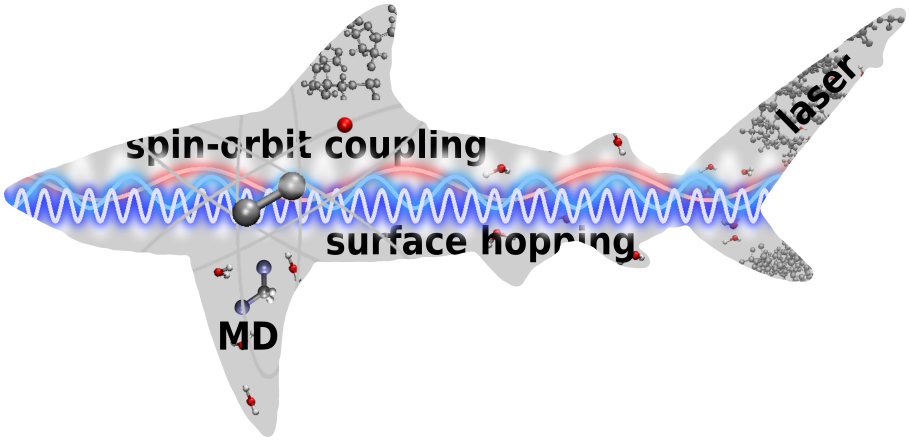

SHARC: Surface Hopping in the Adiabatic Representation Including Arbitrary Couplings

Manual

Version 1.0

AG González
Institute of Theoretical Chemistry
University of Vienna, Austria

universität wien

Vienna, November 17, 2014

Contact:

AG González
Institute of Theoretical Chemistry, University of Vienna
Währinger Straße 17
1090 Vienna, Austria

Website: sharc-md.org
Email: sharc@univie.ac.at

Contents

1 Introduction

When a molecule is irradiated by light, a number of dynamical processes can take place, in which the molecule redistributes the energy among different electronic and vibrational degrees of freedom. Kasha's rule [1] states that radiationless transfer from higher excited singlet states to the lowest-lying S_1 excited state is faster than fluorescence (F). This radiationless transfer is called internal conversion (IC) and involves a changes between electronic states of the same multiplicity. If a transition occurs between electronic states of different spin, the process is called intersystem crossing (ISC). A typical ISC process is from a singlet to a triplet state, and once the lowest triplet is populated, phosphorescence (P) can take place. In figure 1.1, radiative (F and P) and radiationless (IC and ISC) processes are summarized in a so-called Jabłonski diagram.

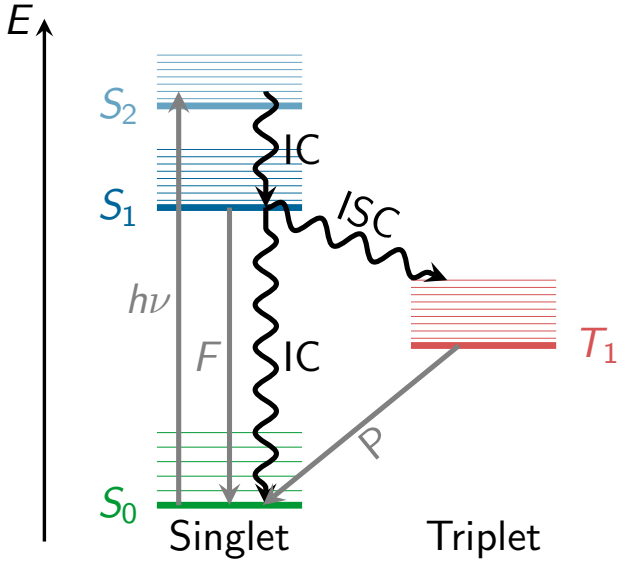

Figure 1.1: Jabłonski diagram showing the conceptual photophysical processes. Straight arrows show radiative processes: absorption ($h\nu$), fluorescence (F), and phosphorescence (P); wavy arrows show radiationless processes: internal conversion (IC) and intersystem crossing (ISC).

The non-radiative IC and ISC process are fundamental concepts which play a decisive role in photochemistry and photobiology. IC processes are present in the excited-state dynamics of many organic and inorganic molecules, whose applications range from solar energy conversion to drug therapy. Even many, very small molecules, for example O_2 and O_3, SO_2, NO_2 and other nitrous oxides, show efficient IC, which has important consequences in atmospheric chemistry and the study of the environment and pollution. IC is also the first step of the biological process of visual perception, where the retinal moiety of rhodopsin absorbs a photon and non-radiatively performs a torsion around one of the double bonds, changing the conformation of the protein and inducing a neural signal. Similarly, protection of the human body from the influence of UV light is achieved through very efficient IC in DNA, proteins and melanins. Ultrafast IC to the electronic ground state

allows quickly converting the excitation energy of the UV photons into nuclear kinetic energy, which is spread harmlessly as heat to the environment.

ISC processes are completely forbidden in the frame of the non-relativistic Schrödinger equation, but they become allowed when including spin-orbit couplings, a relativistic effect [2]. Spin-orbit coupling depends on the nuclear charge and becomes stronger for heavy atoms, therefore it is typically known as a "heavy atom" effect. However, it has been recently recognized that even for molecules with only first- and second-row atoms, ISC might be relevant and can be competitive in time scales with IC. A small selection of the growing number of molecules where efficient ISC in a sub-ps time scale has been predicted are SO_2 [3–5], benzene [6], aromatic nitrocompounds [7] or DNA nucleobases and derivatives [8–12].

Theoretical simulations can greatly contribute to understand non-radiative processes by following the nuclear motion on the excited-state potential energy surfaces (PES) in real time. These simulations are called excited-state dynamics simulations. Since the Born-Oppenheimer approximation is not applicable for this kind of dynamics, non-adiabatic effects need to be incorporated into the simulations.

The principal methodology to tackle excited-state dynamics simulations is to numerically integrate the time-dependent Schrödinger equation, which is usually called full quantum dynamics simulations (QD). Given accurate PESs, QD is able to reach or surpass experimental accuracy. However, the need of an "a priori" knowledge of the full multi-dimensional PES renders this type of simulations quickly unfeasible for more than few degrees of freedom. Several alternative methodologies are possible to alleviate this problem. One of the most popular ones is to use surface hopping non-adiabatic dynamics.

Surface hopping was originally devised by Tully [13] and greatly improved later by the "fewest-switches criterion"[14] and it has been reviewed extensively since then, see e.g. [15–17]. In surface hopping, the motion of the excited-state wavepacket is approximated by the motion of an ensemble of many independent, classical trajectories. Each trajectory is at every instant of time tied to one particular PES, and the nuclear motion is integrated using the gradient of this PES. However, non-adiabatic population transfer can lead to the switching of a trajectory from one PES to another PES. This switching (also called "hopping", which is the origin of the name "surface hopping") is based on a stochastic algorithm, taking into account the change of the electronic population from one time step to the next one.

The advantages of the surface hopping methodology and thus its popularity are well summarized in Ref. [15]:

- The method is conceptually simple, since it is based on classical mechanics. The nuclear propagation is based on Newton's equations and can be performed in cartesian coordinates, avoiding any problems with curved coordinate systems as in QD.
- For the propagation of the trajectories only local information of the PESs is needed. This avoids the calculation of the full, multi-dimensional PES in advance, which is the main bottleneck of QD methods. In surface hopping dynamics, all degrees of freedom can be included in the simulation. Additionally, all necessary quantities can be calculated on-demand, usually called "on-the-fly" in this context.
- The independent trajectories can be trivially parallelized.

The strongest of these points of course is the fact that all degrees of freedom can be included easily in the calculations, allowing to describe large systems. One should note, however, that surface hopping methods in the standard formulation [13, 14] – due to the classical nature of the trajectories – do not allow to treat some purely quantum-mechanical effects like tunneling, (tunneling for selected degrees of freedom is possible [18]). Additionally, quantum coherence between the electronic states is usually described poorly, because of the independent-trajectory ansatz. This can be treated with

some ad-hoc corrections, e.g., in [19].

In the original surface hopping method, only non-adiabatic couplings are considered, only allowing for population transfer between electronic states of the same multiplicity. The SHARC methodology is a generalization of standard surface hopping since it allows to include any type of coupling. Beyond non-adiabatic couplings (for IC), spin-orbit couplings (for ISC) or interactions of dipole moments with electric fields (to explicitly describe laser-induced processes) can be included. A number of methodologies for surface hopping including one or the other type of potential couplings have been proposed in references [20–26], but SHARC can include all types of potential couplings on the same footing.

The SHARC methodology is an extension to standard surface hopping which allows to include these kinds of couplings. The central idea of SHARC is to obtain a fully diagonal Hamiltonian, which is adiabatic with respect to all couplings. The diagonal Hamiltonian is obtained by unitary transformation of the Hamiltonian including all couplings. Surface hopping is conducted on the transformed electronic states. This has a number of advantages over the standard surface hopping methodology, where no diagonalization is performed:

- Potential couplings (like spin-orbit couplings and laser-dipole couplings) are usually delocalized. Surface hopping, however, rests on the assumption that the couplings are localized and hence surface hops only occur in the small region where the couplings are large. Within SHARC, by transforming away the potential couplings, additional terms of non-adiabatic (kinetic) couplings arise, which are localized.
- The potential couplings have an influence on the gradients acting on the nuclei. To a good approximation, within SHARC it is possible to include this influence in the dynamics.
- When including spin-orbit couplings for states of higher multiplicity, diagonalization solves the problem of rotational invariance of the multiplet components (see [24]).

The SHARC suite of programs is an implementation of the SHARC method. Besides the core dynamics code, it comes with a number of tools aiding in the setup, maintenance and analysis of the trajectories.

1.1 Capabilities

The main features of the SHARC suite are:

- Non-adiabatic dynamics based on the surface hopping methodology able to describe internal conversion and intersystem crossing with any number of states (singlets, doublets, triplets, or higher multiplicities).
- Algorithms for stable wavefunction propagation in the presence of very small or very large couplings.
- Inclusion of interactions with laser fields in the long-wavelength limit. The derivatives of the dipole moments can be included in strong-field applications.
- Propagation using either non-adiabatic couplings vectors $\langle\alpha|\frac{\partial}{\partial\mathbf{R}}|\beta\rangle$, time-derivative couplings $\langle\alpha|\frac{\partial}{\partial t}|\beta\rangle$ or wavefunction overlaps $\langle\alpha(t_0)|\beta(t)\rangle$ (via the local diabatization procedure [19]).
- Gradients including the effects of spin-orbit couplings (with the approximation that the diabatic spin-orbit couplings are slowly varying).
- Energy-difference-based partial coupling approximation to speed up calculations [27].
- Energy-based decoherence correction [19].
- Flexible interface to quantum chemistry programs. Existing interfaces to MOLPRO 2010 and 2012, MOLCAS 7.8 and 8.0, and to COLUMBUS 7.0 (only if interfaced to MOLCAS). An interface to implement dynamics based on analytical expressions is also available.
- Calculation of Dyson norms for single-photon ionization spectra through the COLUMBUS interface.

- Suite of auxiliary Python scripts for all steps of the setup procedure and for various analysis tasks.
- Comprehensive tutorial.

1.2 References

The following references should be cited when using the SHARC suite:

- [28] M. Richter, P. Marquetand, J. González-Vázquez, I. Sola, L. González: "SHARC: ab Initio Molecular Dynamics with Surface Hopping in the Adiabatic Representation Including Arbitrary Couplings". *J. Chem. Theory Comput.*, **7**, 1253–1258 (2011).
- [29] S. Mai, M. Richter, M. Ruckenbauer, M. Oppel, P. Marquetand, L. González: "SHARC: Surface Hopping Including Arbitrary Couplings – Program Package for Non-Adiabatic Dynamics". `sharc-md.org` (2014).

Further details can be found in the following references:

The theoretical background of SHARC is described in Refs. [28, 30–32].

Applications of the SHARC code can be found in Refs. [4, 9, 11, 33, 34].

Other features implemented in the SHARC suite are described in the following references:

- Energy-based decoherence correction: [19].
- Local diabatization and wavefunction overlap calculation: [27, 35, 36].
- Sampling of initial conditions from a quantum-mechanical harmonic Wigner distribution: [37, 38].
- Excited state selection for initial condition generation: [39].
- Calculation of ring puckering parameters and their classification: [40, 41].

The quantum chemistry programs to which interfaces with SHARC exist are described in the following sources:

- COLUMBUS: [42–45],
- MOLPRO: [46, 47],
- MOLCAS: [48, 49].

1.3 Authors

The current version of the SHARC suite has been programmed by Sebastian Mai and Martin Richter of the AG González of the Institute of Theoretical Chemistry of the University of Vienna with contributions from Jesús González-Vázquez, Philipp Marquetand, Matthias Ruckenbauer, Markus Oppel and Leticia González.

1.4 Suggestions and Bug Reports

Bug reports and suggestions for possible features can be submitted to sharc@univie.ac.at.

1.5 Notation in this Manual

Names of programs The SHARC suite consists of Fortran90 programs as well as Python and Shell scripts. The executable Fortran90 programs are denoted by the extension **`.x`**, the Python scripts have the extension **`.py`** and the Shell scripts **`.sh`**. Within this manual, all program names are given in **`bold monospaced font`**.

Shaded Sections Important sections are given in blue boxes like the following one:

Important sections are given in blue boxes like this one.

On the other hand, examples of input files and command lines are marked like this:

```
user@host> example example.dat
```

2 Installation

2.1 How To Obtain

SHARC can be obtained from the SHARC homepage www.sharc-md.org. In the Download section, register with your e-mail adress and affiliation. You will receive a download link to the stated e-mail adress. Clicking on the link in the email will download the archive file containing the SHARC package. Note that the link is active only for 24 h and the number of downloads is limited.

Note that you must accept the Terms of Use given in the following section in order to download SHARC.

2.2 Terms of Use

1. Recitals

The University of Vienna (in following the "University"), whose principal address is at Universitätsring 1, 1010 Vienna, Austria, developed the Software "SHARC: Surface Hopping with Arbitrary Couplings. A program suite to study the excited-state dynamics of molecules" (in the following the "Software").

The Software is not to be used for commercial purposes. Our Software provided on the relevant website is solely to be used for research, teaching and demonstration purposes and is free of any charge. The user of the Software is required to have the appropriate knowledge and know-how to use the Software (in the following the "User"). Any User should seek further guidance and make independent enquiries before relying on the Software.

2. Research, Teaching and Demonstration Purposes, Content and Liability

The Software on this website is based on scientific work and is solely to be used for research, teaching and demonstration purposes. The University, its organs, associated companies, employees and representatives do not warrant, represent or guarantee the functionality of the Software whatsoever.

The use of the Software is at your own risk and your own responsibility. The University, its organs, associated companies, employees and representatives do not assume any liability for the use the Software including its results. The University does not warrant, represent or guarantee any quality or performance of the Software and shall not be responsible for including but not limited to

- any contents of the Software in any kind;
- any intermediate or permanent amendments of the Software of any kind;
- any damages of any kind arising out of or in connection with the use of the Software (including without limitation damages for any consequential loss or loss of business opportunities or projects, or loss of profits) including, but not limited to, data theft or data loss;

- any errors, faults, incompleteness, inaccuracy, inadequacy or unsuitability or incorrectness of the provided Software or its results;
- any personal damages or damages to property due to the use of the Software;
- any damages of computers, any viruses or Trojans or similar which were transferred or conveyed by the Software;
- any defaults or incompleteness of the Software including any loss or damages of any kind due to or arising out of the use of the Software provided on the website of the University;
- any damages caused by third persons or subcontractors.

Hence, the University explicitly reserves the right to amend, supplement, delete or remove the Software, partially or as a whole, at any time and without prior notice.

The University explicitly refrains from and does not assume any liability whatsoever for any products or services offered or advertised by third persons via the provided Software.

3. Acknowledgements

The User shall acknowledge the authors or originators and use of the Software in the publication of any results achieved through use of the Software in the form of including the following references:

- S. Mai, M. Richter, M. Ruckenbauer, M. Oppel, P. Marquetand, L. González: "SHARC: Surface Hopping Including Arbitrary Couplings - Program Package for Non-Adiabatic Dynamics". www.sharc-md.org (2014).
- M. Richter, P. Marquetand, J. González-Vázquez, I. Sola, L. González: "SHARC: ab Initio Molecular Dynamics with Surface Hopping in the Adiabatic Representation Including Arbitrary Couplings". J. Chem. Theory Comput., 7, 1253-1258 (2011).

The User shall reproduce a copyright notice on every copy of the Software including partial copies and on any accompanying manuals and documentation in the form "Copyright © 2014 University of Vienna. All rights reserved." Trademark and other proprietary notices must also be reproduced but the User has no longer the right to use the name, arms, trademark, logo or other designation of the University of Vienna.

4. Intellectual Property

The contents of this Software are exclusively governed and construed by Austrian intellectual property laws. Any copying, editing, distribution or use of the contents require the prior consent of the respective authors or originators which is not granted or deemed to be granted without the prior written consent as described above.

If and to the extent the User amends, edits or otherwise supplements the Software with own information or other information of third persons, the User shall be deemed as the new operator of the Software. Consequently, the User expressly assumes any and all liability in connection with the Software.

5. Indemnity

The User is irrevocably obliged to indemnify the University, its employees, its organs, its associated companies and representatives for any damages in particular claims and damage claims including legal cost for

- any use of the Software by the User;

- any violation or infringement of this terms of use partially or in whole;
- any violation of rights of third persons in particular of intellectual property rights or personality rights of publicity or
- any claims of third persons due to the use of the Software.

This obligation of the User to indemnify shall not impede any other provision of this disclaimer described herein and shall remain in force and effect after the User has ceased to use the Software.

If and to the extent third persons assert any claims against the University due to or arising out of the use by any User, such User shall be obliged to indemnify and hold harmless the University for any damages asserted against the University.

6. Data Protection

The User explicitly acknowledges that the University uses services of third party suppliers (e.g. Google Analytics etc.) and consents to the use of its data by the third party supplier. Further, the User consents to the use of the respective IP address of the User or similar by third persons (e.g. for statistical use). The University shall be entitled to provide third persons with such data for statistical purposes. If personal data of Users is provided to or used by third persons, the University shall not be deemed as a sponsor pursuant to the Austrian Data Protection Act 2000.

7. Final Provisions

The Software was developed in Austria and is applicable through a server located in Austria. Any and all interpretation of the content of the Software, any claims with regards to the Software and any disputes in connection with the Software shall be governed by Austrian law excluding any applicable collision provisions. Exclusive place of jurisdiction shall be Vienna, Austria.

2.3 Installation

In order to install and run SHARC under Linux (Windows and OS X are currently not supported), you need the following:

- A Fortran90 compiler (This release is tested against GNU Fortran 4.4.7 and Intel Fortran 15.0).
- The BLAS, LAPACK and FFTW3 libraries.
- Python 2 (This release is tested against Python 2.6.6 and 2.7.3).

The source code of the SHARC suite is distributed as a tar archive file. In order to install it, first extract the content of the archive to a suitable directory:

```
tar -xzvf sharc.tgz
```

This should create a new directory called **sharc/** which contains all the necessary subdirectories and files.

To compile the Fortran90 programs of the SHARC suite, go to the **source/** directory.

```
cd source/
```

and edit the **Makefile** by adjusting the **F90** variable to point to the Fortran compiler of your choice. Issuing the command:

```
make
```

will compile the source and create all the binaries.

```
make install
```

will copy the binary files into the **sharc/bin/** directory of the SHARC distribution, which already contains all the python scripts which come with SHARC. In figure 2.1 the directory structure of the complete SHARC directory is shown.

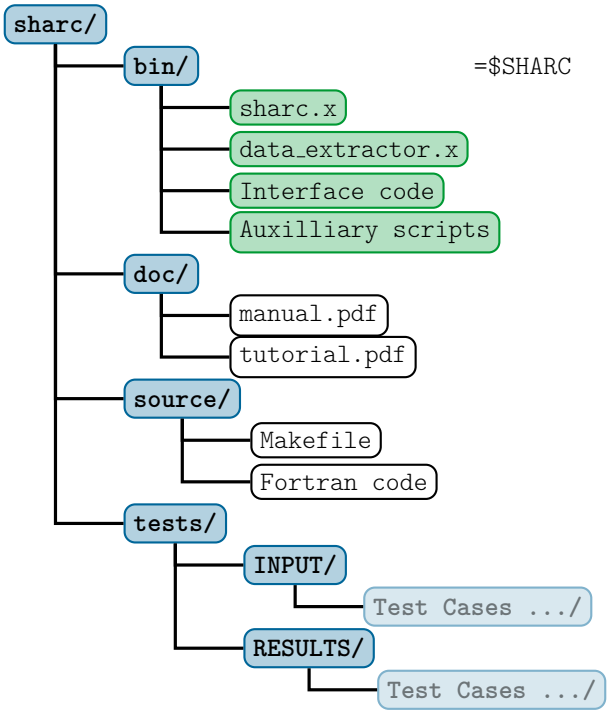

Figure 2.1: Directory tree containing a complete SHARC installation.

In order to use the SHARC suite, set the environment variable **$SHARC** to the **bin/** directory of the SHARC installation. This ensures that all programs of the SHARC suite find the other executables and all calls are successful. For example, if you have unpacked SHARCinto your home directory, just set:

```
export SHARC=~/sharc/bin   (for bourne shell users)
```

or

```
setenv SHARC $HOME/sharc/bin   (for c-shell type users)
```

Note that it is advisable to put this line into your shell's login scripts.

2.3.1 Libraries

SHARC requires the BLAS, LAPACK and FFTW3 libraries. During the installation, it might be necessary to alter the **LDFLAGS** string in the **Makefile**, depending on where the relevant libraries are located on your system. In this way, it is for example possible to use vendor-provided libraries like the Intel MKL. For more details see the **INSTALL** file which is included in the SHARC distribution.

2.3.2 Test Suite

After the installation, it is advisable to first execute the test suite of SHARC, which will test the fundamental functionality of SHARC. Change to an empty directory and execute

```
$SHARC/tests.py
```

The interactive script will first verify the Python installation (no message will appear if the Python installation is fine). Subsequently, the script prompts the user to enter which tests should be executed. Their is at least one test for each of the auxiliary scripts and interfaces. Tests whose names start with **scripts_** test the functionality of the auxiliary programs in the SHARC suite. Tests whose names start with **Analytical_**, **COLUMBUS_**, **MOLCAS_** or **MOLPRO_** run short trajectories, testing whether the main dynamics code, the interfaces and the quantum chemistry programs work together correctly.

If the installation was successful and Python is installed correctly, the tests named **scripts_<NAME>** and **Analytical_overlap** should execute without error.

The test calculations involving the quantum chemistry programs (COLUMBUS, MOLCAS, MOLPRO) can be used to check that SHARC can correctly call these programs and that they are installed correctly.

If any of the tests show differences between output and reference output, it is advisable to check the respective files. Note that small differences in the output can already occur when using a different version of the quantum chemistry programs, and that these small differences can add up during the simulations.

