometries must be parsed. A new **initconds** file is created. This file contains the relative coordinates of the QM part from the Wigner distribution, but not the velocities.

8) The script **$SHARC/combine_initconds.py** is able to read the velocities for the original **initconds** file (to be renamed **initconds_original**) and include it in the new **initconds** file, combining the **R,v** of the QM and MM parts in a file called **initconds_new**.

9) With this file it is possible to standardly setup the trajectories with SHARC scripts. Nevertheless, two more additional steps are suggested.

10) A rattle file to constrain the distance between pair of X-H atoms can be written manually or via **$COBRAM_PATH/util/cobramm-rattle.py** (see section 6.44 of COBRAM manual; in this case, remember to remove the first and last line to transform in the right format readable by **sharc.x**).
11) An equilibration of the ground states at the SHARC/COBRAMM level of theory is recommended. This is to completely equilibrate the system and to avoid artificial kinetic energy adjustment and temperature gradients during the excited states dynamics. In order to do that, it's it possible to set-up SHARC/COBRAMM trajectories at the ground state correspondent level of theory of the one chosen for the dynamics (CASSCF, DFT, MP2) for, if possible, some hundreds of femtoseconds. Once these trajectories are completed, the **$SHARC/sharctraj_to_initconds.py** (see section 7.3 of SHARC manual) can be used to generate a new **initconds** file that can be used to setup the excited states dynamics.