2.3.3 Additional Programs

For full functionality of the SHARC suite, several additional programs are recommended:

- The Python package NUMPY.
- The GNUPLOT plotting software.
- A program for molecular visualization, able to read files in the xyz file format (e.g. MOLDEN, GABEDIT, MOLEKEL or VMD)

Optimally, within your Python installation the NUMPY package (which provides many numerical methods, e.g., matrix diagonalization) should be available. If NUMPY is not available, the SHARC suite is still functional, and the affected scripts will fall back to use a small Fortran code (front-end for LAPACK) within the SHARC package. Since in the Python scripts no large-scale matrix calculations are carried out, there should be no significant performance loss if NUMPY is not available.

GNUPLOT is not strictly necessary, since all output files could be plotted using other plotting programs. However, a number of scripts from the SHARC suite automatically generate GNUPLOT scripts after data processing, allowing to quickly plot the output files.

2.3.4 Quantum Chemistry Programs

Even though SHARC comes with an interface for analytical potentials (and hence can be used without any quantum chemistry program), the main application of SHARC is certainly on-the-fly ab initio dynamics. Hence, one of the following interfaced quantum chemistry programs is necessary:

- MOLPRO (this release was checked against MOLPRO 2010 and 2012).
- MOLCAS (this release was checked against MOLCAS 7.8 and 8.0).
  - COLUMBUS 7, interfaced to MOLCAS, for correlated multi-reference wavefunctions.

See the relevant sections in chapter 6 for a description of the quantum chemical methods available with each of these programs.

3 Execution

The SHARC suite consists of the main dynamics code **sharc.x** and a number of auxiliary programs, like setup scripts and analyse tools. Additionally, the suite comes with interfaces to suitable quantum chemistry software, e.g. MOLPRO, MOLCAS or COLUMBUS.

In the following, first it is explained how to run a single trajectory by setting up all necessary input for the dynamics code **sharc.x** manually. Afterwards, the usage of the auxiliary scripts is explained. Detailed infos on the SHARC input files is given in chapter 4 and on the auxiliary scripts in chapter 7. The interfaces are described in chapter 6.

3.1 Running a single trajectory

3.1.1 Input files

SHARC requires a number of input files, which contain the settings for the dynamics simulation (**input**), the initial geometry (**geom**), the initial velocity (**veloc**), the initial coefficients (**coeff**) and the laser field (**laser**). Only the first two (**input**, **geom**) are mandatory, the others are optional. The necessary files are shown in figure 3.1. The content of the main input file is explained in detail in section 4.1, the geometry file is specified in section 4.2. The specifications of the velocity, coefficient and laser files are given in sections 4.3, 4.4 and 4.5, respectively.

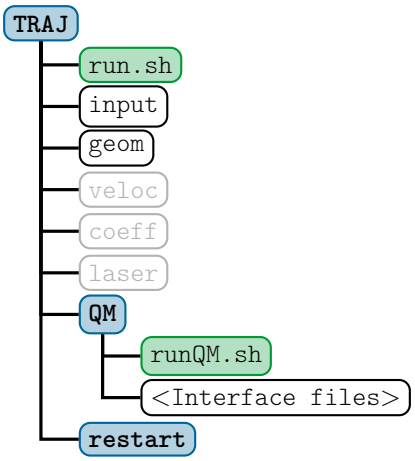

Figure 3.1: Input files for a SHARC dynamics simulation. Directories are in blue, executable scripts in green, regular files in white and optional files in grey.

Additionally, the directory **QM/** and the script **QM/runQM.sh** need to be present, since the on-the-fly ab initio calculations are implemented through these files. The script **QM/runQM.sh** is called each time SHARC performs an on-the-fly calculation of electronic properties (usually by a quantum chemistry program). In order to do so, SHARC first writes the request for the calculation to **QM/QM.in**, then calls **QM/runQM.sh**, waits for the script to finish and then reads the requested quantities from **QM/QM.out**. The script **QM/runQM.sh** is fully responsible to generate the requested results from the provided

input. In virtually all cases, this task is handled by the SHARC-quantum chemistry interfaces (see chapter 6), so that the script **QM/runQM.sh** has a particularly simple form:

```
cd QM/
$SHARC/<INTERFACE> QM.in
```

with the corresponding interface name given. Note that the interfaces in all cases need additional input files, which must be present in **QM/**. Those input files contain the specifications for the quantum chemistry information, e.g., basis set, active and reference space, memory settings, path to the quantum chemistry program, path to scratch directories; or for **SHARC_Analytical.py**, the expressions for the analytical potentials. For each interface, the input files are slightly different. See sections 6.2, 6.3, 6.4 or 6.5 for the necessary information.

3.1.2 Running the dynamics code

Given the necessary input files, SHARC can be started by executing

```
user@host> $SHARC/sharc.x input
```

Note that besides the input file, at least the geometry file needs to be present (see in the input section for details).

3.1.3 Output files

Figure 3.2 shows the content of a trajectory directory after the execution of SHARC. There will be six new files. These files are **output.log**, **output.lis**, **output.dat** and **output.xyz**, as well as **restart.ctrl** and **restart.traj**.

The file **output.log** contains mainly a listing of the chosen options and the resulting dynamics settings. At higher print levels, the log file contains also information per timestep (useful for debugging). **output.lis** contains a table with one line per timestep, giving active states, energies and expectation values. **output.dat** contains a list of all important matrices and vectors at each timestep. This information can be extracted with **data_extractor.x** to yield plottable table files. **output.xyz** contains the geometries of all timesteps (the comments to each geometry give the active state). For details about the content of the output files, see chapter 5.

The restart files contain the full state of a trajectory and its control variables from the last successful timestep. These files are needed in order to restart a trajectory at this timestep (either because the calculation failed, or in order to extend the simulation time beyond the original maximum simulation time).

3.2 Typical workflow for an ensemble of trajectories

Usually, one is not interested in running only a single trajectory, since a single trajectory cannot reproduce the branching of a wavepacket into different reaction channels. In order to do so, within surface hopping an ensemble of independent trajectories is employed.

When dealing with a (possibly large) ensemble of trajectories, setup and analysis need to be automatized. Hence, the SHARC suite contains a number of scripts fulfilling different tasks in the usual

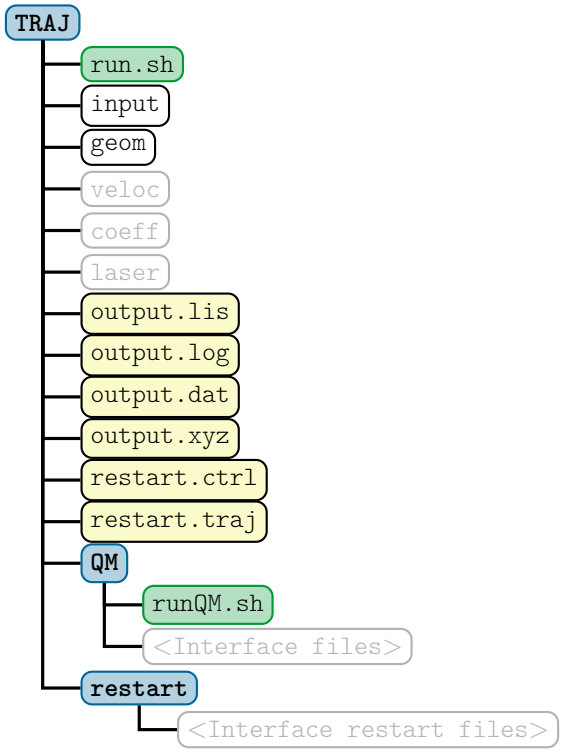

Figure 3.2: Files of a SHARC dynamics simulation after running. Directories are in blue, executable scripts in green, regular files in white and optional files in grey. Output files are in yellow.

workflow of setting up ensembles of trajectories. The typical workflow is given schematically in figure 3.3.

3.2.1 Initial condition generation

In the typical workflow, the user will first create a set of suitable initial conditions. In the context of the SHARC package, an initial condition is a set of an initial geometry, initial velocity, initial occupied state and initial wavefunction coefficients. Many such sets are needed in order to setup physically sound dynamics simulations.

Generation of initial geometries and velocities Currently, within the SHARC suite, initial geometries and velocities can be generated based on a quantum harmonic oscillator Wigner distribution. The theoretical background is given in section 8.12. The calculation is performed by **`wigner.py`**, which is explained in section 7.1.

As given in figure 3.3, **`wigner.py`** needs as input the result of a frequency calculation in MOLDEN format. The calculation can be performed by any quantum chemistry program and any method the user sees fit. **`wigner.py`** produces the file **`initconds`**, which contains a list of initial conditions ready for further processing.

Generation of initial coefficients and states In the second preparation step, for each of the sampled initial geometries it has to be decided which excited state should be the initial one. In simple cases, the user may manually choose the initial excited state using **`excite.py`** (see 7.3). Alternatively, the

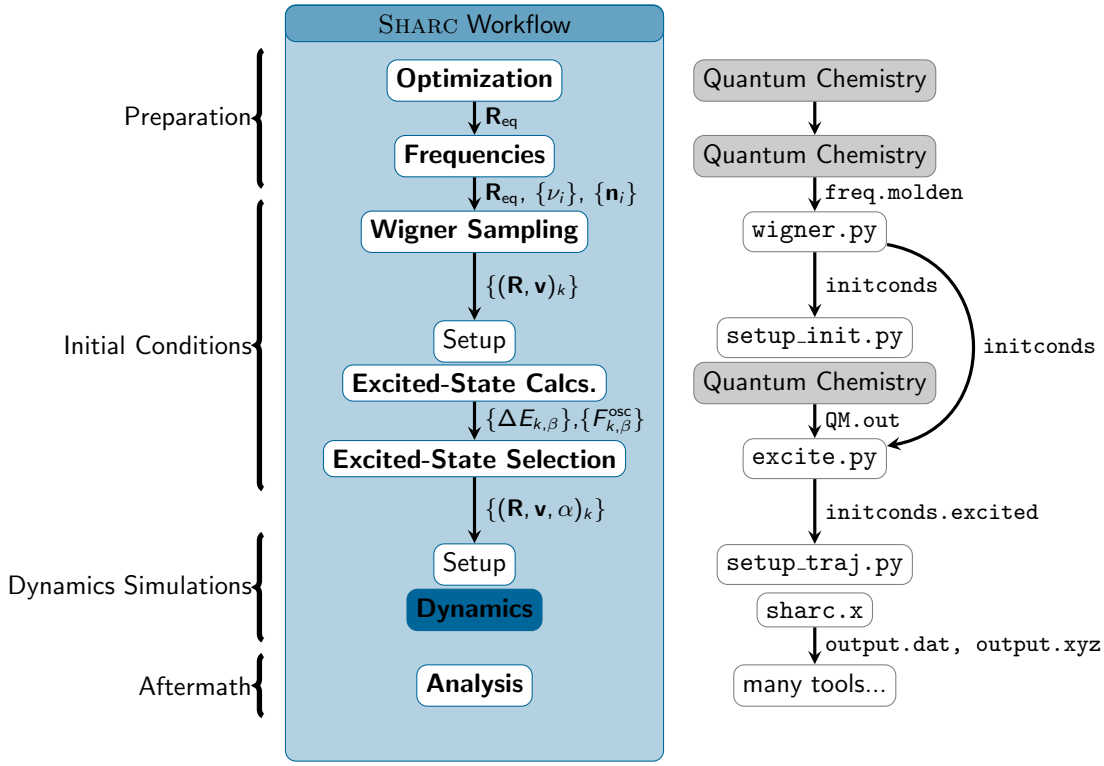

Figure 3.3: Typical workflow for conducting excited-state dynamics simulations with SHARC.

selection of initial states can be performed based on the excitation energies and oscillator strengths of the excited states at each initial geometry (this approximately simulates a delta-pulse excitation).

The latter option makes it necessary to carry out vertical excitation calculation before the selection of the initial states. The calculations can be set up with **`setup_init.py`** (see section 7.2). This script prepares for each initial condition in the **`initconds`** file a directory with the necessary input to perform the calculation. The user should then execute the run script (**`run.sh`**) in each of the directories (either manually or through a batch queueing system).

After the vertical excitation calculations are completed, the vertical excitation energies and oscillator strengths of each calculation are collected by **`excite.py`** (see 7.3). The same script then performs the selection of the initial electronic state for each initial geometry. The results are written to a new file, **`initconds.excited`**. This file contains all information needed to setup the ensemble.

Additionally, **`spectrum.py`** (7.6) can calculate absorption spectra based on the **`initconds.excited`** file. This may be useful to verify that the level of theory chosen is appropriate (e.g., by comparing to an experimental spectrum), or to choose a suitable excitation window for the determination of the initial state.

3.2.2 Running the dynamics simulations

Based on the initial conditions given in **`initconds.excited`**, the input for all trajectories in the ensemble can be setup by **`setup_traj.py`** (see section 7.4). The script produces one directory for each trajectory, containing the input files for SHARC and the selected interface.

In order to run a particular trajectory, the user should execute the run script (**`run.sh`**) in the directory of the trajectory. Since those calculations can run between minutes and several weeks (depending

on the level of theory used and the number of timesteps), it is advisable to submit the run scripts to a batch queueing system.

The progress of the simulations can be monitored most conveniently in the **`output.lis`** files. If the calculations are running in some temporary directory, the output files can be copied to the local directory (where they were setup) with the **`scp`** wrapper **`retrieve.sh`** (see 7.7). This allows to perform ensemble analysis while the trajectories are still running.

If a trajectory fails, the temporary directory where the calculation is running is not deleted. The file **`README`** will be created in the trajectory's directory, giving the time of the failure and the location of the temporary data, so that the case can be investigated.

3.2.3 Analysis of the dynamics results

Each trajectory can be analyzed independently by inspecting the output files (see chapter 5). Most importantly, calling **`data_extractor.x`** (7.8) on the **`output.dat`** file of a trajectory creates a number of formatted files which can be plotted with the help of **`make_gnuscript.py`** (7.9) and GNUPLOT. The nuclear geometries in **`output.xyz`** file can be analyzed in terms of internal coordinates (bond lengths, angles, ring conformations, etc.) using **`geo.py`** (7.12).

For the complete ensemble, the script **`populations.py`** (section 7.10 can calculate average excited-state populations. The script **`crossing.py`** (7.11) can find and extract notable geometries, e.g., those geometries where a surface hop between two particular states occured.

3.3 Auxilliary Programs and Scripts

The following tables list the auxiliary programs in the SHARC suite. The rightmost column gives the section where the program is documented.

3.3.1 Setup

`wigner.py`	Creates initial conditions from Wigner distribution.	7.1
`setup_init.py`	Sets up initial vertical excitation calculations.	7.2
`excite.py`	Generates excited state lists for initial conditions and selects initial states.	7.3
`setup_traj.py`	Sets up the dynamics simulations based on the initial conditions.	7.4
`laser.x`	Prepares files containing laser fields.	7.5
`molpro_input.py`	Prepares MOLPRO input files for common jobs.	6.2.5
`molcas_input.py`	Prepares template files for the MOLCAS interface.	6.3.3
`diagonalizer.x`	Helper program for **`excite.py`**. Only required if NUMPY is not available.	7.13

3.3.2 Analysis

`spectrum.py`	Generates absorption spectra from initial conditions files.	7.6
`retrieve.sh`	**scp** wrapper to retrieve dynamics output during the simulation.	7.7
`data_extractor.x`	Extracts plottable results from the SHARC output data file.	7.8
`make_gnuscript.py`	Creates gnuplot scripts for a given number of states.	7.9
`populations.py`	Calculates ensemble populations.	7.10
`crossing.py`	Extracts specific geometries from ensembles.	7.11
`geo.py`	Calculates internal coordinates from xyz files.	7.12

3.3.3 Interfaces

`SHARC_MOLPRO.py`	Allows for the calculation of SOC, gradients, nonadiabatic couplings, time derivatives, overlaps, dipole moments and angular momentum expectation values at the CASSCF level of theory. Symmetry or RASSCF are not supported. Only segmented basis sets are possible.	6.2
`SHARC_MOLCAS.py`	Allows for the calculation of SOC, gradients, overlaps and dipole moments at the CASSCF and RASSCF level of theory. Symmetry is not supported.	6.3
`SHARC_COLUMBUS.py`	Allows for the calculation of SOC, gradients, overlaps, dipole moments and Dyson norms at the CASSCF, RASSCF, MRCISD and LRT-MRAQCC levels of theory. Symmetry is not supported. Only works with the COLUMBUS-MOLCAS interface.	6.4
`SHARC_Analytical.py`	Allows to calculate SOC, gradients, overlaps, dipole moments and dipole moment gradients based on analytical expressions of diabatic matrix elements. Works only with cartesian coordinates.	6.5

4 Input files

In this chapter, the format of all SHARC input files are presented. Those are the main input file (here called **input**), the geometry file, the velocity file, the coefficients file and the laser file. Only the first two are mandatory, the others are optional input files. All input files are ASCII text files.

4.1 Main input file

This section presents the format and all input keywords for the main SHARC input. Note that when using **setup_traj.py**, full knowledge of the SHARC input keywords is not required.

4.1.1 General remarks

The input file has a relatively flexible structure. With very few exceptions, each single line is independent. An input line starts with a keyword, followed optionally by a number of arguments to this keyword. Example:

```
stepsize 0.5
```

Here, **stepsize** is the keyword, referring to the size of the timesteps for the nuclear motion in the dynamics. **0.5** gives the size of this timestep, in this example 0.5 fs.

A number of keywords have no arguments and act as simple switches (e.g., **restart**, **gradcorrect**, **grad_select**, **nac_select**, **ionization**, **track_phase**, **dipole_gradient**). Those keywords can be prefixed with **no** to explicitly deactivate the option (e.g., **norestart** deactivates restarts).

In each line a trailing comment can be added in the input file, by using the special character **#**. Everything after **#** is ignored by the input parser of SHARC. The input file also can contain arbitrary blank lines and lines containing only comments. All input is case-insensitive.

The input file is read by SHARC by subsequently searching the file for all known keywords. Hence, unknown or misspelled keywords are ignored. Also, the order of the keywords is completely arbitray. Note however, that if a keyword is repeated in the input only the *first* instance is used by the program.

4.1.2 Input keywords

In Table 4.1, all input keywords for the SHARC input file are listed.

Table 4.1: Input keywords. The first column gives the name of the keyword, the second lists possible arguments and the third line provides an explanation. Defaults are marked like **this**. $\$n$ denotes the n-th argument to the keyword.

Keyword	Arguments	Explanation
`printlevel`	**integer** $1=0 $1=1 $1=**2** $1=3 $1=4 $1=5	Controls the verbosity of the log file. Log file is empty + List of internal steps + Input parsing information + Some information per timestep + More information per timestep + Much information per timestep
`restart` `norestart`		Dynamics is resumed from restart files. Dynamics is initialized from input files. `norestart` takes precedence.
`nstates`	list of **integer**s $1 (**1**) $2 (**0**) $3 (**0**) $... (**0**)	Number of states per multiplicity. Number of singlet states Number of doublet states Number of triplet states Number of states of higher multiplicities
`actstates`	list of **integer**s **same as `nstates`**	Number of active states per multiplicity. By default, all states are active.
`state`	**integer**, **string** $1 $2=MCH $2=diag	Specifies the initial state (no default; SHARC exits if **`state`** is missing). Initial state. Initial state and coefficients are given in MCH representation. Initial state and coefficients are given in diagonal representation.
`coeff`	**string** $1=**auto** $1=external	Sets the wavefunction coefficients. Initial coefficient are determined automatically from initial state. Initial coefficients are read from file.
`coefffile`	**quoted string** **"coeff"**	File containing the initial wavefunction coefficients.
`geomfile`	**quoted string** **"geom"**	File name containing the initial geometry.
`veloc`	**string** $1=**zero** $1=random $2 **float** $1=external	Sets the initial velocities. Initial velocities are zero. Initial velocities are determined randomly with $2 eV kinetic energy per atom. Initial velocities are read from file.
`velocfile`	**quoted string** **"veloc"**	File containing the initial velocities.
`laser`	**string** $1=**none**	Sets the laser field. No laser field is applied.

Continued on next page

Table 4.1 – Continued from previous page

Keyword	Arguments	Explanation
	$1=internal	Laser field is calculated at each timestep from internal function.
	$1=external	Laser field for each timestep is read during initialization.
`laserfile`	**quoted string** **"laser"**	File containing the laser field.
`laserwidth`	**float** **1.0 eV**	Laser bandwidth used to detect induced hops.
`stepsize`	**float** **0.5 fs**	Length of the nuclear dynamics timesteps in fs.
`nsubsteps`	**integer** **25**	Number of substeps for the integration of the electronic equation of motion.
`nsteps`	**integer** **3**	Number of simulation steps.
`tmax`	**float**	Total length of the simulation in fs. No effect if **`nsteps`** is present.
`killafter`	**float** **-1**	Terminates the trajectory after $1 fs in the lowest state. If $1<0, trajectories are never killed.
`surf`	**string** $1**=sharc** $1=fish	Chooses the potential energy surfaces used in surface hopping. Uses the diagonal potentials. Uses the MCH potentials.
`coupling`	**string** $1**=ddr** $1=ddt $1=overlap	Chooses the quantities describing the non-adiabatic couplings. Uses vectorial non-adiabatic couplings $\langle\psi_\alpha\|\partial/\partial R\|\psi_\beta\rangle$. Uses temporal non-adiabatic couplings $\langle\psi_\alpha\|\partial/\partial t\|\psi_\beta\rangle$. Uses the overlaps $\langle\psi_\alpha(t_0)\|\psi_\beta(t)\rangle$ (Local Diabatization).
`gradcorrect` **`nogradcorrect`**		Includes $(E_\alpha - E_\beta)\langle\psi_\alpha\|\partial/\partial R\|\psi_\beta\rangle$ in the gradient transformation. Transforms only the gradients itself.
`ekincorrect`	**string** $1=none $1**=parallel_vel** $1=parallel_nac	Adjustment of the kinetic energy after a surface hop. Kinetic energy is not adjusted. Jumps are never frustrated. Velocity is rescaled to adjust kinetic energy. Only the velocity component in the direction of $\langle\psi_\alpha\|\partial/\partial R\|\psi_\beta\rangle$ is rescaled.
`grad_select`		Only some gradients are calculated at every timestep.

Continued on next page

Table 4.1 – Continued from previous page

Keyword	Arguments	Explanation
grad_all		All gradients are calculated at every timestep (Alias: **nograd_select**). **grad_all** takes precedence.
nac_select **nac_all**		Only some $\langle\psi_\alpha\|\partial/\partial R\|\psi_\beta\rangle$ are calculated at every timestep. All $\langle\psi_\alpha\|\partial/\partial R\|\psi_\beta\rangle$ are calculated at every timestep (Alias: **nonac_select**). **nac_all** takes precedence.
eselect	**float** **0.5 eV**	Parameter for selection of gradients and non-adiabatic couplings (in eV).
select_directly		Selection of gradients and NACs in one call.
ezero	**float** **0.0**	Energy shift for the diagonal elements of the Hamiltonian (in hartree). Is not determined automatically.
scaling	**float** **1.0**	Scaling factor for the Hamiltonian matrix and the gradients. 0. <$1
dampeddyn	**float** **1.0**	Scaling factor for the kinetic energy at each timestep. 0. ≤$1≤ 1.
rngseed	**integer** **10997279**	Seed for the random number generator.
decoherence **nodecoherence**		Applies decoherence correction. No decoherence correction. **nodecoherence** takes precedence.
decoherence_param	**float** **0.1**	The value α in the decoherence correction (in Hartree). 0. <$1
ionization **noionization**		Calculates ionization probabilities on-the-fly. No ionization probabilities.
ionization_step	**integer** **1**	Calculates ionization probabilities every $1 timestep. By default calculated every timestep (if at all).
track_phase notrack_phase		Follows the phase of the transformation matrix **U**. No phase following of **U** (not recommended, only for debugging).
dipole_gradient **nodipole_gradient**		Include the derivatives of the dipole moments in the gradients. Neglect the derivatives of the dipole moments.

4.1.3 Detailed Description of the Keywords

Printlevel The **printlevel** keyword controls the verbosity of the log file. The data output file (**output.dat**) and the listing file (**output.lis**) are not affected by the printlevel. The printlevels are described in section 5.1.

Restart There are two keywords controlling trajectory restarting. The keyword **restart** enables restarting, while **norestart** disables restart. If both keywords are present, **norestart** takes precedence. The default is no restart.

When restarting, all control variables are read from the restart file instead of the input file. The only exceptions are **nsteps** and **tmax**. In this way, a trajectory which ran for the full simulation time can easily be restarted to extend the simulation time.

Number of States and Active States The keyword **nstates** controls how many states are taken into account in the dynamics. The keyword arguments specify the number of singlet, doublet, triplet, etc. states. There is no hard-coded maximum multiplicity in the SHARC code, however, some interfaces may restrict the maximum multiplicity.

Using the **actstates** keyword, the dynamics can be restricted to some lowest states in each multiplicity. For each multiplicity, the number of active states must not be larger than the number of states. All couplings between the active states and the frozen states are deleted. These couplings include off-diagonal elements in the H^{MCH} matrix, in the overlap matrix and in the matrix containing the non-adiabatic couplings. Freezing states can be useful if transient absorption spectra are to be calculated without increasing computational cost due to the large number of states.

Note that the initial state must not be frozen.

Initial State The initial state can be given either in MCH or diagonal representation. The keyword **state** is followed by an integer specifying the initial state and either the string **mch** or **diag**. For the MCH representations, states are enumerated according to the canonical state ordering, see 8.16. The diagonal states are ordered according to energy. Note that the initial state must be active.

If the initial state is given in the MCH basis, determination of the initial diagonal state is carried out after the initial QM calculation.

Initial Coefficients The initial coefficients can be read from a file. The default filename is **coeff**, but the filename can be given with the keyword **coefffile**. Note that the filename has to be enclosed in single or double quotes. The file must contain the real and imaginary part of the initial coefficients, one line per state with no blank lines inbetween. These coefficients are interpreted to be in the same representation as the initial state, i.e. the **state** keyword influences the initial coefficients. For details on the file format, see section 4.4.

Alternatively, the initial coefficients can be determined automatically from the initial state, using **coeff auto** in the input file. If the initial state is given in the diagonal representation as i, the initial coefficients are $c_j^{\mathrm{diag}} = \delta_{ij}$. If the initial state is, however, given in the MCH representation, then $c_j^{\mathrm{MCH}} = \delta_{ij}$ and the determination of $\mathbf{c}^{\mathrm{diag}} = \mathbf{U}^{\dagger}\mathbf{c}^{\mathrm{MCH}}$ is carried out after the initial QM calculation.

Geometry Input The initial geometry must be given in a second file in the input format also used byCOLUMBUS. The default name for this file is **geom**. The geometry filename can be given in the input file with the **geomfile** keyword. Note that the filename has to be enclosed in single or double quotes. See section 4.2 for more details.

Velocity Input Using the **veloc** keyword, the initial velocities can be either set to zero, determined randomly or read from a file. Random determination of the velocities is such that each atom has the same kinetic energy, which must be specified after **veloc random** in units of eV. Determination of the random velocities is detailed in 8.10. Note that after the initial velocities are generated, the RNG is reseeded (i.e., the sequence of random numbers in the surface hopping procedure is independent of whether random initial velocities are used).

Alternatively, the initial velocities can be read from a file. The default velocity filename is **veloc**, but the filename can be specified with the **velocfile** keyword. Note that the filename has to be enclosed in single or double quotes. The file must contain the cartesian components of the velocity for each atom on a new line, in the same order as in the geomety file. The velocity is interpreted in terms of atomic units (bohr/atu). See section 4.3 for more details.

Laser Input The keyword **laser** controls whether a laser field is included in the dynamics (influencing the coefficient propagation and the energies/gradients by means of the Stark effect).

The input of an external laser field uses the file **laser**. This file is specified in 4.5.

Laser Width In order to detect laser-induced hops, SHARC compares the instantaneous central laser energy with the energy gap between the old and new states. If the difference between the laser energy and the energy gap is smaller than the laser bandwidth, the hop is classified as laser-induced. Those hops are never frustrated and the kinetic energy is not scaled to preserve total energy (instead, the kinetic energy is preserved).

Simulation Timestep The keyword **stepsize** controls the length of a timestep (in fs) for the dynamics. The nuclear motion is integrated using the Velocity-Verlet algorithm with this timestep. Surface hopping is performed once per timestep and 1–3 quantum chemistry calculations are performed per timestep (depending on the selection schedule). Each timestep is divided in **nsubsteps** substeps for the integration of the electronic equation-of-motion. Since integration is performed in the MCH representation, the default of 25 substeps is usually sufficient, even if very small potential couplings are encountered.

Simulation Time The keyword **nsteps** controls the total length of the simulation. The total simulation time is **nsteps** times **stepsize**. **nsteps** does not include the initial quantum chemistry calculation. Instead of the number of steps the total simulation time can be given directly (in fs) using the keyword **tmax**. In this case, **nsteps** is calculated as **tmax** divided by **stepsize**. If both keywords (**nsteps** and **tmax**) are present, **nsteps** is used.

Using the keyword **killafter**, the dynamics can be terminated before the full simulation time. **killafter** specifies (in fs) the time the trajectory can move in the lowest-energy state before the simulation is terminated. By default, simulations always run to the full simulation time and are not terminated prematurely.

Surface Treatment The keyword **surf** controls whether the dynamics runs on diagonal potential energy surfaces (which makes it a SHARC simulation) or on the MCH PESs (which corresponds to a spin-diabatic dynamics [24] or FISH [23] simulation). Internally, dynamics on the MCH potentials is conducted by setting the U matrix equal to the unity matrix at each timestep.

Description of Non-adiabatic Coupling The code allows to propagate the electronic wavefunction using three different quantities describing non-adiabatic effects, see 8.19. The keyword **coupling** controls which of these quantities is requested from the QM interfaces and used in the propagation. The default is **ddr**, which requires the non-adiabatic coupling vectors $\langle\psi_\alpha|\partial/\partial\mathbf{R}|\psi_\beta\rangle$. For the wavefunction propagation, the scalar product of these vectors and the nuclear velocity is calculated to obtain the matrix $\langle\psi_\alpha|\partial/\partial t|\psi_\beta\rangle$. During the propagation, this matrix is interpolated linearly within each classical timestep.

Alternatively, some interfaces can directly calculate the matrix elements $\langle\psi_\alpha|\partial/\partial t|\psi_\beta\rangle$, which can be used for the propagation. The corresponding argument to **coupling** is **ddt**. In this case, the matrix is taken as constant throughout each classical timestep.

The third possibility is the use of the overlap matrix, requested with **coupling overlaps**. The overlap matrix is used subsequently in the Local Diabatization algorithm for the wavefunction propagation.

Correction to the Gradients The correct transformation of the gradients to the diagonal representation includes contributions from the non-adiabatic coupling vectors. Using **gradcorrect true**, these contributions are included. In this case SHARC will request the calculation of the non-adiabatic coupling vectors, even if they are not used in the wavefunction propagation.

Frustrated Hops and Adjustment of the Kinetic Energy The keyword **ekincorrect** controls how the kinetic energy is adjusted after a surface hop to preserve total energy. **ekincorrect none** deactivates the adjustment, so that the total energy is not preserved after a hop. Using this option, jumps can never be frustrated and are always performed according to the hopping probabilities. Using **ekincorrect parallel_vel**, the kinetic energy is adjusted by simply rescaling the nuclear velocities so that the new kinetic energy is $E_{\text{tot}} - E_{\text{pot}}$. Jumps are frustrated if the new potential energy would exceed the total energy. Finally, using **ekincorrect parallel_nac**, the kinetic energy is adjusted by rescaling the component of the nuclear velocities parallel to the non-adiabatic coupling vector between the old and new state. The hop is frustrated if there is not enough kinetic energy in this direction to conserve total energy. Note that **ekincorrect parallel_nac** implies the calculation of the non-adiabatic coupling vector, even if they are not used for the wavefunction propagation.

Selection of Gradients and Non-Adiabatic Couplings SHARC allows to selectively calculate only certain gradients and non-adiabatic coupling vectors at each timestep. Those gradients and non-adiabatic coupling vectors not selected are not requested from the interfaces, thus decreasing the computational cost. The selection procedure is detailed in 8.15. Selection of gradients is activated by **grad_select**, selection for non-adiabatic couplings by **nac_select**. Selection is turned off by default.

The selection procedure picks only states which are closer in energy to the classically occupied state than a given threshold. The threshold is 0.5 eV by default and can be adjusted using the **eselect** keyword.

By default, if SHARC performs such selection it will do two quantum chemistry calls per timestep. In the first call, all quantities are requested except for the ones to be selected. The energies are used to determine which gradients and NACs to calculate in a second quantum chemistry call. The keyword **select_directly** tells SHARC instead to use the energies of the last timestep, so that only one call per timestep is necessary.

Reference Energy The keyword **ezero** gives the energy shift for the diagonal elements of the Hamiltonian. The shift should be chosen so that the shifted diagonal elements are reasonably

small (large diagonal elements in the Hamiltonian lead to rapidly changing coefficients, requiring extremely short subtimesteps).

Note that the energy shift default is 0, i.e., SHARC does not choose an energy shift based on the energies at the first time step (this would lead to each trajectory having a different energy shift).

Scaling and Damping The scaling factor for the energies and gradients must be positive (not zero), see section 8.13.

The damping factor must be in the interval $[0, 1]$ (first, since the kinetic energy is always positive; second, because a damping factor larger than 1 would lead to exponentially growing kinetic energy). Also see section 8.3.

RNG Seed The RNG seed is used to initialize the random number generator, which provides the random numbers for the surface hopping procedure. For details how the seed is used internally, see section 8.14.

Decoherence The decoherence correction according to Granucci et al. [50] can be applied to the diagonal coefficients by using the keyword **`decoherence`**. By default, no decoherence correction is applied. However, the keyword **`nodecoherence`** can be used to explicitly turn decoherence off.

The keyword **`decoherence_param`** can be used to change the parameter α (see 8.4). The default is 0.1 Hartree, which is the value recommended by Granucci et al. [50].

Ionization The keyword **`ionization`** activates the on-the-fly calculation of ionization transition properties. If the keyword is given, by default these properties are calculated every timestep. The keyword **`ionization_step`** can be used to calculate these properties only every n-th timestep. If the keyword is given, SHARC will request the calculation of the ionization properties from the interface, which needs to be able to calculate them (currently only the COLUMBUS interface allows to perform these calculations in combination with a program which computes the Dyson norms from COLUMBUS output).

Phase Tracking Using the keywords **`track_phase`** and **`no_track_phase`**, the tracking of the eigenvector phases of the transformation matrix $\mathbf{U}$ can be switched on or off. However, it is usually not a good idea to deactivate the phase tracking, since this might lead to spurious behaviour in the wavefunction propagation and the surface hopping.

Dipole Moment Gradients The derivatives of the dipole moments can be included in the gradients. This can be activated with the keyword **`dipole_gradient`**. Currently, only the analytical interface can deliver these quantities.

4.1.4 Example

The following input sample shows a typical input for excited-state dynamics including IC within a singlet manifold plus intersystem crossing to triplet states. It includes a large number of excited singlet states in order to calculate transient absorption spectra. Only the lowest three singlet states actually participate in the dynamics.

```
nstates   8 0 3       # many singlet states for transient absorption
actstates 3 0 3       # only few states to reduce cost

stepsize 0.5          # typical timestep for a molecule containing H
tmax 1000.0           # one ps

surf sharc
state 3 mch                 # start on the S2 singlet state
coeff auto                  # coefficient of S2 will be set to one
coupling ddr                # \
decoherence                 # | typical settings
ekincorrect parallel_vel    # /
grad_select           # \
nac_select            # | improve performance
eselect 0.3           # /

veloc external        # velocity comes from file "veloc"
velocfile "veloc"     #

RNGseed 65435
ezero -399.41494751   # ground state energy of molecule
```

4.2 Geometry file

The geometry file (default file name is **geom**) contains the initial coordinates of all atoms. This file must be present when starting a new trajectory.

It uses the COLUMBUS geometry file format (however,). For each atom, the file contains one line, giving the chemical symbol (a string), the atomic number (a real number), the x, y and z coordinates of the atom in Bohrs (three real numbers), and the relative atomic weight of the atom (a real number). The six items must be separated by spaces. The real numbers are read in using Fortran list-directed I/O, and hence are free format (can have any numbers of decimals, exponential notation, etc.). Element symbols can have at most 2 characters.

The following is an example of a **geom** file for CH_2:

```
C 6.0  0.0 0.0  0.0 12.000
H 1.0  1.7 0.0 -1.2  1.008
H 1.0  1.7 0.0  3.7  1.008
```

4.3 Velocity file

The velocity file (default **veloc**) contains the initial nuclear velocities (e.g., from a Wigner distribution sampling). This file is optional (the velocities can be initialized with the **veloc** input keyword).

The file contains one line of input for each atom, where the order of atoms must be the same as in the **geom** file. Each line consists of three items, separated by spaces, where the first is the x component of the nuclear velocity, followed by the y and z components (three real numbers). The input is interpreted in atomic units (Bohr/atu).

The following is an example of a **veloc** file:

```
 0.0001  0.0000  0.0002
-0.0002  0.0000  0.0012
-0.0003  0.0000 -0.0007
```

4.4 Coefficient file

The coefficient file contains the initial wavefunction coefficients. The file contains one line per state (total number of states, i.e., multiplets count multiple times). Each line specifies the initial coefficient of one state. If the initial state is specified in the MCH representation (input keyword **state**), then the order of the initial coefficients must be as given by the canonical ordering (see section 8.16). If the initial state is given in diagonal representation, then the initial coefficients correspond to the states given in energetic ordering, starting with the lowest state. Each line contains two real numbers, giving first the real and then the imaginary part of the initial coefficient of the respective state.

Example:

```
0.0 0.0
1.0 0.0
0.0 0.0
```

4.5 Laser file

The laser file contains a table with the amplitude of the laser field $\boldsymbol{\epsilon}(t)$ at each timestep of the *electronic* propagation. Given a laser field of the general form:

$$\boldsymbol{\epsilon}(t) = \begin{pmatrix} \Re(\epsilon_x(t)) + \mathrm{i}\Im(\epsilon_x(t)) \\ \Re(\epsilon_y(t)) + \mathrm{i}\Im(\epsilon_y(t)) \\ \Re(\epsilon_z(t)) + \mathrm{i}\Im(\epsilon_z(t)) \end{pmatrix} \tag{4.1}$$

each line consists of 8 elements: t (in fs), $\Re(\epsilon_x(t))$, $\Im(\epsilon_x(t))$, $\Re(\epsilon_y(t))$, $\Im(\epsilon_y(t))$, $\Re(\epsilon_z(t))$, $\Im(\epsilon_z(t))$, (all in atomic units), and finally the instantaneous central frequency (also atomic units).

The timestep in the laser file must exactly match the timestep used for the electronic propagation, which is the timestep used for the nuclear propagation (keyword **`stepsize`**) divided by the number of substeps (keyword **`nsubsteps`**). The first line of the laser file must correspond to t=0 fs.

```
0.00E+00 -0.68E-03  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.31E+00
0.10E-02 -0.77E-02  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.33E+00
0.20E-02 -0.13E-01  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.00E+00  0.35E+00
     ...       ...       ...       ...       ...       ...       ...       ...
```

5 Output files

This chapter documents the content of the output files of SHARC. Those output files are **output.log**, **output.lis**, **output.dat** and **output.xyz**.

5.1 Log file

The log file **output.log** contains general information about all steps of the SHARC simulation, e.g., about the parsing of the input files, results of quantum chemistry calls, internal matrices and vectors, etc. The content of the log file can be controlled with the keyword **printlevel** in the SHARC main input file.

In the following, all printlevels are explained.

Printlevel 0 At printlevel 0, only the execution infos (date, host and working directory at execution start) and build infos (compiler, compile date, building host and working directory) are given.

Printlevel 1 At printlevel 1, also the content of the input file (cleaned of comments and blank lines) is echoed in the log file. Also, the start of each timestep is given.

Printlevel 2 At printlevel 2, the log file also contains information about the parsing of the input files (echoing all enabled options, initial geometry, velocity and coefficients, etc.) and about the initialization of the coefficients after the first quantum chemistry calculation. This printlevel is recommended, since it is the highest printlevel where no output per timestep is written to the log file.

Printlevel 3 This and higher printlevels add output per timestep to the log file. At printlevel 3, the log file contains at each timestep the data from the velocity-Verlet algorithm (old and new acceleration, velocity and geometry), the old and new coefficients, the surface hopping probabilities and random number, the occupancies before and after decoherence correction as well as the kinetic, potential and total energies.

Printlevel 4 At printlevel 4, additionally the log file contains information on the quantum chemistry calls (file names, which quantities were read, gradient and non-adiabatic coupling vector selection) and the propagator matrix.

Printlevel 5 At printlevel 5, additionally the log file contains the results of each quantum chemistry calls (all matrices and vectors), all matrices involved in the propagation as well as the matrices involved in the gradient transformation. This is the highest printlevel currently implemented.

5.2 Listing file

The listing file **`output.lis`** is a tabular summary of the progress of the dynamics simulation. At the end of each timestep (including the initial timestep), one line with 11 elements is printed. These are, from left to right:

1. current step (counting starts at zero for the initial step),
2. current simulation time (fs),
3. current state in the diagonal representation,
4. approximate corresponding MCH state (see subsection 8.11.1),
5. kinetic energy (eV),
6. potential energy (eV),
7. total energy (eV),
8. current gradient norm (in eV/Å),
9. current expectation values of the state dipole moment (Debye),
10. current expectation values of total spin,
11. wallclock time needed for the timestep.

The listing file also contains one extra line for each surface hopping event. For accepted hops, the old and new states (in diagonal representation) and the random number are given. Frustrated hops and resonant hops are also mentioned. Note that the extra line for surface hopping occurs before the regular line for the timestep.

The listing file can be plotted with standard tools like GNUPLOT.

Energies The kinetic energy is calculated at the end of each time step (i.e., after surface hopping events and the corresponding adjustments). The potential energy is the energy of the currently active diagonal state. The total energy is the sum of those two.

Expectation values The gradient norms given in the listing file is calculated as follows:

$$g_{\text{list}} = \sqrt{\frac{1}{3N_{\text{atom}}} \sum_{a}^{N_{\text{atom}}} \sum_{d=x,y,z} g_{ad}^2} \tag{5.1}$$

which is then transformed to eV/Å.

The expectation values of the dipole moment for the active state β is calculated from:

$$\mu = \sqrt{\sum_{p=x,y,z} \left(\sum_{\sigma} \sum_{\tau} \Re \left[U_{\beta\sigma}^{\dagger} \mu_{\sigma\tau}^{p} U_{\tau\beta} \right] \right)^2} \tag{5.2}$$

The expectation value of the total spin of the active state β is calculated as follows:

$$S = \sum_{\alpha} |U_{\alpha\beta}|^2 S_{\alpha} \tag{5.3}$$

where S_α is the total spin of the MCH state with index α.

5.3 Data file

The data file **`output.dat`** contains all relevant data from the simulation for all timesteps, in ASCII format. Accordingly, this file can become quite large for long trajectories or if many states are

included, but for most file systems it is easier to deal with a single large file than with many small files.

Usually, after the simulation is finished the data file is processed by **`data_extractor.x`** to obtain a number of tabular files which can be plotted. For this, see sections 7.13 for the data extractor and 7.9 for plotting.

5.3.1 Specification of the data file

The data file contains a short header followed by the data per timestep. All quantities are commented in the data file.

The header contains:

1. maximum multiplicity,
2. number of states per multiplicity,
3. number of atoms,
4. timestep,
5. energy shift,
6. flag (overlap matrices),
7. flag (laser field),
8. number of steps,
9. number of substeps,
10. full laser field for all substeps (only if flag is set).

The entry for each timestep contains:

1. step index
2. Hamiltonian in MCH representation,
3. transformation matrix **U**,
4. MCH dipole moment matrices (x, y, z),
5. overlap matrix in MCH representation (only if flag is set),
6. coefficients in the diagonal representation,
7. hopping probablities in the diagonal representation
8. kinetic energy,
9. currently active state in diagonal representation and approximate state in MCH representation,
10. random number for surface hopping,
11. wallclock time (in seconds)
12. geometry (Bohrs),
13. velocities (atomic units),
14. property matrix.

5.4 XYZ file

The file **`output.xyz`** contains the geometries of all timesteps in standard xyz file format. It can be used with visualization programs like MOLDEN, GABEDIT or MOLEKEL to create movies of the molecular motion, or with **`geo.py`** (see 7.12) to calculate internal coordinates for each timestep.

The comments of the geometries (given in the second line of each geometry block) contain information about the simulation time and the active state (first in diagonal basis, then in MCH basis).

6 Interfaces

This chapter describes the interface between SHARC and quantum chemistry programs. In the first section, the interface is specified (e.g., for users who attempt to create their own interfaces). The description of the currently existing interfaces takes the remainder of this chapter.

6.1 Interface Specifications

From the SHARC point of view, quantum chemical calculation proceeds as follows in the **QM** directory:

1. write a file called **QM/QM.in**
2. call a script called **QM/runQM.sh**
3. read the output from a file called **QM/QM.out**

For specifications of the formats of these two files (**QM.in** and **QM.out**) see below. The executable script **QM/runQM.sh** must accomplish that all necessary quantum chemical output is available in **QM/QM.out**.

6.1.1 QM.in Specification

The **QM.in** file is written by SHARC every time a quantum chemistry calculation is necessary. It contains all information available to SHARC. This information includes the current geometry (and velocity), the timestep, the number of states, the timestep and the unit used to specify the atomic coordinates. The file also contains *control* keywords and *request* keywords.

The file format is consistent with a standard xyz file. The first line contains the number of atoms, the second line is a comment. SHARC writes the trajectory ID (a hash of all SHARC input files) to this line. The following lines specify the atom positions. As a fourth, fifth and sixth column, these lines may contain the atomic velocities. All following lines contain keywords, one per line and possibly with arguments. Comments can be inserted with '#', and empty lines are permissible. Comments and empty lines are only permissible below the xyz file part. An examplary **QM.in** file is given in the following:

```
3
Jobname
S     0.0     0.0     0.0    0.000  -0.020   0.002
H     0.0     0.9     1.2    0.000  -0.030   0.000
H     0.0    -0.9     1.2    0.000   0.010  -0.000
# This is a comment
Init
States 3 0 2
Unit Angstrom
SOC
DM
```

Table 6.1: Control keywords for SHARC interfaces. These keywords pass information from SHARC to the interface.

Keyword	Description
`init`	Specifies that this is the first calculation. The interface should create a save directory to save all information necessary for a restart.
`samestep`	Specifies that this is an additional calculation at the same geometry/-timestep.
`cleanup`	Specifies that all output files of the interface (except **`QM.out`**) should be deleted (including the save directory).
`unit`	Specifies in which unit the atomic coordinates are to be interpreted. Possible arguments are "angstrom" and "bohr".
`states`	Gives the number of excited states per multiplicity (singlets, doublet, triplets, ...).
`dt`	Gives the time between the last calculation and the current calculation in atomic units.
`savedir`	Gives a path to the directory where the interface should save files needed for restart and between timesteps. If the interface-specific input files also have this keyword, SHARC assumes that the path in **`QM.in`** takes precedence.

```
GRAD 1 2
OVERLAP
NACDR select
  1 2
  end
```

There exist two types of keywords, *control* keywords and *request* keywords. Control keywords pass some information to the interface. Request keywords tell the interface to provide a quantity in the **`QM.out`** file. Table 6.1 contains all control keywords while table 6.2 lists all request keywords.

6.1.2 `QM.out` Specification

The **`QM.out`** file communicates back the results of the quantum chemistry calculation to the dynamics code. After SHARC called **`QM/runQM.sh`**, it expects that the file **`QM/QM.out`** exists and contains the relevant data.

The following quantities are expected in the file (depending whether the corresponding keyword is in the **`QM.in`** file): Hamiltonian matrix, dipole matrices, gradients, non-adiabatic couplings (either NACDR or NACDT), overlaps, wavefunction phases, property matrices. The format of **`QM.out`** is described in the following.

Each quantity is given as a data block, which has a fixed format. The order of the blocks is arbitrary, and between blocks arbitrary lines can be written. However, within a block no extraneous lines are allowed. Each data block starts with a exclamation mark `!`, followed by whitespace and an integer flag which specifies the type of data:

Table 6.2: Request keywords for SHARC interfaces.

Keyword	Description
`H`	Calculate the molecular Hamiltonian (diagonal matrix with the energies of the states of the model space).
`SOC`	Calculate the molecular Hamiltonian including the SOCs (not diagonal anymore within the model space).
`DM`	Calculate the state dipole moments and transition dipole moments between all states.
`ION`	Calculate transition properties between neutral and ionic wavefunctions.
`GRAD`	Calculate gradients for all states. If followed by a list of states, calculate only gradients for the specified states.
`NACDT`	Calculate the time-derivatives $\langle\Psi_1\|\partial/\partial t\|\Psi_2\rangle$ by finite differences between the last timestep and the current timestep
`NACDR`	Calculate non-adiabatic coupling vectors $\langle\Psi_1\|\partial/\partial\mathbf{R}\|\Psi_2\rangle$ between all pairs of states. If followed by "select", read the list of pairs on the following lines until "end" and calculate non-adiabatic coupling vectors between the specified pairs of states.
`OVERLAP`	Calculate overlaps $\langle\Psi_1(t_0)\|\Psi_2(t)\rangle$ between all pairs of states (between the last and current timestep). If followed by "select", read the list of pairs on the following lines until "end" and calculate overlaps between the specified pairs of states.
`DMDR`	Calculate the cartesian gradients of the dipole moments and transition dipole moments of all states.

1 Hamiltonian matrix
2 Dipole matrices
3 Gradients
4 Non-adiabatic couplings (NACDT)
5 Non-adiabatic couplings (NACDR)
6 Overlap matrix
7 Wavefunction phases(optional)
8 Wallclock time for QM calculation (optional)
11 Property matrix (e.g. ionization probabilities)
12 Dipole moment gradients

On the next line, two integers are expected giving the dimensions of the following matrix. Note, that all these matrices must be square matrices. On the following lines, the matrix or vector follows. Matrices are in general complex, and real and imaginary part of each element is given as a pair of floating point numbers.

The following shows an example of a 4×4 Hamiltonian matrix. Note that the imaginary parts directly follow the real parts (in this example, the Hamiltonian is real).

```
 ! 1
 4 4
-548.6488 0.0000    0.0000 0.0000    0.0003 0.0000    0.0003 0.0000
   0.0000 0.0000 -548.6170 0.0000    0.0003 0.0000    0.0003 0.0000
   0.0003 0.0000    0.0003 0.0000 -548.5986 0.0000    0.0000 0.0000
   0.0003 0.0000    0.0003 0.0000    0.0000 0.0000 -548.5912 0.0000
```

The three dipole moment matrices (x, y and z polarization) must follow directly after each other, where the dimension specifier must be present for each matrix. The dipole matrices are also expected to be complex-valued.

```
 ! 2
 2 2
 0.1320 0.0000 -0.0020 0.0000
-0.0020 0.0000 -1.1412 0.0000
 2 2
 0.0000 0.0000  0.0000 0.0000
 0.0000 0.0000  0.0000 0.0000
 2 2
 2.1828 0.0000  0.0000 0.0000
 0.0000 0.0000  0.6422 0.0000
```

Gradient and non-adiabatic couplings vectors are written as $3 \times n_{\text{atom}}$ matrices, with the x, y and z components of one atom per line. These vectors are expected to be real valued. Each vector is preceeded by its dimensions.

```
6 3
 0.0000 -6.5429 -8.1187
 0.0000  5.8586  8.0160
 0.0000  6.8428  1.0265
 0.0000  6.5429  8.1187
 0.0000 -5.8586 -8.0160
 0.0000 -6.8428 -1.0265
```

If gradients are requested, SHARC expects every gradient to be present, even if only some gradients are requested. The gradients are expected in the canonical ordering (see section 8.16), which implies that for higher multiplets the same gradient has to be present several times. For example, with 3 singlets and 3 triplets, SHARC expects 12 gradients in the **QM.out** file (each triplet has three components with M_s = -1, 0 or 1).

Similarly, for non-adiabatic coupling vectors, SHARC expects all pairs, even between states of different multiplicity. The vectors are also in canonical ordering, where the inner loop goes over the ket states. For example, with 3 singlets and 3 triplets (12 states), SHARC expects 144 (12^2) non-adiabatic coupling vectors in the **QM.out** file.

```
! 5 Non-adiabatic couplings (ddr) (2x2x1x3, real)
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 1 ms2 0
 0.0e+0  0.0e+0  0.0e+0
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 2 ms2 0
+2.0e+0  0.0e+0  0.0e+0
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 1 ms2 0
-2.0e+0  0.0e+0  0.0e+0
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 2 ms2 0
 0.0e+0  0.0e+0  0.0e+0
```

The non-adiabatic coupling matrix (NACDT keyword), the overlap matrix and the property matrix are single $n \times n$ matrices (n is the total number of states), respectively, like the Hamiltonian.

The wavefunction phases are a vector of complex numbers.

The wallclock time is a single real number.

The dipole moment gradients are a list of $3 \times n_{\text{atom}}$ vectors, each specifying the gradient of one polarization of one dipole moment matrix element. In the outmost loop, the bra index is counted, then the ket index, then the polarization. Hence, the respective entry in **QM.out** would look like (for 2 states and 1 atom):

```
! 12 Dipole moment derivatives (2x2x3x1x3, real)
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 1 ms2 0   pol 0
```

```
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 1 ms2 0   pol 1
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 1 ms2 0   pol 2
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 2 ms2 0   pol 0
 1.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 2 ms2 0   pol 1
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 1 ms1 0   m2 1 s2 2 ms2 0   pol 2
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 1 ms2 0   pol 0
 1.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 1 ms2 0   pol 1
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 1 ms2 0   pol 2
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 2 ms2 0   pol 0
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 2 ms2 0   pol 1
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
1 3 ! m1 1 s1 2 ms1 0   m2 1 s2 2 ms2 0   pol 2
 0.000000000000E+000  0.000000000000E+000  0.000000000000E+000
```

6.1.3 Further Specifications

The interfaces may require additional input files beyond **QM.in**, which contain static information. This may include paths to the executable quantum chemistry program, paths to scratch directories or input templates for the quantum chemistry calculation (e.g. active space specifications, basis sets, etc.). The dynamics code does not depend on these additional files.

6.1.4 Save Directory Specification

The interfaces must be able to save all information necessary for restart to a given directory. The absolute path is written to **QM.in** by SHARC. Hence, for the trajectories the path to the save directory is always a subdirectory of the working directory of SHARC.

Table 6.3: Keywords for the **SH2PRO.inp** file.

Keyword	Description
molpro	Is followed by a string giving the path to the MOLPRO executable. Relative and absolute paths, environment variables and ~ can be used.
scratchdir	Is a path to the temporary directory. Relative and absolute paths, environment variables and ~ can be used. If the directory does not exist, the interface will create it. In any case, the interface will delete this directory after the calculation.
gradaccudefault	(float) Default accuracy for CP-MCSCF.
gradaccumax	(float) Worst acceptable accuracy for CP-MCSCF (see below).
checknacs	(boolean) Use the MRCI overlaps to check whether the time derivatives and overlap matrices are correct.
correctnacs	(boolean) Replace bad time derivatives with the scalar product of non-adiabatic coupling vectors and velocities.
checknacs_mrcio	(float) Threshold for the MRCI overlaps to be considered bad.
checknacs_ediff	(float, eV) If MRCI overlaps indicate inaccurate couplings, set to zero couplings between states which are farther apart than this value.

6.2 MOLPRO Interface

The SHARC-MOLPRO interface allows to run SHARC dynamics with MOLPRO's CASSCF wavefunctions. RASSCF is not supported, since on RASSCF level state-averaging over different multiplicities is not possible. The interface uses MOLPRO's CI program in order to calculate transition dipole moments and spin-orbit couplings. Gradients and non-adiabatic coupling vectors are calculated using MOLPRO's CADPACK code (hence no generally contracted basis sets can be used). Time derivatives and overlaps are obtained with the DDR prodecure. Hence, all types of couplings usable by SHARC are available through this interface. The interface also allows to calculate the expectation value of the angular momentum operators.

The SHARC-MOLPRO interface needs two additional input files, which should be present in **QM/**. Those input files are **SH2PRO.inp**, which contains the paths to MOLPRO and the scratch directory, and **MOLPRO.template**, which is a minimal input from which the full input can be built. If **QM/wf.init** is present, it will be used as a MOLPRO wavefunction file containing the initial MOs.

6.2.1 Interface specific input file: SH2PRO.inp

The interface requires some additional information beyond the content of **QM.in**. This information is given in the file **SH2PRO.inp**, which must reside in the directory where the interface is started. This file uses a simple "**keyword argument**" syntax. Comments using # and blank lines are possible, the order of keywords is arbitrary. Lines with unknown keywords are ignored, since the interface just searches the file for certain keywords.

Table 6.3 lists the existing keywords.

Note that only the first two keywords are mandatory. The last four keywords are only used for calculations with time derivatives or overlaps. The checking and correction of these couplings has to be considered experimental. Normally, **checknacs** should be "False" (in this case the other three keywords are not necessary).

6.2.2 Template file: `MOLPRO.template`

The template file is a MOLPRO input file specifying a state-averaged CASSCF calculation. There must not be any **`file`** specifications and no geometry input in this file. It should contain memory specifications, basis set, and optionally settings like Douglas-Kroll.

Most importantly, it has to contain a CASSCF input block with the number of frozen, closed-shell and occupied orbitals. Additionally, all wavefunctions for the state-averaging have to be defined. For each multiplicity, the **`wf`**, **`state`** and **`weight`** keywords must be present. The following shows an example input for CASSCF(12,9) with 4 singlets and 3 triplets in the state-averaging.

```
***,Example
memory,100,M

dkroll=1
dkho=2
basis=6-31G*

{casscf
frozen,0
closed,23
occ,32
wf,58,1,0
state,4
weight,1,1,1,1
wf,58,1,2
state,3
weight,1,1,1
};
```

6.2.3 Error checking

The interface is written such that the output of MOLPRO is checked for commonly occuring errors, mostly bad convergence in the MCSCF or CP-MCSCF parts. In these cases, the input is adjusted and MOLPRO restarted. This will be done until all calculations are finished or an unrecoverable error is detected. The interface will try to solve the following error messages:

EXCESSIVE GRADIENT IN CI This error message can occur in the MCSCF part. The calculation is restarted with a P-space threshold (see MOLPRO manual) of 1. If the error remains, the threshold is quadrupled until the calculation converges or the threshold is above 100.

NO CONVERGENCE IN REFERENCE CI The error occurs in the CI part. The calculation is restarted with a P-space threshold (see MOLPRO manual) of 1. If the error remains, the threshold is quadrupled until the calculation converges or the threshold is above 100.

NO CONVERGENCE OF CP-MCSCF This error occurs when solving the linear equations needed for the calculation of MCSCF gradients or non-adiabatic coupling vectors. In this case, the interface finds in the output the value of closest convergence and restarts the calculation with the value found as the new convergence criterion. This ensures that the CP-MCSCF calculation converges, albeit with lower accuracy for this gradient for this timestep.

This error check is controlled by two keywords in the **SH2PRO.inp** file. The interface first tries to converge the CP-MCSCF calculation to **gradaccudefault**. If this fails, it tries to converge to the best value possible within 900 iterations. **gradaccumax** defines the worst accuracy accepted by the interface. If a CP-MCSCF calculation cannot be converged below **gradaccumax** then the interface exits with an error, leading to the abortion of the trajectory.

6.2.4 Things to keep in mind

Initial orbital guess For CASSCF calculations it is always a good idea to start from converged MOs from a nearby geometry. For the first timestep, if a file **QM/wf.init** is present, the SHARC-MOLPRO interface will take this file for the starting orbitals. In subsequent calculations, the MOs from the previous step will be used.

CP-MCSCF calculations vs. DDR calculations In the current version of the interface, the keywords **grad** and **nacdr** cannot be combined with the keywords **nacdt** and **overlap**. This is related to the way the interface allocates the records for gradients in the wavefunction file of **Molpro**. This issue might be resolved in future releases.

This has the consequence that for trajectories using time derivatives or overlaps the gradient selection *must* be activated (though the selection criterion can be chosen arbitrarily high). Also, the **select_directly** keyword is not compatible with time derivatives and overlaps. **setup_traj.py** automatically takes care of this constraints.

Basis sets Note that **Molpro** cannot calculation SA-CASSCF gradients for generally contracted basis sets (like Dunning's "cc" basis sets or Roos' "ANO" basis sets). Only segmented basis sets are allowed, like the Pople basis sets and the "def" family from TURBOMOLE.

6.2.5 Molpro input generator: molpro_input.py

In order to quickly setup simple calculations using MOLPRO, the SHARC suite contains a small script called **molpro_input.py**. It can be used to setup singlepoint calculations, optimizations and frequency calculations on the HF, DFT, MP2 and CASSCF level of theory. Of course, MOLPRO has far more capabilities, but these are not covered by **molpro_input.py**. However, **molpro_input.py** can also prepare template files which are compatible with the SHARC-Molpro interface.

The script interactively asks the user to specify the calculation and afterwards writes an input file and optionally a run script.

Input

Type of calculation Choose to either perform a single-point calculation or an optimization (including optionally frequency calculation), or to generate a template file. In the latter case, no geometry file is needed. The script looks for a **MOLPRO.input** in the same directory and allows to copy the settings.

For single-point calculations, optimizations and frequency calculations, files in MOLDEN format called **geom.molden**, **opt.molden** or **freq.molden**, respectively, are created (containing the orbitals, optimization steps and normal modes, respectively). The file **freq.molden** can be used to generate initial conditions with **wigner.py**.

Geometry Specify the geometry file in xyz format. Number of atoms and total nuclear charge is detected automatically. After the user inputs the total charge, the number of electrons is calculated automatically.

In the case of the generation of a template file, instead only the number of electrons is required.

Non-default atomic masses If a frequency calculation is requested, the user may modify the mass of specific atoms (e.g. to investigate isotopic effects). In the following menu, the user can add or remove atoms with their mass to a list containing all atoms with non-default masses. Each atom is referred to by its number as in the geometry file. Using the command **show** the user can display the list of atoms with non-default masses. Typing **end** confirms the list.

Note that when using the produced MOLDEN file later with **wigner.py**, the user has to enter the same non-default masses again, since the MOLDEN file does not contain the masses and **wigner.py** has no way to retrieve these numbers.

Level of theory Supported are Hartree-Fock (HF), Density Functional Theory (DFT), Møller-Plesset perturbation theory (MP2) and CASSCF (either single-state or state-averaged). All methods are compatible with odd-electron wavefunctions (**molpro_input.py** will use the corresponding UHF, UMP2 and UKS keywords in the input file, if necessary).

For template generation state-average CASSCF is automatically chosen. All methods can be combined with optimizations and frequency calculations, however, the frequency calculation is much more efficient with HF or SS-CASSCF.

DFT functional For DFT calculations, enter a functional and choose whether dispersion correction should be applied. Note that the functional is just a string which is not checked by **molpro_input.py**.

Basis set The basis set is just a string which is not checked by **molpro_input.py**.

CASSCF settings For CASSCF calculations, enter the number of active electrons and orbitals.

For SS-CASSCF, only the multiplicity needs to be specified. For SA-CASSCF, specify the number of states per multiplicity to be included. Note that MOLPRO allows to average over states with different numbers of electrons. This feature is not supported in **molpro_input.py**. However, the user can generate a closely-matching input and simply add the missing states to the CASSCF block manually.

For optimizations at SA-CASSCF level, the state to be optimized has to be given.

Memory Enter the amount of memory for MOLPRO. Note that values smaller than 50 MB are ignored, and 50 MB are used in this case.

Table 6.4: Keywords for the **SH2CAS.inp** file.

Keyword	Description
molcas	Is the path to the MOLCAS installation. Relative and absolute paths, environment variables and ~ can be used. The interface will set **$MOLCAS** to this path. If this keyword is not present in **SH2CAS.inp**, the interface will use the environment variable **$MOLCAS**, if it is set.
scratchdir	Is a path to the temporary directory. Relative and absolute paths, environment variables and ~ can be used. If it does not exist, the interface will create it. In any case, the interface will delete this directory after the calculation.
memory	The memory usable by MOLCAS. The interface will set **$MOLCASMEM** to this value. The default is 10 MB.
project	Can be used to set a MOLCAS project name (**$Project**). If not used, the interface will generate a default project name.

Run script If requested, the script also generates a simple Bash script (**run_molpro.sh**) to directly execute MOLPRO. The user has to enter the path to MOLPRO and the path to a suitable (fast) scratch directory.

Note that the scratch directory will be deleted after the calculation, only the wavefunction file **wf** will be copied back to the main directory.

6.3 MOLCAS Interface

The SHARC-MOLCAS interface can be used to conduct excited-state dynamics based on MOLCAS' CASSCF wavefunctions. RASSCF wavefunctions are not supported currently. The interface uses the modules GATEWAY, SEWARD (integrals), RASSCF (wavefunction, energies), RASSI (transition dipole moments, spin-orbit couplings, overlaps), MCLR and ALASKA (gradients). Since MOLCAS is not able to calculate the full non-adiabatic coupling vectors, only wavefunction overlaps are currently implemented in the interface.

The interface needs two additional input files, a template file for the quantum chemistry (file name is **MOLCAS.template**) and a general input file (**SH2CAS.inp**). If **QM/MOLCAS.<i>.JobIph.old** files are present, they are used as initial wavefunction files, where **<i>** is the multiplicity.

6.3.1 Interface specific input file: SH2CAS.inp

The file **SH2CAS.inp** contains mainly paths (to the MOLCAS executables, to the scratch directory, etc.). This file must reside in the same directory where the interface is started. It uses a simple "**keyword argument**" syntax. Comments using # and blank lines are possible, the order of keywords is arbitrary. Lines with unknown keywords are ignored, since the interface just searches the file for certain keywords.

Table 6.4 lists the existing keywords.

Note that the interface sets all environment variables necessary to run MOLCAS (e.g., **$MOLCAS**, **$MOLCASMEM**, **$WorkDir** ,**$Project**) automatically, based on the paths to MOLCAS and the scratch directory from **SH2CAS.inp**.

Table 6.5: Keywords for the **MOLACS.template** file.

Keyword	Description
basis	The basis set used. Note that some basis sets (e.g., Pople basis sets) do not work, since the spin-orbit integrals cannot be calculated.
ras2	Number of active orbitals for CASSCF.
nactel	Number of active electrons for CASSCF.
inactive	Number of inactive orbitals.
spin	The full line reads as **spin s roots r** and specifies the number of roots for the given multiplicity. This influences only the number of states in the state-averaging procedure. The number of states in the dynamics must not be larger than the number of states given here.

6.3.2 Template file: MOLCAS.template

This file contains the specifications for the wavefunction. Note that this is not a valid MOLCAS input file. No sections like **$GATEWAY**, etc., can be used. The file only contains a number of keywords, given in table 6.5.

The template file can be setup with the tool **molcas_input.py**.

6.3.3 Template file generator: molcas_input.py

This is a small interactive script to generate template files for the SHARC-MOLCAS interface. It simply queries the user for some input parameters and then writes the file **SH2CAS.inp**, which can be used to run SHARC simulations with the SHARC-MOLCAS interface.

Geometry file The geometry file is only used to calculate the nuclear charge.

Charge This is the overall charge of the molecule. This number is used with the nuclear charge to calculate the number of electrons.

Basis set This is simply a string, which is *not* checked by the script to be a valid basis set of the MOLCAS library.

Number of active electrons and orbitals These settings are necessary for the definition of the CASSCF wavefunction. The number of inactive orbitals is automatically calculated from the total number of electrons and the number of active electrons.

States for state-averaging For each multiplicity, the number of states for the state-averaging procedure must be equal or larger than the number of states used in the dynamics.

6.4 COLUMBUS Interface

The SHARC-COLUMBUS interface allows to run SHARC dynamics based on COLUMBUS' CASSCF, RASSCF, MRCI and LRT-MRAQCC wavefunctions. The interface is compatible to COLUMBUS calculations utilizing the COLUMBUS-MOLCAS interface only (SEWARD integrals and ALASKA gradients). Unfortunately,

time derivatives and non-adiabatic coupling vectors cannot be calculated with this setup, only wavefunction overlaps can be obtained. The interface utilizes the **cioverlap** program by Jiri Pittner [27] to calculate the overlap matrices needed for local diabatization propagation. The interface can also calculate Dyson norms between neutral and ionic wavefunctions using a suitable user-supplied code.

The interface needs as additional input the file **QM/SH2COL.inp** and a template directory containing all input files needed for the COLUMBUS calculations. Additionally, initial MOs can be given in the file **QM/mocoef_mc.init**.

6.4.1 Template input

The interface does not generate the full COLUMBUS input on-the-fly. Instead, the interface uses an existing set of input files and performs only necessary modifications (e.g., the number of states). The set of input files must be provided by the user. Please see the COLUMBUS online documentation and, most importantly, the COLUMBUS SOCI tutorial for a documentation of the necessary input.

Generally, the input consists of a directory with one subdirectory with input for each multiplicity (singlets, doublets, triplets, ...). However, even-electron wavefunctions of different multiplicities can be computed together in the same job if spin-orbit couplings are desired. Otherwise independent multiple-DRT inputs (ISC keyword) are also acceptable. Note that symmetry is not allowed when using the interface.

The path to the template directory must be given in **SH2COL.inp**.

Integral input The interface is only able to use input for calculations using MOLPRO/SEWARD integrals. Also make sure that you include scalar-relativistic (Douglas-Kroll-Hess) and spin-orbit (AMFI) effects in the integral input.

It is important to make sure that **the order of atoms** in the template input files and in the SHARC input **is consistent**.

MCSCF input The MCSCF section can use any desired state-averaging scheme. However, frozen core orbitals in the MCSCF step are not possible (since otherwise gradients cannot be computed). Prepare the MCSCF input for CI gradients. It is advisable to use very tight MCSCF convergence criteria.

MRCI input Either prepare a single-DRT input without SOCI (to cover a single multiplicity), a single-DRT input with SOCI and a sufficient maximum multiplicity for spin-orbit couplings or a independent multiple-DRT input (ISC case). Make sure that all multiplicities are covered with all input directories.

In the MRCI input, make sure to use sequential **ciudg**. Also take care to setup gradient input on MRCI level.

Job control Setup a single-point calculation with the following steps:

- SCF
- MCSCF
- MR-CISD (serial operation) or SO-CI coupled to non-rel CI (for SOCI DRT inputs)
- one-electron properties for all methods
- transition moments for MR-CISD

- nonadiabatic couplings (and/or gradients)

Request first transition moments and interstate couplings in the following dialogues. Analysis in internal coordinates and intersection slope analysis are not required.

6.4.2 Interface specific input file: `SH2COL.inp`

Beyond the information from **`QM.in`** the interface needs additional input, which is read from the file **`SH2COL.inp`**. Table 6.6 lists the keywords which can be given in the file.

6.4.3 Template setup

The template directory contains several subdirectories with input for different multiplicities. An example is given in figure 6.1. In **`SH2COL.inp`**, the user has to associate each multiplicity to a subdirectory. The line "**`DIR 1 Sing_Trip`**" would make the interface use the input files from the subdirectory **`Sing_Trip`** when calculating singlet states (the **`1`** refers to singlet calculations). All calculations using a particular input subdirectory are called a job.

Additionally, the user must specify which job(s) provide the MO coefficients (e.g., the calculation for doublet states could be based on the same MOs as the singlet and triplet calculation). The line "**`MOCOEF Doub_Quar Sing_Trip`**" would tell the interface to do a MCSCF calculation in the **`Sing_Trip`** job, and reuse the MOs when doing the **`Doub_Quar`** job without reoptimizing the MOs.

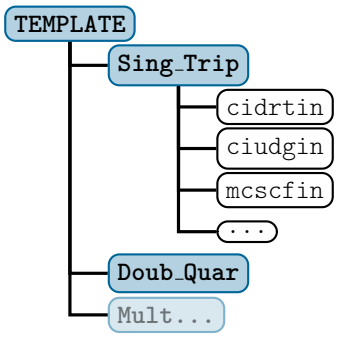

Figure 6.1: Example directory structure of the COLUMBUS template directory

6.5 Analytical PESs Interface

The SHARC suite also contains an interface which allows to run dynamics simulations on PESs expressed with analytical functions. In order to allow dynamics simulations in the same way as it is done on-the-fly, the interface uses two kinds of potential couplings:

- couplings that are pre-diagonalized in the interface (yielding the equivalent of the MCH basis),
- couplings given by the interface to SHARC as off-diagonal elements.

The interface needs one additional input file, called **`SH2Ana.inp`**, which contains the definitions of all analytical expressions.

Table 6.6: Keywords for the **SH2COL.inp** file.

Keyword	Description
columbus	Is followed by a string giving the path to the COLUMBUS main directory. Relative and absolute paths, environment variables and ~ can be used.
molcas	Is followed by a string giving the path to the MOLCAS main directory. Relative and absolute paths, environment variables and ~ can be used. This path is only used for overlap calculations (since in this case the interface calls MOLCAS explicitly). Otherwise COLUMBUS will use the path to MOLCAS specified during the installation of COLUMBUS.
scratchdir	Is a path to the temporary directory. Relative and absolute paths, environment variables and ~ can be used. If it does not exist, the interface will create it. In any case, the interface will delete this directory after the calculation.
savedir	Is a path to another temporary directory. Relative and absolute paths, environment variables and ~ can be used. The interface will store files needed for restart there.
dyson	Path to the dyson code. Relative and absolute paths, environment variables and ~ can be used. Only necessary if Dyson norms are calculated. Needs a suitable code that computes Dyson norms.
civecconsolidate	Path to the civecconsolidate code suitable for preparation of input for the Dyson program (CI vectors in terms of determinants). Relative and absolute paths, environment variables and ~ can be used. Only used if Dyson norms are calculated. If not present, the civecconsolidate code in the **$COLUMBUS** directory is used. Note that in the current version of COLUMBUS (Oct. 2014) civecconsolidate is not compatible with the calculation of Dyson norms.
memory	(integer, MB) Memory for COLUMBUS. Note that the maximum amount of memory used by cioverlaps cannot be controlled with this keyword.
ciothres	(float) Threshold for including a pair of determinants in the cioverlaps program.
dysonthres	(float) Threshold for including a pair of determinants in the calculation of Dyson norms.
nooverlap	(no argument) Do not keep the binary files necessary to calculate wavefunction overlaps in the next timestep.
template	Is followed by the path to the directory containing the template subdirectories. Relative and absolute paths, environment variables and ~ can be used.
DIR	See 6.4.3.
MOCOEF	See 6.4.3.

6.5.1 Parametrization

The interface has to be provided with analytical expressions for all matrix elements of the following matrices in the diabatic basis:

- Hamiltonian: $\mathbf{H}$
- Derivative of the Hamiltonian with respect to each atomic coordinate: $\mathbf{H}_{x_i}$
- (Transition) dipole matrices for each polarization direction: $\mathbf{M}_i$
- Real and imaginary part of the SOC matrix: $\mathbf{\Sigma}$
- (Optionally) the derivatives of the transition dipole matrices: $\mathbf{D}_{i,x_j}$

The diabatic Hamiltonian is diagonalized:

$$\mathbf{H}^{\mathrm{d}} = \mathbf{W}^{\dagger}\mathbf{H}\mathbf{W} \tag{6.1}$$

Then the following calculations lead to the MCH matrices which are passed to SHARC:

$$\mathbf{H}^{\mathrm{MCH}} = \mathbf{H}^{\mathrm{d}} + \mathbf{W}^{\dagger}\mathbf{\Sigma}\mathbf{W} \tag{6.2}$$

$$\left(\mathbf{g}_{\alpha}^{\mathrm{MCH}}\right)_{x_i} = \left(\mathbf{W}^{\dagger}\mathbf{H}_{x_i}\mathbf{W}\right)_{\alpha\alpha} \tag{6.3}$$

$$\mathbf{M}_i^{\mathrm{MCH}} = \mathbf{W}^{\dagger}\mathbf{M}_i\mathbf{W} \tag{6.4}$$

$$\mathbf{S}^{\mathrm{MCH}}(t_0, t) = \mathbf{W}^{\dagger}(t_0)\mathbf{W}(t) \tag{6.5}$$

$$\mathbf{D}_{i,x_j}^{\mathrm{MCH}} = \mathbf{W}^{\dagger}\mathbf{D}_{i,x_j}\mathbf{W} \tag{6.6}$$

The MCH Hamiltonian is the diagonalized diabatic Hamiltonian plus the SO matrix transformed into the MCH basis. The gradients in the MCH basis are obtained by transforming the derivative matrices into the MCH basis. The dipole matrices are also simply transformed into the MCH basis. The overlap matrix is the overlap of old and new transformation matrix.

6.5.2 Interface-specific input file

The interface-specific input file is called **`SH2Ana.inp`**. It contains the analytical expressions for all matrix elements mentioned above. All analytical expressions in this file are evaluated considering the atomic coordinates read from **`QM.in`**.

The file consists of a file header and the file body. The file body consists of variable definition blocks and matrix blocks.

Header The header looks similar to an xyz file:

```
2
2
I       xI      0       0
Br      xBr     0       0
```

Here, the first line gives the number of atoms and the second line the number of states.

On the remaining lines, each cartesian component of the atomic coordinates is associated to a variable name, which can be used in the analytical expressions. If a zero (**`0`**) is given instead of a variable

name, then the corresponding cartesian coordinate is neglected. In the above example, the variable name **xI** is associated with the x coordinate of the first atom given in **QM.in**. The y and z coordinates of the first atom are neglected.

All variable names must be valid Python identifiers and must not start with an underscore. Hence, all strings starting with a letter, followed by an arbitrary number of letters, digits and underscores, are valid. It is not allowed to use a variable name twice.

Note that the file header also contains the atom labels, which are just used for cross-checking against the atom labels in **QM.in**.

The file header must not contain comments, neither at the end of a line nor separate lines. Also, blank lines are not allowed in the header. After the last line of the header (where the variables for the n_{atom}-th atom are defined), blank lines and comments can be used freely (except in matrix blocks).

Variable definition blocks Variable definition blocks can be used to store additional numerical values (beyond the atomic coordinates) in variables, which can then be used in the equations in the matrix blocks. The most obvious use for this is of course to define values which will appear several times in the equations.

A variable definition block looks like:

```
Variables
A1      0.067
g1      0.996   # Trailing comment
# Blank line with comment only
R1      4.666
End
```

Each block starts with the keyword "Variables" and is terminated with "End". Inbetween, on each line a variable name and the corresponding numerical value (separated by blanks) can be given. Note that the naming conventions given above also apply to variables defined in these blocks.

There can be any number of complete variable definitions blocks in the input file. All blocks are read first, before any matrix expressions are evaluated. Hence, the relative order of the variable blocks and the matrix blocks does not matter. Also, note that variable names must not appear twice, so variables cannot be redefined halfway through the file.

Matrix blocks The most important information in the input file are of course contained in the expressions in the matrix blocks. In general, a matrix block has the following format:

```
Matrix_Identifier
V11
V12,    V22
V13,    V23,    V33
...
```

The first line identifies the type of matrix. Those are valid identifiers:

`Hamiltonian`	Defines the Hamiltonian including the diabatic potential couplings.
`Derivatives` followed by a variable name	Derivative of the Hamiltonian with respect to the given variable.
`Dipole` followed by 1, 2 or 3	(Transition) dipole moment matrix for cartesian direction x, y or z, respectively.
`Dipolederivatives` followed by 1, 2 or 3 followed by a variable name	Derivative of the respective dipole moment matrix.
`SpinOrbit` followed by **`R`** or **`I`**	Real or Imaginary (respectively) part of the spin-orbit coupling matrix $\mathbf{\Sigma}$.

Since the interface searches the file for these identifiers starting from the top until it is found, for each matrix only the first block takes effect. Note that the Hamiltonian and all relevant derivatives must be present. If dipole matrix, dipole derivative matrix or SO matrix definitions are missing, they will be assumed zero.

In the lines after the identifier, the expressions for each matrix element are given. Note the lower triangular format (all matrices are assumed symmetric, except the imaginary part of the SO matrix, which is assumed antisymmetric). Matrix elements must be separated by commas (so that whitespace can be used inside the expressions). There must be at least as many lines as the number of states (additional lines are neglected). If any line or matrix element is missing, the interface will abort.

An examplary block looks like:

```
Hamiltonian
A1*( (1.-math.exp(g1*(R1-xI+xBr)))**2-1.),
0.0006,                                          3e-5*(xI-xBr)**2
```

It is important to understand that the expressions are directly evaluated by the Python interpreter, hence all expressions must be valid Python expressions which evaluate to numeric (integer or float) values. Only the variables defined above can be used.

Note that exponentiation in Python is **`**`**. In order to provide most usual mathematical functions, the **`math`** module is available. Among others, the **`math`** module provides the following functions:

- **`math.exp(x)`**: Exponential function
- **`math.log(x)`**: Natural logarithm
- **`math.pow(x,y)`**: x^y
- **`math.sqrt(x)`**: $\sqrt{x}$
- **`math.cos(x)`**, **`math.sin(x)`**, **`math.tan(x)`**
- **`math.acos(x)`**, **`math.asin(x)`**, **`math.atan(x)`**
- **`math.atan2(y,x)`**: $\tan^{-1}\left(\frac{y}{x}\right)$, as in many programming languages (takes care of phases)
- **`math.pi`**, **`math.e`**: π and Euler's number
- **`math.cosh(x)`**, **`math.sinh(x)`**: Hyperbolic functions (also tanh, acosh, asinh, atanh are available)

7 Auxilliary Scripts

In this chapter, all auxiliary scripts and programs are documented. Input generators (currently **molpro_input.py** and **molcas_input.py**) are documented in the relevant interface sections.

All auxiliary scripts are either interactive – prompting user input from stdin in order to setup a certain task – or non-interactive, meaning they are controlled by command-line arguments and options, in the same way as many command-line tools work.

All interactive scripts sequentially ask a number of questions to the user. In many cases, a default value is presented, which is either preset or detected by the scripts based on the availability of certain files. Furthermore, the scripts feature auto-completion of paths and filenames (use TAB), which is active only in questions where auto-completion is relevant.

All interactive scripts write a file called **KEYSTROKES.<script_name>** which contains the user input from the last completed session. These files can be piped to the interactive scripts to perform the same task again, for example:

```
user@host> cat KEYSTROKES.excite - | $SHARC/excite.py
```

Note the **-**, which tells **cat** to switch to stdin after the file ends, so that the user can proceed if the script asks for more input than contained in the **KEYSTROKES** file.

All non-interactive scripts can be called with the **-h** option to obtain a short description, usage information and a list of the command line options.

All scripts can be safely killed during a run by using **CTRL+C**. In the case of interactive scripts, a **KEYSTROKES.tmp** file remains, containing the user input made so far.

7.1 Wigner Distribution Sampling: `wigner.py`

The first step in preparing the dynamics calculation is to obtain a set of physically reasonable initial conditions. Each initial condition is a set of initial atomic coordinates, initial atomic velocities and initial electronic state. The initial geometry and velocities can be obtained in different ways. With SHARC, usually sampling of a quantum-harmonic Wigner distribution is performed.

The sampling is carried out with the non-interactive Python script **wigner.py**. The theoretical background is summarized in section 8.12.

7.1.1 Usage

The general usage is

```
user@host> python $SHARC/wigner.py [options] filename.molden
```

wigner.py takes exactly one command-line argument (the input file with the frequencies and normal modes), plus some options. Usually, the **-n** option is necessary, since by default only 3 initial conditions are created.

Table 7.1: Command-line options for script **wigner.py**.

Option	Description	Default
`-h`	Display help message and quit.	–
`-m`	Modify atom masses	Most common isotope.
`-M`	Assume a MOLPRO input file.	Assume a MOLDEN file.
`-n INTEGER`	Number of initial conditions to generate.	3
`-o FILENAME`	Output filename.	**initconds**
`-r INTEGER`	Seed for random number generator.	16661
`-s FLOAT`	Scaling factor for the frequencies.	1.0

The argument is the filename of the file containing the information about the vibrational frequencies and normal modes. The file is by default assumed to be in the MOLDEN format. For usage with **wigner.py**, only the following blocks have to be present:

- [FREQ]
- [FR-COORD]
- [FR-NORM-COORD]

Alternatively, the information about the vibrational frequencies and normal modes can be read directly from a MOLPRO output file of a frequency calculation. In this case, in addition to the filename also the switch **-M** has to be given:

```
user@host> python $SHARC/wigner.py -M MOLPRO.out
```

The script accepts a number of command-line options, specified in table 7.1.

7.1.2 Output

The script **wigner.py** generates a single output file, by default called **initconds**. All information about the initial conditions is stored in this file. Later steps in the preparation of the initial conditions add information about the excited states to this file. The file is formatted in a human-readable form.

The **initconds** file format is specified in section 7.3.5.

7.1.3 Non-default Masses

When the **-m** option is used, the script will ask the user to interactively modify the atom masses. For each atom (referred to by the atom index as in the MOLDEN file), a mass can be given (relative atomic weights). Note that the frequency calculation which produces the MOLDEN or MOLPRO file should be done with the same atomic masses.

7.2 Setup of Initial Calculations: `setup_init.py`

The interactive script **setup_init.py** creates input for singlepoint calculations at the initial geometries given in an **initconds** file. These calculations might be necessary for some schemes to select the initial electronic state of the trajectory, based on the excitation energies and oscillator strength of the transitions from ground state to the excited state under consideration.

There are other choices of the initial state possible, which do not require singlepoint calculations at all initial geometries. See the description of **excite.py** (section 7.3). In this case, **setup_init.py** can

be used to set up only the calculation at the equilibrium geometry (see below at "Range of Initial Conditions").

7.2.1 Usage

The script is interactive, and can be started by simply typing

```
user@host> python $SHARC/setup_init.py
```

Please be aware that the script will setup the calculations in the directory where it was started, so the user should **cd** to the desired directory before executing the script.

Please note that the script does not expand **~** or shell variables, except where noted otherwise.

7.2.2 Input

The script will prompt the user for the input. In the following, all input parameters are documented:

Initial Conditions File Enter the filename of the initial conditions file, which was generated beforehand with **wigner.py**. If the script finds a file called **initconds**, the user is asked whether to use this file, otherwise the user has to enter an appropriate filename. The script detects the number of initial conditions and number of atoms automatically from the initial conditions file.

Range of Initial Conditions The initial conditions in **initconds** are indexed, starting with the index 1. In order to prepare ab initio calculations for a subset of all initial conditions, enter a range of indices, e.g. a and b. This will prepare all initial conditions with indices in the interval $[a, b]$. In any case, the script will additionally prepare a calculation for the equilibrium geometry (except if a finished calculation for the equilibrium geometry was found).

If the interval $[0, 0]$ is given, the script will only setup the calculation at the equilibrium geometry.

Number of states Here the user can specify the number of excited states to be calculated. Note that the ground state has to be counted as well, e.g. if 4 singlet states are specified, the calculation will involve the S_0, S_1, S_2 and S_3. Also states of higher multiplicity can be given, e.g. triplet or quintett states. For even-electron molecules, including odd-electron states (e.g. doublets) is only useful if transition properties for ionization can be computed (e.g. Dyson norms with the COLUMBUS interface). These transition properties can be used to calculate ionization spectra or to obtain initial conditions for dynamics after ionization.

Spin-orbit calculation Usually it is sufficient to calculate the spin-free excitation energies and oscillator strengths in order to decide for the initial state. However, using this option, the effects of spin-orbit coupling on the excitation energies and oscillator strengths can be included. Note that the script will never calculate spin-orbit couplings if only singlet states are included in the calculation.

Interface In this point, choose any of the displayed interfaces to carry out the ab initio calculations. Enter the corresponding number.

The following input section depends on the chosen interface.

7.2.3 Input for Molpro

Path to Molpro Here the user is prompted to provide the path to the MOLPRO executable.

Note that the setup script will not expand the user (**~**) and shell variables (since the calculations might be running on a different machine than the one used for setup, so the meaning of shell variables could be different). **~** and shell variables will only be expanded by the interfaces during the actual calculation.

Scratch directory The script takes the string without any checking. Each individual initial condition uses a subdirectory **ICOND_%05i** with the appropriate number. **~** and shell variables will only be expanded by the interface during the actual calculation.

Note that you can use, e.g., **./temp/** as scratch directory, which is a subdirectory of the working directory of the interface. Using **./** as scratch directory is not recommended, since the interface will delete the scratch directory.

Template file Enter the filename for the MOLPRO template input. This file contains the details of the ab initio calculation, like basis set, active orbitals and electrons, number of electrons and number of states for state-averaging. For MOLPRO, it also contains the memory usage. For details, see the section about the SHARC-MOLPRO interface (6.2). The setup script will check whether the template file contains the necessary entries.

Initial wavefunction file You can provide an initial wavefunction (with an appropriate MO guess) to MOLPRO. This usually provides a drastic speedup for the CASSCF calculations and helps in ensuring that all calculations are based on the same active space. In simple cases it might be acceptable to not provide an initial wavefunction.

7.2.4 Input for Columbus

Path to Columbus Here the user is prompted to provide the path to the COLUMBUS directory. Note that the script will not expand the user (**~**) and shell variables (since possibly the ab initio calculations are running on a different machine than the one used for setup). **~** and shell variables will only be expanded by the interfaces during the actual calculation.

Note that **$COLUMBUS** does not have to be defined on the setup host, but it has to be defined on the hosts running the calculations.

Scratch directory The script takes the string without any checking. Each individual initial condition uses a subdirectory **ICOND_%05i** with the appropriate number. **~** and shell variables will only be expanded by the interface during the actual calculation.

Note that you can use, e.g., **./temp/** as scratch directory, which is a subdirectory of the working directory of the interface. Using **./** as scratch directory is not recommended, since the interface will delete the scratch directory.

Template directory Enter the path to a directory containing subdirectories with COLUMBUS input files necessary for the calculations. The setup script will expand **~** and shell variables and will pass the absolute path to the calculations.

The script will auto-detect (based on the **cidrtin** files) which subdirectory contains the input for each multiplicity. The user has to check in the following dialog whether the association of multiplicities

with job directories is correct. Additionally, the association of MO coefficient files to the jobs has to be checked. See the section on the SHARC-COLUMBUS interface (6.4) for further details.

Note that the setup script does not check any content of the input files beyond the multiplicity. Note that **`transmomin`** and **`control.run`** do not need to be present (and that their content has no effect on the calculation), since these files are written on-the-fly by the interface.

Initial orbital coefficient file You can provide initial MO coefficients for the COLUMBUS calculation. This usually provides a drastic speedup for the CASSCF calculations and helps to ensure that all calculations are based on the same active space. In simple cases it might be acceptable to not provide an initial wavefunction.

Memory For COLUMBUS, the available memory must be given here.

7.2.5 Input for Molcas

Path to Molcas The script will first look for a shell variable **`$MOLCAS`**. The user can either confirm that **`$MOLCAS`** is the correct executable, or enter the correct path. Note that the setup script will not expand the user (**~**) and shell variables (since possibly the ab initio calculations are running on a different machine than the one used for setup). The SHARC-MOLCAS interface will expand **~** and shell variables.

Scratch directory The script takes the string without any checking. Each individual initial condition uses a subdirectory **`ICOND_%05i`** with the appropriate number. **~** and shell variables will only be expanded by the interface during the actual calculation.

Note that you can use, e.g., **`./temp/`** as scratch directory, which is a subdirectory of the working directory of the interface. Using **`./`** as scratch directory is not recommended, since the interface will delete the scratch directory.

Template file Enter the filename for the MOLCAS template input. This file contains the details of the ab initio calculation, like basis set, active orbitals and electrons, number of electrons and number of states for state-averaging. For details, see the section about the SHARC-MOLCAS interface (6.3). The setup script will check whether the template file contains the necessary entries.

Initial wavefunction file You can provide initial MO coefficients to MOLCAS. This usually provides a drastic speedup for the CASSCF calculations and helps in ensuring that all calculations are based on the same active space. In simple cases it might be acceptable to not provide an initial wavefunction. For MOLCAS, you have to specify one file for each multiplicity.

7.2.6 Input for analytical potentials

Template file Enter the filename for the template input specifying the analytical potentials. For details, see the section about the SHARC-Analytical interface (6.5). The setup script will check whether the template file is valid.

7.2.7 Input for Run Scripts

Run script mode The script **setup_init.py** generates a run script (Bash) for each initial condition calculation. Due to the large variety of cluster architectures, these run scripts might not work in every case. It is the user's responsibility to adapt the generated run scripts to his needs.

setup_init.py can generate run scripts for two different schemes how to execute the calculations. With the first scheme, the ab initio calculations are performed in the directory where they were setup (subdirectories of the directory where **setup_init.py** was started). Note that the interfaces will still use their scratch directories to perform the actual quantum chemistry calculations.

With the second option, the run scripts will transfer the input files for each ab initio calculation to a temporary directory, where the interface is started. After the interface finishes all calculations, the results files are transferred back to the primary directory and the temporary directory is deleted. Note that **setup_init.py** in any case creates the directory structure in the directory where it was started. The name of the temporary directory can contain shell variables, which will be expanded when the script is running (on the compute host).

Submission script The setup script can also create a Bash script for the submission of all ab initio calculations to a queueing system. The user has to provide a submission command for that, including any options which might be necessary. This submission script might not work with all queueing systems.

Project name The user can enter a project name. This is used currently only for the job names of submitted jobs (**-N** option for queueing system).

7.2.8 Output

setup_init.py will create for each initial condition in the given range a directory whose names follow the format **ICOND_%05i**, where **%05i** is the index of the initial condition padded with zeroes to 5 digits. Additionally, the directory **ICOND_00000** is created for the calculation of the excitation energies at the equilibrium geometry.

To each directory, the following files will be added:

- **QM.in**: Main input file for the interface, contains the geometry and the control keywords (to specify which quantities need to be calculated).
- **run.sh**: Run script, which can be started interactively in order to perform the ab initio calculation in this directory. Can also be adapted to a batch script for submission to a queue
- Interface-specific files: Usually a template file, a file containing additional input for the interface, and an initial wavefunction.

The calculations in each directory can be simply executed by starting **run.sh** in each directory. In order to perform this task consecutively on a single machine, the script **all_run.sh** can be executed. The file **DONE** contains the progress of this calculation. Alternatively, each run script can be sent to a queueing system (you might need to adapt this script to you cluster system).

In figure 7.1, the directory tree structure setup by **setup_init.py** is given.

After all calculations are finished, **excite.py** can be used to collect the results.

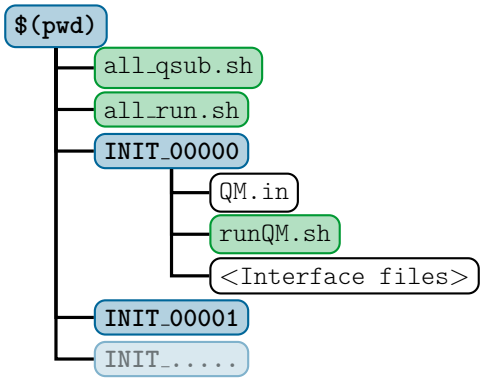

Figure 7.1: Directory structure created by **setup_init.py**. Directories are in blue, executable scripts in green and regular files in black and white. Interface files usually include initial MO coefficients, template files and interface input files.

7.3 Excitation Selection: excite.py

excite.py has two tasks: adding excited-state information to the **initconds** file, and deciding which excited state for which initial condition is a valid initial state for the dynamics.

7.3.1 Usage

The script is interactive, and can be started by simply typing

```
user@host> python $SHARC/excite.py
```

7.3.2 Input

Initial condition file Enter the path to the initial conditions file, to which **excite.py** will add excited-state information. This file can already contain excited-state information (in this case this information can be reused).

Generate excited state list There are three possibilities to add excited-state information to the **initconds** file:

1. generate a list of dummy excited states,
2. read excited-state information from the output of the initial ab initio calculations (prepare the calculations with **setup_init.py**),
3. keep the existing excited-state information in the **initconds** file.

The first option is mainly used if no initial ab initio calculations need to be performed (e.g., the initial state is known).

In order to use the second option, one should first setup initial excited-state calculations using **setup_init.py** (see 7.2) and run the calculations. **excite.py** can then read the output of the initial calculations and calculate excitation energies and oscillator strengths.

The third option can be used to reuse the information in the **initconds** file, e.g., to apply a different selection scheme to the states or to just read the number of states.

Path to ab initio results If **excite.py** will read the excited-state information from the ab initio calculation results, here the user has to provide the path to the directory containing the **ICOND_%05i** subdirectories.

Number of states If a dummy list of states will be generated, the user has to provide the number of states per multiplicity. Note that a singlet ground state has to be counted as well, e.g. if 4 singlet states are specified, the calculation will involve the S_0, S_1, S_2 and S_3. Also states of higher multiplicity can be given, e.g. doublet or triplet states (e.g., **2 2 1** for two singlets, two doublets and one triplet).

If the ab initio results are read the number of states will be automatically determined from the results.

Excited-state representation When generating new lists of excited states (either dummy states or from ab initio results), the user has to specify the representation of the excited states (either MCH or diagonal representation). The MCH representation is spin-free, meaning that transition dipole moments are only allowed between states of the same multiplicity. For molecules without heavy atoms, this option is sufficient. For heavier atoms, the diagonal representation can be used, which includes the effects of spin-orbit coupling on the excitation energies and oscillator strengths.

When reading ab initio results, **excite.py** will diagonalize the Hamiltonian and transform the transition dipole matrices for each initial condition to obtain the diagonal representation.

When a dummy state list is generated, the representation will only be written to the output file **initconds.excited** (but has no actual numeric effect for **excite.py**). Note that the representation which is declared in the **initconds.excited** file influences how SHARC determines the initial coefficients (see the paragraph on initial coefficients in 4.1.3).

Note that the representation cannot be changed if existing excited-state information is kept.

Hint: If the **ICOND_%05i** directories need to be deleted (e.g., due to disk space restrictions), making one read-out with **excite.py** for each representation and saving the results to two different files will preserve all necessary information.

Ionization probabilities If **excite.py** detects that the ab initio results contain ionization probabilities, then those can be used instead of the transition dipole moments. Note that in this case the transition dipole moments are not written to the **initconds.excited** file.

Reference energy **excite.py** can read the reference energy (ground state equilibrium energy) directly from the ab initio results. If the ab initio data is read anyways, **excite.py** already knows the relevant path. If a dummy list of states is generated, the user can provide just the path to the **QM.out** file of the ab initio calculation for the equilibrium geometry. Otherwise, **excite.py** will prompt the user to enter a reference energy manually (in hartree).

Initial state selection Every excited state of each initial condition has a flag specifying it either as a valid initial state or not. **excite.py** has four modes how to flag the excited states:

1. Unselect all excited states,
2. User provides a list of initial states,
3. States are selected stochastically based on excitation energies and oscillator strengths,
4. Keep all existing flags.

The first option can be used if **excite.py** is only used to read the ab initio results for the generation of an absorption spectrum (using **spectrum.py**).

The second option can be used to directly specify a list of initial states, if the initial state is known (e.g., starting in the ground state and exciting with an explicit laser field). In this case, the given states of *all* initial conditions are flagged as initial states.

The third option is only available if excited-state information exists (i.e., if no dummy list is generated). For details on the stochastic selection procedure, see section 8.5.

The fourth option can only be used if the existing state information is kept. In this case **`excite.py`** does nothing except counting the number of flagged initial states.

Excitation window This option allows to exclude excited states from the selection procedure if they are outside a given energy window. This option is only available if excited state information exists, but not if a dummy list of states is generated (because the dummy states have no defined excitation energy).

For the stochastic selection procedure, states outside the excitation window do not count for the determination of $p_{\max}$ (see equation (8.10)). This allows to excite, e.g., to a dark $n\pi^*$ state despite the presence of a much brighter $\pi\pi^*$ state.

For the keep-flags option, this option can be used to count the number of excited states in the energy window.

Considered states Here the user can specify the list of desired initial states. For the stochastic selection procedure, the user can instead exclude certain states from the procedure. Excluded states do not count for the determination of $p_{\max}$ (see equation (8.10)).

If the number of states per multiplicity is known, **`excite.py`** will print a table giving for each state index the multiplicity, quantum number and M_s value.

Random number generator seed The random number generator in **`excite.py`** is used in the stochastic selection procedure. Instead of typing an integer, typing "**`!`**" will initialize the RNG from the system time. Note that this will not be reproducible, i.e. repeating the **`excite.py`** run with "**`!`**" as random seed will give a different selection in each run.

7.3.3 Matrix diagonalization

When using the diagonal representation, **`excite.py`** needs to diagonalize and multiply matrices. By default, the Python package NUMPY is used, if available. If the script does not find a NUMPY installation, it will use a small Fortran code which comes with the SHARC suite. In order for this to work, you need to set the environment variable **`$SHARC`** to the **`bin/`** directory within your SHARC installation. See section 7.13 for more details.

7.3.4 Output

`excite.py` writes all output to a file **`<BASE>.excited`**, where **`<BASE>`** is the name of the initial conditions file used as input. The output file is also an initial conditions file, but contains additional information regarding the excited states, the reference energy and the representation of the excited states. An initial conditions file with excited-state information is needed for the final preparatory step: setting up the dynamics with **`setup_traj.py`**. Additionally, **`spectrum.py`** can calculate absorption spectra from excited-state initial condition files.

7.3.5 Specification of the `initconds.excited` file format

The initial conditions files **`initconds`** and **`initconds.excited`** contain lists of initial conditions, which are needed for the setup of trajectories. An initial condition is a set of initial coordinates of all atoms and corresponding initial velocities of each atom, and optionally a list of excited state informations. In the following, the format of this file is specified.

The file contains of a header, followed by the body of the file containing a list of the initial conditions.

File header An examplary header looks like:

```
SHARC Initial conditions file, version 0.2   <Excited>
Ninit     100
Natom     2
Repr      MCH
Eref         -0.50
Eharm           0.04
States    2 0 1

Equilibrium
 H   1.0  0.0  0.0  0.0   1.00782503   0.0  0.0  0.0
 H   1.0  1.5  0.0  0.0   1.00782503   0.0  0.0  0.0
```

The first line must read **`SHARC Initial conditions file, version <VERSION>`**, with the correct version string followed. The string **`Excited`** is optional, and marks an initial conditions file as being an output file of **`excite.py`** (**`setup_traj.py`** will only accept files marked like this). The following lines contain:

1. the number of initial conditions,
2. the number of atoms,
3. the electronic state representation (a string which is **`None`**, **`MCH`** or **`diag`**),
4. the reference energy (hartree),
5. the harmonic energy (zero point energy in the harmonic approximation, hartree),
6. optionally the number of states per multiplicity.

After the header, first the equilibrium geometry is expected. It is demarked with the keyword **`Equilibrium`**, followed by n_{atom} lines, each specifying one atom. Unlike the actual initial conditions, the equilibrium geometry does not have a list of excited states or defined energies.

File body The file body contains a list of initial conditions. Each initial condition is specified by a block starting with a line containing the string **`Index`** and the number of the initial condition. In the file, the initial conditions are expected to appear in order.

A block specifying an initial condition looks like:

```
Index     1
Atoms
 H   1.0  -0.02  0.0  0.0   1.00782503  -0.001  0.0  0.0
 H   1.0   1.52  0.0  0.0   1.00782503   0.001  0.0  0.0
States
```

```
001    -0.49    -0.49  -0.16   0.0  -0.03   0.0   0.05   0.0   0.0   0.00 False
002    -0.25    -0.49   0.02   0.0   0.43   0.0  -1.77   0.0   6.5   0.53 True
003    -0.40    -0.49   0.00   0.0   0.00   0.0   0.00   0.0   2.5   0.00 False
004    -0.40    -0.49   0.00   0.0   0.00   0.0   0.00   0.0   2.5   0.00 False
005    -0.40    -0.49   0.00   0.0   0.00   0.0   0.00   0.0   2.5   0.00 False
Ekin        0.004 a.u.
Epot_harm   0.026 a.u.
Epot        0.013 a.u.
Etot_harm   0.030 a.u.
Etot        0.018 a.u.
```

The formal structure of such a block is as follows. After the line containing the keyword **Index** and the index number, the keyword **Atoms** indicates the start of the list of atoms. Each atom is specified on one line:

1. symbol,
2. nuclear charge,
3. x, y, z coordinate in Bohrs,
4. atomic mass,
5. x, y and z component of nuclear velocity in atomic units.

After the atom list, the keyword **States** indicates the list of electronic states. This list consists of one line per electronic state, but can be empty, if no information of the electronic states is available. Each line consists of:

1. state number (starting with 1),
2. state energy in Hartree,
3. reference energy in Hartree (usually the energy of the lowest state),
4. six numbers defining the transition dipole moment to the reference state (usually the lowest state),
5. the excitation energy in eV,
6. the oscillator strength,
7. a string which is either **True** or **False**, specifying whether the electronic state was selected by **excite.py** as initial electronic state.

The transition dipole moments are specified by six floating point numbers, which are real part of the x component, imaginary part of the x component, then the real and imaginary parts for the y and finally the z component (the transition dipole moments can be complex in the diagonal representation).

The electronic state list is terminated with the keyword **Ekin**, which at the same time gives the kinetic energy of all atoms. The remaining entries give the potential energy in the harmonic approximation and the actual potential energy, as well as the total energy.

7.4 Setup of Trajectories: `setup_traj.py`

This interactive script prepares the input for the excited-state dynamics simulations with SHARC. It works similarly to **setup_init.py**, reading an initial conditions file, prompting the user for a number of input parameters, and finally prepares one directory per trajectory. However, the **setup_traj.py** input section is noticeably longer, because many options for the dynamics are covered.

7.4.1 Input

Initial conditions file Please be aware that **`setup_traj.py`** needs an initial conditions file generated by **`excite.py`** (Files generated by **`wigner.py`** are not allowed). The script reads the number of initial states, the representation and the reference energy automatically from the file.

Number of states This is the total number of states per multiplicity included in the dynamics calculation. Affects the keyword **`nstates`** in the SHARC input file.

Only advanced users should use here a different number of states than given to **`setup_init.py`**. In this case, the excited-state information in the initial conditions file might be inconsistent. For example, if 10 singlets and 10 triplets were included in the initial calculations, but only 5 singlets and 5 triplets in the dynamics, then the sixth entry in the initial conditions file corresponds to S_5, while **`setup_traj.py`** assumes the sixth entry to correspond to T_1.

Active states States can be frozen for the dynamics calculation here. See section 8.2 for a general description of state freezing in SHARC. Only the highest states in each multiplicity can be frozen, it is not possible to, e.g., freeze the ground state in simulations where ground state relaxation is negligible. Affects the keyword **`actstates`**.

Contents of the initial conditions file Optionally, a map of the contents of the initial conditions file can be displayed during the execution of **`setup_traj.py`**, showing for each state which initial conditions were selected (and which initial conditions do not have the necessary excited-state information). For each state, a table is given, where each symbol represents one initial condition. A dot "**`.`**" represents an initial condition where information about the current excited state is available, but which is not selected for dynamics. A hash mark "**`#`**" represents an initial condition which is selected for dynamics. A question mark "**`?`**" represents initial conditions for which no information about the excited state is available (e.g. if the initial excited-state calculation failed). The tutorial shows an example of this output.

The content of the initial conditions file is also summarized in a table giving the number of initial conditions selected per state.

Initial states for dynamics setup The user has to input all states from which trajectories should be launched. The numbers must be entered according to the above table giving the number of selected initial conditions per state. It is not allowed to specify inactive states as initial states. The script will give the number of trajectories which can be setup with the specified set of states. If no trajectories can be setup, the user has to specify another set of initial states. The initial state will be written to the SHARC input, specified in the same representation as given in the initial conditions file. The initial coefficients will be determined automatically by SHARC, according to the description in section 4.1.3.

Starting index for dynamics setup Specifies the first initial condition within the initial condition file to be included in the setup. This is useful, for example, if the user might setup 50 trajectories starting with index 1. **`setup_traj.py`** reports afterwards the last initial condition to be used for setup, e.g. index 90. Later, the user can setup additional trajectories, starting with index 91.

Random number generator seed The random number generator in **`setup_traj.py`** is used to randomly generate RNG seeds for the SHARC input. Instead of typing an integer, typing "**`!`**" will initialize the RNG from the system time. Note that this will not be reproducible, i.e. repeating the

setup_traj.py run (with the same input) with "**!**" as random seed will give for the same trajectories different RNG seeds. Affects the keyword **RNGseed**.

Interface In this point, choose any of the displayed interfaces to carry out the ab initio calculations. Enter the corresponding number. The choice of the interface influences some dynamics options which can be set in the next section of the **setup_traj.py** input.

Simulation Time This is the maximum time that SHARC will run the dynamics simulation. If trajectories need to be run for longer time, it is recommended to first let the simulation finish. Afterwards, increase the simulation time in the corresponding SHARC input file (keyword **tmax**) and add the restart keyword (also make sure that the **norestart** keyword is not present). Then the simulation can be restarted by running again the **run.sh** script. Sets the keyword **tmax** in the SHARC input files.

Simulation Timestep This gives the timestep for the dynamics. The on-the-fly ab initio calculations are performed with this timestep, as is the propagation of the nuclear coordinates. A shorter timestep gives more accurate results, especially if light atoms (hydrogen) are subjected to high kinetic energies or steep gradients. Of course a shorter timestep is computationally more expensive. A good compromise in many situations is 0.5 fs. Sets the keyword **stepsize** in the SHARC input files.

Number of substeps This gives the number of substeps for the interpolation of the Hamiltonian for the propagation of the electronic wavefunction. Usually, 25 substeps are sufficient. In cases where the diagonal elements of the Hamiltonian are very large (very large excitation energies or a badly chosen reference energy) more substeps are necessary. Sets the keyword **nsubsteps** in the SHARC input files.

Prematurely terminate trajectories Usually, trajectories which relaxed to the ground state do not recross to an excited state, but vibrate indefinitely in the ground state. If the user is not interested in these vibrations, such trajectories can be terminated prematurely in order to save computational ressources. A threshold of 10–20 fs is usually a good choice to safely detect ground state relaxation. Sets the keyword **killafter** in the SHARC input files.

Representation for the dynamics Either the diagonal representation can be chosen (by typing "**yes**") to perform dynamics with the SHARC methodology, or the dynamics can be performed on the MCH states (spin-diabatic dynamics [24], FISH [23]). Sets the keyword **surf** in the SHARC input files.

Non-adiabatic couplings Electronic propagation can be performed with temporal derivatives, non-adiabatic coupling vectors or overlap matrices (Local diabatization). Enter the corresponding number. Note that depending on the chosen interface, some options might not be available, as displayed by **setup_traj.py**. Sets the keyword **coupling** in the SHARC input files.

Gradient transformation The non-adiabatic coupling vectors can be used to correctly transform the gradients to the diagonal representation. If non-adiabatic coupling vectors are used anyways, this option is strongly recommended, since it gives more accurate gradients for no additional cost. Sets the keyword **gradcorrect** in the SHARC input files. If the dynamics uses the MCH representation, this question is not asked.

Surface hop treatment This option determines how the total energy is conserved after a surface hop. Sets the keyword **`ekincorrect`** in the SHARC input files.

Decoherence correction For most applications, a decoherence correction should be enabled. Requesting decoherence correction here leads to the addition of the keyword **`decoherence`** to the SHARC input files.

Note that **`setup_traj.py`** does not allow to modify the α value. In order to change the keyword **`decoherence_param`**, the user has to manually edit the SHARC input files.

Scaling and Damping These two prompts set the keywords **`scaling`** and **`damping`** in the SHARC input files. The scaling parameter has to be positive, and the damping parameter has to be in the interval $[0, 1]$.

Gradient and non-adiabatic coupling selection For dynamics in the MCH representation, selection of gradients is used by default, and only one gradient (of the current state) is calculated. Selection of non-adiabatic couplings is only relevant if they are used (for propagation, gradient correction or rescaling of the velocities after a surface hop). For the selection threshold, usually 0.5 eV is sufficient, except if spin-orbit coupling is very strong and hence the gradients mix strongly. Sets the keywords **`grad_select`** and **`nac_select`** in the SHARC input files.

Laser file The user can specify to use an external laser field during the dynamics, and has to provide the path to the laser file (see section 7.5 and 4.5). **`setup_traj.py`** will check whether the number of steps and the timesteps are compatible to the dynamics. If the interface can provide dipole moment gradients, **`setup_traj.py`** will also ask whether dipole moment gradients should be included in the simulations.

7.4.2 Interface-specific input

This input section is basically the same as for **`setup_init.py`** (sections 7.2.3 to 7.2.6). Note that for the dynamics simulations an initial wavefunction file is even more strongly recommended than for the initial excited-state calculations.

For MRCI dynamics using the COLUMBUS interface, for performance reasons it is strongly advisable to obtain the **`excitlistfiles`** before starting the dynamics.

7.4.3 Run script setup

Also this input section is very similar to the one in **`setup_init.py`** (see section 7.2).

7.4.4 Output

`setup_traj.py` will create for each initial state a directory where all trajectories starting in this state will be put. If the initial conditions file specified that the initial conditions are in the MCH representation, then the initial states will be assumed to be in the MCH representation as well. In this case, the directories will be named **`Singlet_0`**, **`Singlet_1`**, ..., **`Doublet_0`**, **`Triplet_1`**, ... If the initial states are in the diagonal representation, then the directories are simply called **`X_1`**, ... since they do not have a definite spin.

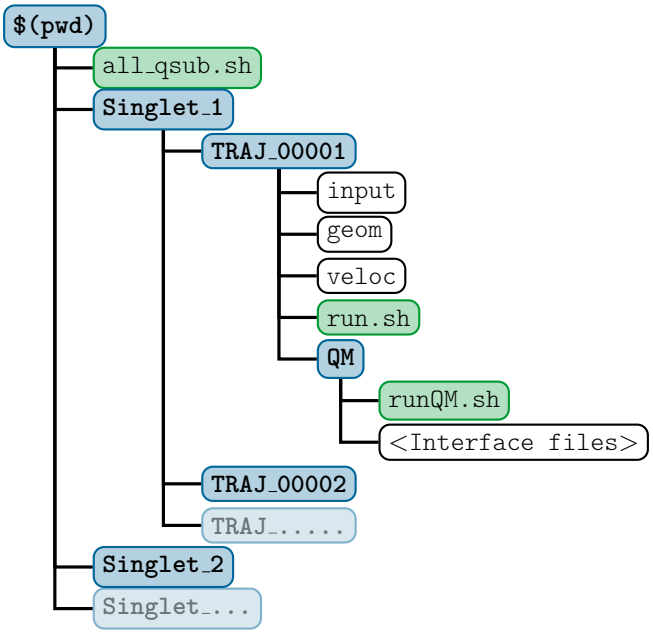

Figure 7.2: Directory structure created by **setup_traj.py**. Directories are in blue, executable scripts in green and regular files in black and white. Interface files usually include initial MO coefficients, template files and interface input files.

In each directory, subdirectories called **TRAJ_%05i** are created, where **%05i** is the initial condition index, padded to 5 digits with zeroes. In each trajectory's directory, an SHARC input file called **input** will be created, which contains all the dynamics options chosen during the **setup_traj.py** run. Also, files **geom** and **veloc** will be created. For trajectories setup with **setup_traj.py**, the determination of the initial wavefunction coefficients is done by SHARC. Furthermore, in each trajectory directory a subdirectory **QM** is created, where the **runQM.sh** script containing the call to the interface is put. In the directory **QM** also all interface-specific input files will be copied.

For each trajectory, a **run.sh** script will be created, which can be executed to run the dynamics simulation. You might need to adapt the run script to your cluster setup.

Optionally, **setup_traj.py** also creates a script **all_qsub.sh**, which can be executed to submit all trajectories to a queueing system. You might need to adapt also this script to your cluster setup.

The full directory structure created by **setup_traj.py** is given in figure 7.2.

7.5 Laser field generation: laser.x

The Fortran code **laser.x** can generate files containing laser fields which can be used with SHARC. It is possible to superimpose several lasers, use different polarizations and apply a number of chirp parameters.

7.5.1 Usage

The program is simply called by

```
user@host> python $SHARC/laser.x
```

It will interactively ask for the laser parameters. After input is complete, it writes the laser field to the file **`laser`** in the format which SHARC expects (see 4.5).

Similar to the interactive Python scripts, **`laser.x`** will also write the user input to **`KEYSTROKES.laser`**. After modifying this file, it can be used to directly execute **`laser.x`** without doing the interactive input again:

```
user@host> python $SHARC/laser.x < KEYSTROKES.laser
```

7.5.2 Input

The first four options are global and need to be entered only once, all remaining input options need to be given for every laser pulse. For the definition of laser fields see section 8.8.

Number of lasers Any number of lasers can be used. The output file will contain the sum of all laser pulses defined.

Real-valued field If this is true, the output file will only contain the real parts of the laser field, while the columns defining the imaginary part of the field will be zero. Note, however, that SHARC will anyways only use the real part of the field in the simulations.

Time interval and steps The definitions of the starting time, end time and time step of the laser field must exactly match the simulation time and time substeps of the SHARC simulation. Note, that the laser field must always start at t=0 fs to be used with SHARC. The end time for the laser field must therefore coincide with the total simulation time given in the SHARC input. The number of time steps for the laser field is $t_{\text{total}}/\Delta t_{\text{sub}} + 1$.

Files for debugging This option is normally not needed, and can be set to False. If set to True, the chirped and unchirped laser fields in both time and frequency domain will be written to files called **`DEBUG_...`**.

Polarization vector The polarization vector $\mathbf{p}$ (will be normalized).

Type of envelope There are two options possible for the envelope function $\mathcal{E}(t)$, either a Gaussian envelope or a sinusoidal one (see 8.8).

Field strength There are two input lines for the field strength $\mathcal{E}_0$, the first defining the unit in which the field strength is defined, the second gives the corresponding number. Field strength can be read in in GV/m, TW/cm^{-2} or atomic units.

FWHM and time intervals This option depends on the type of envelope chosen. While in both cases all 5 numbers need to be entered, for a Gaussian pulse only the first and third number have an effect. For a sinusoidal pulse all but the first number has an effect.

For a Gaussian pulse, the first argument corresponds to FWHM in equation (8.27) and the third argument to t_c in (8.26).

For a sinusoidal pulse, the second, third, fourth and fifth argument correspond to t_0, t_c, t_{c2} and t_e, respectively, in equation (8.28).

Table 7.2: Command-line options for script **spectrum.py**.

Option	Description	Default
`-h`	Display help message and quit.	–
`-o FILENAME`	Output filename (for the spectrum)	**spectrum.out**
`-n INTEGER`	Number of grid points	500
`-e FLOAT FLOAT`	Energy range (eV) for the spectrum	1 to 10 eV
`-i INTEGER INTEGER`	Index range for the initial conditions	1 to 1000
`-f FLOAT`	FWHM (eV) for the spectrum	0.1 eV
`-G`	Gaussian convolution	Gaussian
`-L`	Lorentzian convolution	Gaussian
`-s`	Use only selected initial conditions	Use all
`-l`	Make a line spectrum	Convolution
`--gnuplot FILENAME`	Write a GNUPLOT script	No GNUPLOT script

Central frequency There are two input lines for the central frequency ω_0. The first defines the unit (wavelength in nm, energy in eV, or atomic units). The second line gives the value.

Phase The total phase ϕ is given in multiples of π. For example, the input "**1.5**" gives a phase of $\frac{3\pi}{2}$.

Chirp parameters There are four lines giving the chirp parameters b_1, b_2, b_3 and b_4. See equation (8.30) for the meaning of these parameters.

7.6 Calculation of Absorption Spectra: spectrum.py

Aside from setting up trajectories, the **initconds.excited** files can also be used to generate absorption spectra based on the excitation energies and oscillator strengths in the file. The script **spectrum.py** calculates Gaussian or Lorentzian convolutions of these data in order to obtain spectra. See section 8.1 for further details.

spectrum.py evaluates the absorption spectrum on a grid for all states it finds in an initial conditions file. Using command-line options, some initial conditions can be omitted in the convolution, see table 7.2.

7.6.1 Input

The script is executed with the initial conditions file as argument:

```
user@host> python $SHARC/spectrum.py [OPTIONS] initconds.excited
```

The script accepts a number of command-line options, which are given in table 7.2.

7.6.2 Output

The script writes the absorption spectrum to a file (by default **spectrum.out**). Using the **-o** option, the user can redirect the output to a suitable file. The output is a table containing $n + 2$ columns,

where n is the number of states found in the initial conditions file. The first column gives the energy in eV, within the given energy interval. In columns 2 to $n + 1$ the state-wise absorption spectra are given. The last column contains the total absorption spectrum, i.e., the sum over all states. The table has $n_{\text{grid}} + 1$ rows. For line spectra the output format is exactly the same, however, the file will contain one row for each excited state of each initial condition in the initial conditions file.

Additionally, the script writes some information about the calculation to standard output, among these the maximum of the spectrum, which can be used in order to normalize the spectrum. The reported maximum is simply the largest value in the last column of the spectrum.

If requested, the script generates a GNUPLOT script, which can be used to directly plot the spectrum.

7.7 File transfer: `retrieve.sh`

Usually, SHARC will run on some temporary directory, and not in the directory where the trajectories have been submitted from. The shell script **`retrieve.sh`** is a simple **`scp`** wrapper, which can be executed (in a directory where a trajectory has been sent from) in order to retrieve the output files of this trajectory. This might not work for every cluster setup.

It relies on the presence of the file **`host_infos`**. All trajectories set up with **`setup_traj.py`** create this file after the trajectory has been started with **`run.sh`**. **`retrieve.sh`** reads **`host_infos`** to determine the hostname and working directory of the trajectory and then uses **`scp`** to retrieve the output and restart files.

The script can be called with the option "**`-lis`**" in order to only retrieve the **`output.lis`** file, but not the other output files.

If the script is called with the option "**`-res`**" then also the restart files and the content of the **`restart/`** directory are copied.

It is advisable to configure public-key authentification for the hosts running the trajectories, so that not for every execution of **`retrieve.sh`** a password has to be entered.

7.8 Data Extractor: `data_extractor.x`

The output of SHARC is mainly written to **`output.dat`**. In order to obtain plottable files in tabular format, the Fortran program **`data_extractor.x`** is used.

7.8.1 Usage

The **`data_extractor.x`** is a command line tool, and is called with the **`output.dat`** file as an argument.

```
user@host> $SHARC/data_extractor.x output.dat
```

The program will create a directory **`output_data/`** in the current working directory (not necessarily in the directory where **`output.dat`** resides). In this directory, several files are written, containing e.g. the potential energies depending on time, etc.

The program will extract the complete **`output.dat`** file until it reaches the EOF. No further input except for the data file can be given.

7.8.2 Output

After the program finishes, the directory **output_data/** contains a number of files. In each file, the number of columns is dependent in the total number n of states $i \in \{1...n\}$. The content of the files is listed in Table 7.3.

The file **expec.out** contains the information of **energy.out**, **spin.out** and **fosc.out** in one file. The content of **expec.out** can be conveniently plotted by using **make_gnuscript.py** to generate a GNUPLOT script.

7.9 Plotting the Extracted Data: make_gnuscript.py

The contents of the output files of **data_extractor.x** can be plotted with GNUPLOT. In order to quickly generate an appropriate GNUPLOT script, **make_gnuscript.py** can be used. The usage is:

```
user@host> python $SHARC/make_gnuscript.py <S> [<D> [<T> [<Q> ... ] ] ]
```

make_gnuscript.py takes between 1 and 8 integers as command-line arguments, specifying the number of singlet, doublet, triplet, etc. states. It writes an appropriate GNUPLOT script to standard out, hence redirect the output to a file, e.g.:

```
user@host> python $SHARC/make_gnuscript.py 3 0 2 > gnuscript.gp
```

Then, GNUPLOT can be run in the **output_data** directory of a trajectory:

```
user@host> gnuplot gnuscript.gp
```

This can also be accomplished in one step using a pipe, e.g.:

```
user@host> python $SHARC/make_gnuscript.py 3 0 2 | gnuplot
```

The created plot script generates four different plots (press ENTER in the command-line where you started GNUPLOT to go to the next plot). The first plot shows the potential energy of all states in the dynamics over time in the diagonal representation. The currently occupied state is marked with black circles. A thin black line gives the total energy (sum of the kinetic energy and the potential energy of the currently occupied state). Each state is colored, with one color as contour and one color at the core of the line. The contour color represents the total spin expectation value of the state. The core color represents the oscillator strength of the state with the lowest state. See figure 7.3 for the relevant color code. Note that by definition the "oscillator strength" of the lowest state with itself is exactly zero, hence the lowest state is also light grey. This dual coloring allows for a quick recognition of different types of states in the dynamics, e.g. singlets vs. triplets or $n\pi^*$ vs. $\pi\pi^*$ states.

The second plot shows the population $|c_i^{\mathrm{MCH}}|^2$ of the MCH electronic states over time. The line colors are auto-generated in order to give a large spread of all colors over the excited states, but the colors might be sub-optimal, e.g. for printing. In this cases, the user should manually adjust the colors in the generated script.

The third plot shows the population $|c_i^{\mathrm{diag}}|^2$ of the diagonal electronic states over time. These are the populations which are actually used for surface hopping. However, since these states are spin-mixed, it is usually difficult to interpret these populations.

The fourth plot shows the surface hopping probabilities over time. The plot is setup in such a way that the visible area corresponding to a certain state is proportional to the probability to hop into the state. Hence, if for a given timestep the random number (black circles) lies within a colored area, a surface hop to the corresponding state is performed.

Table 7.3: Content of the files written by **`data_extractor.x`**. n is the total number of states, j is a state index ($j \in \{1..n\}$).

File	# Columns	Columns	
`coeff_diag.out`	$2+2n$	1	Time t (fs)
		2	Norm of wavefunction $\sum_j \lvert c_j^{\text{diag}} \rvert^2$
		$1+2j$	$\Re(c_j^{\text{diag}})$
		$2+2j$	$\Im(c_j^{\text{diag}})$
`coeff_MCH.out`	$2+2n$	1	Time t (fs)
		2	Norm of wavefunction $\sum_j \lvert c_j^{\text{MCH}} \rvert^2$
		$1+2j$	$\Re(c_j^{\text{MCH}})$
		$2+2j$	$\Im(c_j^{\text{MCH}})$
`energy.out`	$4+n$	1	Time t (fs)
		2	Kinetic energy (eV)
		3	Potential energy (eV) of active state
		4	Total energy (eV)
		$4+j$	Potential energy (eV) of state j
`fosc.out`	$2+n$	1	Time t (fs)
		2	Oscillator strength of active state
		$2+j$	Oscillator strength of state j
`spin.out`	$2+n$	1	Time t (fs)
		2	Total spin expectation value of active state
		$2+j$	Total spin expectation value of state j
`prob.out`	$2+n$	1	Time t (fs)
		2	Random number from surface hopping
		$2+j$	Cumulated hopping probability $\sum_{k=1}^{j} P_k$
`expec.out`	$4+3n$	1	Time t (fs)
		2	Kinetic energy (eV)
		3	Potential energy (eV) of active state
		4	Total energy (eV)
		$4+j$	Potential energy (eV) of state j
		$4+n+j$	Total spin expectation value of state j
		$4+2n+j$	Oscillator strength of state j

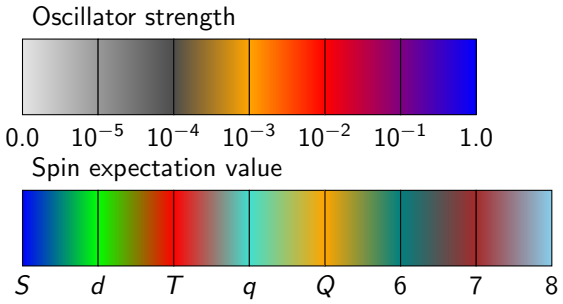

Figure 7.3: Color code for plots generated with the use of **`make_gnuscript.py`**.

7.10 Calculation of Ensemble Populations: `populations.py`

For an ensemble of trajectories, usually one of the most relevant results are ensemble-averaged populations. The interactive script **`populations.py`** collects these populations from a set of trajectories.

Different methods to obtain populations or quantities approximating populations can be collected, as described below.

7.10.1 Usage

The script is interactive, simply start it with no command-line arguments or options:

```
user@host> python $SHARC/populations.py
```

Depending on the analysis mode (see below) it might be necessary to run **`data_extractor.x`** for each trajectory prior to running **`populations.py`**.

Paths to trajectories First the script asks the user to specify all directories for whose content the analysis should be performed. Enter one directory path at a time, and finish the directory input section by typing "**`end`**". Please do not specify each trajectory directory separately, but specify their parent directories, e.g. the directories **`Singlet_1`** and **`Singlet_2`**. **`populations.py`** will automatically include all trajectories contained in these directories.

If you want to exclude certain trajectories from the analysis, it is sufficient to create an empty file called **`CRASHED`** or **`RUNNING`** in the corresponding trajectory directory. **`populations.py`** will ignore all directories containing one of these files.

Analysis mode Using **`populations.py`**, there are two basic ways in obtaining the excited-state populations. The first way is to count the number of trajectories for which a certain condition holds. For example, the number of trajectories in each classical state can be obtained in this way. However, it is also possible to count the number of trajectories for which the total spin expectation value is within a certain interval. The second way to obtain populations is to obtain the sum of the absolute squares of the quantum amplitudes over all trajectories. Table 7.4 contains a list of all possible analysis modes.

Run data extractor For analysis modes 6, 7, 8, 9 and 10 it is necessary to first run the data extractor (see section 7.8). This task can be accomplished by **`populations.py`**. However, for a large ensemble or for long trajectories this may take some time. Hence, it is not necessary to perform this step each time **`populations.py`** is run.

`populations.py` will detect whether the file **`output.dat`** or the content of **`output_data/`** is more recent. Only if **`output.dat`** is newer the **`data_extractor.x`** will be run for this trajectory.

Note that mode 10 can only be used for trajectories using local diabatization propagation (keyword **`coupling overlap`** in SHARC input file).

Number of states For analysis modes 1, 2, 3, 7, 8 and 9 it is necessary to specify the number of states in each multiplicity. The number is auto-detected from the input file of one of the trajectories.

Table 7.4: Analysis modes for **`populations.py`**. The last column indicates whether **`data_extractor.x`** has to be run prior to the ensemble analysis.

Mode	Description	From which file?	Extract?
1	For each diagonal state count how many trajectories have this state as active state.	`output.lis`	No
2	For each MCH state count how many trajectories have this state as approximate active state (see section 8.11.1).	`output.lis`	No
3	For each MCH state count how many trajectories have this state as approximate active state (see section 8.11.1). Multiplet components are summed up.	`output.lis`	No
4	Generate a histogram with definable bins (variable width). Bin the trajectories according to their total spin expectation value (of the currently active diagonal state).	`output.lis`	No
5	Generate a histogram with definable bins (variable width). Bin the trajectories according to their state dipole moment expectation value (of the currently active diagonal state).	`output.lis`	No
6	Generate a histogram with definable bins (variable width). Bin the trajectories according to the oscillator strength between lowest and currently active diagonal states.	`output_data/fosc.out`	Yes
7	Calculate the sum of the absolute squares of the diagonal coefficients for each state.	`output_data/coeff_diag.out`	Yes
8	Calculate the sum of the absolute squares of the MCH coefficients for each state.	`output_data/coeff_MCH.out`	Yes
9	Calculate the sum of the absolute squares of the MCH coefficients for each state. Multiplet components are summed up.	`output_data/coeff_MCH.out`	Yes
10	Calculate the sum of the absolute squares of the diabatic coefficients for each state (Only for trajectories with local diabatization).	`output_data/coeff_diab.out`	Yes

Intervals For analysis modes 4, 5 and 6 the user must specify the intervals for the classification of the trajectories. The user has to input a list of interval borders, e.g.:

```
Please enter the interval limits, all on one line.
Interval limits: 1e-3 0.01 0.1 1
```

Note that scientific notation can be used. Based on this input, for each timestep a histogram is created with the number of trajectories in each interval. The histogram bins are:

1. $x \leq 10^{-3}$
2. $10^{-3} < x \leq 10^{-2}$
3. $10^{-2} < x \leq 10^{-1}$
4. $10^{-2} < x \leq 10^{0}$
5. $10^{0} < x$

Note that there is always one more bin that interval borders entered.

Normalization If desired, **`populations.py`** can normalize the populations by dividing the populations by the number of trajectories.

Maximum simulation time This gives the maximum simulation time until which the populations are analyzed. For trajectories which are shorter than this value, the last population information is used to make the trajectory long enough. Trajectories which are longer are not analyzed to the end. **`populations.py`** prints the length of the shortest and longest trajectories after the analysis.

Gnuplot script **`populations.py`** can generate an appropriate GNUPLOT script for the performed population analysis.

7.10.2 Output

By default, **`populations.py`** writes the resulting populations to **`pop.out`**. If the file already exists, the user is ask whether it shall be overwritten, or to provide an alternative filename. Note that the output file is checked only after the analysis is completed, so the program might run for a considerable amount of time before asking for the output file.

7.11 Obtaining special geometries: `crossing.py`

In many cases, it is also important to obtain certain special geometries from the trajectories. The script **`crossing.py`** extracts geometries fulfilling special conditions from an ensemble of trajectories. Currently, **`crossing.py`** finds geometries where the approximate MCH state (see section 8.11.1) of the last timestep is different from the MCH state of the current timestep (i.e. **`crossing.py`** finds geometries where surface hops occured).

7.11.1 Usage

The script is interactive, simply start it with no command-line arguments or options:

```
user@host> python $SHARC/crossing.py
```

The input to the script is very similar to the one of **populations.py**.

Paths to trajectories First the script asks the user to specify all directories for whose content the analysis should be performed. Enter one directory path at a time, and finish the directory input section by typing "**end**". Please do not specify each trajectory directory separately, but specify their parent directories, e.g. the directories **Singlet_1** and **Singlet_2**. **crossing.py** will automatically include all trajectories contained in these directories.

If you want to exclude certain trajectories from the analysis, it is sufficient to create an empty file called **CRASHED** or **RUNNING** in the corresponding trajectory directory. **crossing.py** will ignore all directories containing one of these files.

Analysis mode Currently, **crossing.py** only supports one analysis mode, where **crossing.py** is scanning for each trajectory the file **output.lis**. If the occupied MCH state (column 4 in output file **output.lis**) changes from one timestep to the next, it is checked whether the old and new MCH states are the ones specified by the user. If this is the case, the geometry corresponding to the new timestep (t) is retrieved from **output.xyz** (lines $t(n_{\text{atom}} + 2) + 1$ to $t(n_{\text{atom}} + 2) + n_{\text{atom}}$).

States involved in surface hop First, the user has to specify the permissible old MCH state. The state has to be specified with two integers, the first giving the multiplicity (**1**=singlet, ...), the second the state within the multiplicity (**1 1**=S_0, **1 2**=S_1, etc.). If a state of higher multiplicity is given, **crossing.py** will report all geometries where the old MCH state is any of the multiplet components.

For the new MCH state, the same is valid.

Third, the direction of the surface hop has to be specified. Choosing "Backwards" has the same effect as exchanging the old and new MCH states.

7.11.2 Output

All geometries are in the end written to an output file, by default **crossing.xyz**. The file is in standard xyz format. The comment of each geometry gives the path to the trajectory where this geometry was extracted, the simulation time and the diagonal and MCH states at this simulation time.

7.12 Internal Coordinates Analysis: geo.py

SHARC writes at every timestep the molecular geometry to the file **output.xyz**. The non-interactive script **geo.py** can be used in order to extract internal coordinates from xyz files. The usage is:

```
user@host> python $SHARC/geo.py [options] < Geo.inp > Geo.out
```

By default, the coordinates are read from **output.xyz**, but this can be changed with the **-g** option (see table 7.6). Note that the internal coordinate specifications are read from standard input and the result table is written to standard out.

7.12.1 Input

The specifications for the desired internal coordinates are read from standard input. It follows a simple syntax, where each internal coordinate is specified by a single line of input. Each line starts with a one-letter key which specifies the type of internal coordinate (e.g. bond length, angle, dihedral, ...). The key is followed by a list of integers, specifying which atoms should be measured. As a simple example, **`r 1 2`** specifies the bond length (**`r`** is the key for bond lengths) between atoms 1 and 2. Note that the numbering of the atoms starts with 1. Each line of input is checked for consistency (whether any atom index is larger than the number of atoms, repeated atom indices, misspelled keys, wrong number of atom indices, ...), and erroneous lines are ignored (this is indicate by an error message).

Table 7.5 lists the available types of internal coordinates. The output is a table, where the first column is the time (Actually, the geometries are just enumerated starting with zero, and the number multiplied by the timestep from the **`-t`** option). The successive columns in the output table list the results of the internal coordinates calculations. Each request generates at least one column, see table 7.5.

Note that for most internal coordinates, the order of the atoms is crucial, since e.g. $a_{123} \neq a_{213}$. This also holds for the Cremer-Pople parameter requests. For these input lines, the atoms should be listed in the order they appear in the ring (clockwise or counter-clockwise).

Table 7.5: Possible types of internal coordinates in **`geo.py`**.

Key	Atom Indices	Description	Output columns
`x`	**`a`**	x coordinate of atom **`a`**	x
`y`	**`a`**	y coordinate of atom **`a`**	y
`z`	**`a`**	z coordinate of atom **`a`**	z
`r`	**`a b`**	Bond length between **`a`** and **`b`**	r
`a`	**`a b c`**	Angle between **`a`**–**`b`** and **`b`**–**`c`**	a
`d`	**`a b c d`**	Dihedral (Angle between planes (**`a,b,c`**) and (**`b,c,d`**))	d
`p`	**`a b c d`**	Pyramidalization angle (90° minus angle between bond **`a`**–**`b`** and plane (**`b,c,d`**))	p
`5`	**`a b c d e`**	Cremer-Pople parameters for 5-membered rings [40].	q_2, ϕ_2
`6`	**`a b c d e f`**	Cremer-Pople parameters for 6-membered rings [40] and Boeyens classification [41].	Q, ϕ, θ, Boeyens
`c`		Writes the comment (second line of the xyz format) to the table.	Comment

As an advice, it is always a good idea to put the comment as the *last* request, if at all. Since the comment may contain blanks, having the comment not as the very last column might make it impossible to plot the resulting table.

The Boeyens classification symbols which are output for 6-membered rings are reported in LaTeX math code. Note that in the Boeyens classification scheme a number of symbols are equivalent, and only one symbol is reported. For example, by definition the chair configuration ${}^1C_4$ is equivalent to ${}^5C_2$ and ${}^3C_6$.

7.12.2 Options

geo.py accepts a number of command-line options, see table 7.6. All options have sensible defaults. However, especially if long comments should be written to the output file, it might be necessary to increase the field width. Note that the minimum column width is 20 so that the table header can be printed correctly.

Table 7.6: Command-line options for **geo.py**.

Option	Description	Default
`-h`	Display help message and quit.	–
`-b`	Report x, y, z, r, q_2 and Q in Bohrs	Angstrom
`-r`	Report a, d, p, ϕ_2, ϕ and θ in Radians	Degrees
`-g FILENAME`	Read coordinates from the specified file	**output.xyz**
`-t FLOAT`	Assumed timestep between successive geometries (fs)	1.0 fs
`-p INTEGER`	Precision (Number of decimals, max. width–3)	4
`-w INTEGER`	Width of each column (min. 20)	20

7.13 Diagonalization Helper: diagonalizer.x

The small program **diagonalizer.x** is used by the Python scripts to diagonalize matrices if the NumPy library is not available. Currently, only **excite.py** and **SHARC_Analytical.py** need to diagonalize matrices. The program **diagonalizer.x** is implemented as a simple front-end to the LAPACK libary.

The program reads from stdin. The first line consists of the letter "r" or "c", followed by two integers giving the matrix dimensions. For "r", the program assumes a real symmetric matrix, for "c" a Hermitian matrix. The matrix must be square. The second line is a comment and is ignored. In the following lines, the matrix elements are given. On each line one row of the matrix has to be written. If the matrix is complex, each line contains the double number of entries, where real and imaginary part are always given subsequently. The following is an example input:

```
c 2 2
comment
1.0  0.0    2.0  0.1
2.0 -0.1    6.0  0.0
```

for the matrix

$$A = \begin{pmatrix} 1 & 2+0.1i \\ 2-0.1i & 6 \end{pmatrix}.$$

The diagonal matrix and the matrix of eigenvectors is written to stdout.

8 Methodology

In this chapter different aspects of SHARC simulations are discussed in detail. The topics are ordered alphabetically.

8.1 Absorption Spectrum

Using **`spectrum.py`**, an absorption spectrum can be calculated as the sum over the absorption spectra of each individual initial condition:

$$\sigma(E) = \sum_{i}^{n_{\text{init}}} \sigma_i(E), \tag{8.1}$$

where i runs over the initial conditions.

The spectrum of a single initial condition is the convolution of its line spectrum, defined through a set of tuples $(E^\alpha, f^\alpha_{\text{osc}})$ for each electronic state α, where E^α is the excitation energy and $f^\alpha_{\text{osc}})$ is the oscillator strength.

The convolution of the line spectrum can be performed with **`spectrum.py`** using either Gaussian or Lorentzian functions. The contribution of a state α to the absorption spectrum $\sigma^\alpha(E)$ is given by:

$$\sigma^\alpha_{\text{Gaussian}}(E) = (f_{\text{osc}})^\alpha_i \, \mathrm{e}^{c\left(E-E^\alpha_i\right)^2}, \tag{8.2}$$

$$\text{with} \qquad c = -\frac{4\ln(2)}{\text{FWHM}^2}, \tag{8.3}$$

or

$$\sigma^\alpha_{\text{Lorentzian}}(E) = \frac{(f_{\text{osc}})^\alpha_i}{\frac{1}{c}\left(E - E^\alpha_i\right)^2 + 1}, \tag{8.4}$$

$$\text{with} \qquad c = \frac{1}{4}\text{FWHM}^2, \tag{8.5}$$

where FWHM is the full width at half maximum.

8.2 Active and inactive states

SHARC allows to "freeze" certain states, which then do not participate in the dynamics. Only energies and dipole moments are calculated, but all couplings are disabled. In this way, these states are never visited (hence also no gradients and non-adiabatic couplings are calculated, making the inclusion of these states cheap). Example:

```
nstates 2 0 2
actstates 2 0 1
```

In the example given, state T_2 is frozen. The corresponding Hamiltonian looks like:

$$\mathbf{H} = \begin{pmatrix}
E(S_0) & & a_{01}^* & a_{02}^* & b_{01}^* & b_{02}^* & a_{01} & a_{02} \\
 & E(S_1) & a_{11}^* & a_{12}^* & b_{11}^* & b_{12}^* & a_{11} & a_{12} \\
a_{01} & a_{11} & E(T_1) & p_{12}^* & & -q_{12}^* & & \\
a_{02} & a_{12} & p_{12} & E(T_2) & q_{12}^* & & & \\
b_{01} & b_{11} & & -q_{12} & E(T_1) & & & -q_{12}^* \\
b_{02} & b_{12} & q_{12} & & & E(T_2) & q_{12}^* & \\
a_{01}^* & a_{11}^* & & & & -q_{12} & E(T_1) & p_{12} \\
a_{02}^* & a_{12}^* & & & q_{12} & & p_{12}^* & E(T_2)
\end{pmatrix}$$

where all matrix elements marked red are deleted, since T_2 is frozen.

The corresponding matrix elements are also deleted from the non-adiabatic coupling and overlap matrices. For propagation including laser fields, also the corresponding transition dipole moments are neglected, while the transition dipole moments still show up in the output (in order to characterize the frozen states).

Active and frozen states are defined with the **`states`** and **`actstates`** keywords in the input file. Note that only the highest states in each multiplicity can be frozen, i.e., it is not possible to freeze the T_1 while having T_2 active. However, it is possible to freeze all states of a certain multiplicity.

8.3 Damping

If damping is activated in SHARC (keyword **`dampeddyn`**), in each timestep the following modification to the velocity vector is made

$$\mathbf{v}' = \mathbf{v} \cdot \sqrt{C} \tag{8.6}$$

where C is the damping factor given in the input. Hence, in each timestep the kinetic energy is modified by

$$E'_{\text{kin}} = E_{\text{kin}} \cdot C \tag{8.7}$$

The damping factor C must be between 0 and 1.

8.4 Decoherence

In surface hopping, without any corrections the coherence between the states is usually too large [19]. A trajectory in state β, but where state α has a large coefficient, will still travel according to the gradient of state β. However, the gradients of state α are almost certainly different to the ones of state β. As a consequence, too much population of state α is following the gradient of state β. Decoherence corrections damp in different ways the population of all states $\alpha \neq \beta$, so that only population of β follows the gradient of state β.

The decoherence correction currently implemented in SHARC is based on the energies of the excited states ("energy-based decoherence", or EDC, as given in [50]). After the surface hopping procedure, when the system is in state β, the coefficients are updated by the following relation

$$c'_\alpha = c_\alpha \cdot \exp\left[-\Delta t \frac{|E_\alpha - E_\beta|}{\hbar}\left(1 + \frac{C}{E_{\text{kin}}}\right)^{-1}\right], \qquad \alpha \neq \beta, \tag{8.8}$$

$$c'_\beta = \frac{c_\beta}{|c_\beta|} \cdot \left[1 - \sum_{\alpha \neq \beta} |c'_\alpha|^2\right]^{\frac{1}{2}} \tag{8.9}$$

where C is the decoherence parameter. The decoherence correction can be activated with the keyword **decoherence** in the input file. The decoherence parameter C can be set with the keyword **decoherence_param** (the default is 0.1 hartree, as suggested in [50]).

8.5 Excitation Selection

excite.py can select initial active states for the dynamics based on for each initial condition k the excitation energies $E_{k,\alpha}$ and the oscillator strengths $f_{k,\alpha}^{\text{osc}}$ are known for the excited states α. The procedure is based on reference [39].

First, for all excited states of all initial conditions, the maximum value p_{max} of

$$p_{k,\alpha} = \frac{f_{k,\alpha}^{\text{osc}}}{E_{k,\alpha}^2} \tag{8.10}$$

is found. Then for each excited state, a random number $r_{k,\alpha}$ is picked from $[0, 1]$. If

$$r_{k,\alpha} < \frac{p_{k,\alpha}}{p_{\text{max}}} \tag{8.11}$$

then the excited state is selected for performing the dynamics. This excited-state selection scheme is taken from [39].

Within **excite.py** it is possible to restrict the selection to a subset of all excited states. In this case, also p_{max} is only determined based on this subset of excited states.

8.6 Internal coordinates definitions

In this section, the internal coordinates available in **geo.py** are defined.

Bond length The bond length between two atoms a and b is:

$$r_{ab} = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2 + (z_a - z_b)^2} \tag{8.12}$$

Bond angle The bond angle for atoms a, b and c is defined as the angle

$$\theta = \cos^{-1}\left(\frac{\mathbf{v}_{ba} \cdot \mathbf{v}_{bc}}{|\mathbf{v}_{ba}| \cdot |\mathbf{v}_{bc}|}\right) \tag{8.13}$$

where $\mathbf{v}_{ba}$ is the vector from atom b to atom a.

Dihedral The dihedral angle is defined via the vectors $\mathbf{w}_1$ and $\mathbf{w}_2$:

$$\mathbf{w}_1 = \mathbf{v}_{ab} \times \mathbf{v}_{bc} \qquad \text{and} \qquad \mathbf{w}_1 = \mathbf{v}_{ab} \times \mathbf{v}_{bc}. \tag{8.14}$$

The dihedral is given as the angle between $\mathbf{w}_1$ and $\mathbf{w}_2$ according to equation (8.13).

Pyramidalization angle The pyramidalization angle is defined via the vectors $\mathbf{v}_{ba}$ and

$$\mathbf{w}_1 = \mathbf{v}_{bc} \times \mathbf{v}_{bd}. \tag{8.15}$$

The pyramidalization angle is then given as $90° - \theta(\mathbf{v}_{ba}, \mathbf{w}_1)$.

Cremer-Pople parameters The definitions of the Cremer-Pople parameters for 5- and 6-membered rings is described in [40].

Boeyens classification The Boeyens classification scheme is described in [41].

8.7 Kinetic energy adjustments

There are several options how to adjust the kinetic energy after a surface hop occured. The simplest option is to not adjust the kinetic energy at all (input: **`ekincorrect none`**), but this obviously leads to violation of the conservation of total energy.

Alternatively, the velocities of all atoms can be rescaled so that the new kinetic energy and the potential energy of the new state β again sum up to the total energy.

$$f = \sqrt{\frac{E_{\text{total}} - E_\beta}{E_{\text{kin}}}} \tag{8.16}$$

$$\mathbf{v}' = f\mathbf{v} \tag{8.17}$$

$$E'_{\text{kin}} = f^2 E_{\text{kin}} \tag{8.18}$$

Within this methodology, when the energy of the old state α was lower than the energy of the new state β the kinetic energy is lowered. Since the kinetic energy must be positive, this implies that there might be states which cannot be reached (their potential energy is above the total energy). A hop to such a state is called "frustrated hop" and will be rejected by SHARC. Rescaling parallel to the nuclear velocity vector is requested with **`ekincorrect parallel_vel`**.

Alternatively, according to Tully's original formulation of the surface hopping method [14], after a hop from α to β only the component of the velocity along the direction of the non-adiabatic coupling vector $\mathbf{K}_{\beta\alpha}$ should be rescaled. With

$$a = \sum_i^{N_{\text{atom}}} \frac{\mathbf{K}_{\beta\alpha,i} \cdot \mathbf{K}_{\beta\alpha,i}}{2M_i} \tag{8.19}$$

$$b = \sum_i^{N_{\text{atom}}} \mathbf{v}_i \cdot \mathbf{K}_{\beta\alpha,i} \tag{8.20}$$

the available energy can be calculated:

$$\Delta = 4a\left(E_\alpha - E_\beta\right) + b^2. \tag{8.21}$$

If $\Delta < 0$, the hop is frustrated and will be rejected. Otherwise, the scaled velocities $\mathbf{v}'$ can be calculated as

$$\mathbf{v}'_i = \mathbf{v}_i - f\frac{\mathbf{K}_{\beta\alpha,i}}{M_i} \tag{8.22}$$

$$\text{with} \quad f = \begin{cases} \frac{b+\sqrt{\Delta}}{2a}, & b < 0, \\ \frac{b-\sqrt{\Delta}}{2a}, & b \geq 0. \end{cases} \tag{8.23}$$

This procedure can be requested with **`ekincorrect parallel_nac`**. Note that in this case SHARC will request the non-adiabatic coupling vectors, even if they are not used for the wavefunction propagation.

8.8 Laser fields

The program **`laser.x`** can calculate laser fields as superpositions of several analytical, possibly chirped, laser pulses. In the following, the laser parametrization is given (see [51] for further details).

8.8.1 Form of the laser field

In general, the laser field $\epsilon(t)$ is a linear superposition of a number of laser pulses $l_i(t)$:

$$\epsilon(t) = \sum_i \mathbf{p}_i l_i(t), \tag{8.24}$$

where $\mathbf{p}_i$ is the normalized polarization vector of pulse i.
A pulse $l(t)$ is formed as the product of an envelope function and a field function.

$$l(t) = \mathcal{E}(t) f(t) \tag{8.25}$$

8.8.2 Envelope functions

There are two types of envelope function defined in **`laser.x`**, which are Gaussian and sinusoidal.
The Gaussian envelope is defined as:

$$\mathcal{E}(t) = \mathcal{E}_0 \mathrm{e}^{-\beta(t-t_c)^2} \tag{8.26}$$

$$\beta = \frac{4 \ln 2}{\mathrm{FWHM}^2} \tag{8.27}$$

where $\mathcal{E}_0$ is the peak field strength, FWHM is the full width at half maximum and t_c is the temporal center of the pulse.
The sinusoidal envelope is defined as:

$$\mathcal{E}(t) = \mathcal{E}_0 \begin{cases} 0 & \text{if } t < t_0, \\ \sin^2\left(\frac{\pi(t-t_0)}{2(t_c-t_0)}\right) & \text{if } t_0 < t < t_c, \\ 1 & \text{if } t_c < t < t_{c2}, \\ \cos^2\left(\frac{\pi(t-t_{c2})}{2(t_e-t_{c2})}\right) & \text{if } t_{c2} < t < t_e, \\ 0 & \text{if } t_e < t, \end{cases} \tag{8.28}$$

where again $\mathcal{E}_0$ is the peak field strength, t_0 and t_c define the interval where the field strength increases, and t_{c2} and t_e define the interval where the field strength decreases. Figure 8.1 shows the general form of the envelope functions and the meaning of the temporal parameters t_0, t_c, t_{2c} and t_e.

8.8.3 Field functions

The field function $f(t)$ is defined as:

$$f(t) = \mathrm{e}^{\mathrm{i}\left(\omega_0(t-t_c)+\phi\right)}, \tag{8.29}$$

where ω_0 is the central frequency and ϕ is the phase of the pulse. Even though the laser field is complex in this expression, in the propagation of the electronic wavefunction in SHARC only the real part is used.

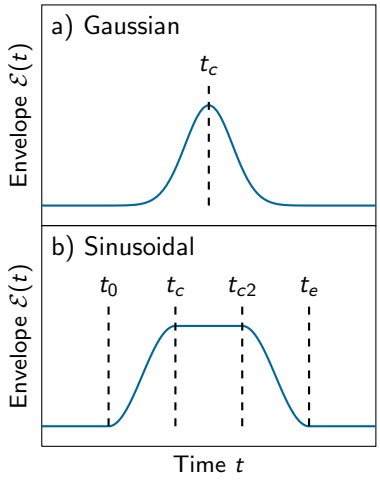

Figure 8.1: Types of laser envelopes implemented in **`laser.x`**.

8.8.4 Chirped pulses

In order to apply a chirp to the laser pulse $l(t)$, it is first Fourier transformed to the frequency domain, giving the function $\tilde{l}(\omega)$. The chirp is applied by calculating:

$$\tilde{l}'(\omega) = \tilde{l}(\omega) \mathrm{e}^{-i\left[b_1|\omega-\omega_0| + \frac{b_2}{2}(\omega-\omega_0)^2 + \frac{b_3}{6}(\omega-\omega_0)^3 + \frac{b_4}{24}(\omega-\omega_0)^4\right]} \tag{8.30}$$

The chirped laser in the time domain $l'(t)$ is then obtained by Fourier transform of the chirped pulse $\tilde{l}'(\omega)$.

8.8.5 Quadratic chirp without Fourier transform

If **`laser.x`** was compiled without the FFTW package, the only accessible chirps are quadratic chirps for Gaussian pulses:

$$l(t) = \mathcal{E}_0' \mathrm{e}^{-\beta'(t-t_c)^2} \mathrm{e}^{-\mathrm{i}\left(\omega_0(t-t_c) + a_2(t-t_c)^2 + \phi\right)} \tag{8.31}$$

$$\beta = \frac{4 \ln 2}{\mathrm{FWHM}^2} \tag{8.32}$$

$$\beta' = \frac{1}{\frac{1}{\beta} + 4\beta b_2^2} \tag{8.33}$$

$$a_2 = \frac{b_2}{\frac{1}{4\beta^2} + b_2^2} \tag{8.34}$$

$$\mathcal{E}_0' = \mathcal{E}_0 \sqrt{\frac{1}{2\mathrm{i}b_2\beta + 1}} \tag{8.35}$$

Other chirps are only possible with the Fourier transformation.

8.9 Laser interactions

The laser field ϵ is included in the propagation of the electronic wavefunction. In each substep of the propagation, the interaction of the laser field with the dipole moments is included in the

Hamiltonian. The contribution $\mathbf{V}_i$ is in each timestep added to the Hamiltonian in equations (8.70) or (8.75), respectively:

$$\mathbf{V}_i = -\Re\left(\boldsymbol{\mu}_i \cdot \boldsymbol{\epsilon}_i\right), \tag{8.36}$$

$$\boldsymbol{\mu}_i = \boldsymbol{\mu}^{\mathrm{MCH}}(t) + \frac{i}{n}\left(\boldsymbol{\mu}^{\mathrm{MCH}}(t+\Delta t) - \boldsymbol{\mu}^{\mathrm{MCH}}(t)\right), \tag{8.37}$$

$$\boldsymbol{\epsilon}_i = \boldsymbol{\epsilon}\left(t + \frac{i}{n}\Delta t\right) \tag{8.38}$$

where i, n t and Δt are defined as in section 8.19.

8.9.1 Surface Hopping with laser fields

If laser fields are present, there can be two fundamentally different types of hops: laser-induced hops and non-adiabatic hops. The latter ones are the same hops as in the laser-free simulations, and demand that the total energy is conserved. The laser-induced hops on the other hand demand that the momentum (kinetic energy) is conserved. Hence, SHARC needs to decide for every hop whether it is laser-induced or not.

Consider a previous state α and a new state β. Currently, the hop is classified based on the energy gap $\Delta E = |E_\beta^{\mathrm{diag}} - E_\alpha^{\mathrm{diag}}|$ and the instantaneous central energy of the laser pulse ω. The hop is assumed to be laser-induced if

$$|\Delta E - \omega| < W, \tag{8.39}$$

where W is a fixed parameter. W can be set using the input keyword **`laserwidth`**.

If a hop has been classified as laser-free, the momentum is adjusted according to the equations given in section 8.7.

8.10 Random initial velocities

Random initial velocities are calculated with a given amount of kinetic energy E per atom a. For each atom, the velocity is calculated as follows, with two uniform random numbers θ and ϕ, from the interval $[0, 1[$:

$$\mathbf{v} = \sqrt{2E/m_a}\begin{pmatrix}\cos\theta\sin\phi\\ \sin\theta\sin\phi\\ \cos\phi\end{pmatrix} \tag{8.40}$$

This procedure gives a uniform probability distribution on a sphere with radius $\sqrt{2E/m_a}$.

Note that the translational and rotational components of random initial velocities are not projected out in the current implementation.

Random initial velocities can be requested in the input with **`veloc random`** E, where E is a float defining the kinetic energy per atom (in eV).

8.11 Representations

Within SHARC, two different representations for the electronic states are used. The first is the so-called MCH basis, which is the basis of the eigenfunctions of the molecular Coulomb Hamiltonian. The molecular Coulomb Hamiltonian is the standard electronic Hamiltonian employed by the majority

of quantum chemistry programs. It contains only the kinetic energy of the electrons and the potential energy arising from the Coulomb interaction between the electrons and nuclei.

$$\hat{H}_{\mathrm{el}}^{\mathrm{MCH}} = \hat{K}_{\mathrm{e}} + \hat{V}_{\mathrm{ee}} + \hat{V}_{\mathrm{ne}} + \hat{V}_{\mathrm{nn}}. \tag{8.41}$$

With this hamiltonian, states of the same multiplicity couple via the non-adiabatic couplings, while states of different multiplicity do not interact at all.

The second representation used in SHARC is the so-called diagonal representation. It is the basis of the eigenfunctions of the total Hamiltonian.

$$\hat{H}_{\mathrm{el}}^{\mathrm{total}} = \hat{H}_{\mathrm{el}}^{\mathrm{MCH}} + \hat{H}_{\mathrm{el}}^{\mathrm{coup}}. \tag{8.42}$$

The term $\hat{H}_{\mathrm{el}}^{\mathrm{coup}}$ contains additional couplings not contained in the molecular Coulomb Hamiltonian. The most common couplings are spin-orbit couplings and interactions with an external electric field.

$$\hat{H}_{\mathrm{el}}^{\mathrm{coup}} = \hat{H}_{\mathrm{el}}^{\mathrm{SOC}} - \boldsymbol{\mu}\boldsymbol{\epsilon}^{\mathrm{ext}} \tag{8.43}$$

Both of these couplings introduce off-diagonal elements in the total Hamiltonian. Thus, the eigenfunctions of the molecular Coulomb Hamiltonian are not the eigenfunctions of the total Hamiltonian.

Within SHARC, usually quantum chemistry information is read in the MCH representation, while the surface hopping is performed in the diagonal one.

8.11.1 Current state in MCH representation

Oftentimes, it is very useful to know to which MCH state the currently active diagonal state corresponds. If $\hat{H}_{\mathrm{el}}^{\mathrm{coup}}$ is small or the state separation is large, then each diagonal state approximately corresponds to one MCH state. Only in the case of large couplings and/or near-degenerate states are the MCH states strongly mixed in the diagonal states.

In order to obtain for a given timestep from the currently active diagonal state β the corresponding MCH state α, a vector $\mathbf{c}^{\mathrm{diag}}$ with $c_i^{\mathrm{diag}} = \delta_{i\beta}$ is generated. The vector is transformed into the MCH representation

$$\mathbf{c}^{\mathrm{MCH}} = \mathbf{U}\mathbf{c}^{\mathrm{diag}}. \tag{8.44}$$

The corresponding MCH state α is the index of the (absolute) largest element of vector $\mathbf{c}^{\mathrm{MCH}}$.

8.12 Sampling from Wigner Distribution

The sampling is based on references [37, 38].

The frequency calculation provides a set of vibrational frequencies $\{\nu_i\}$ and the corresponding normal mode vectors $\{\mathbf{n}_i\}$, where i runs from 1 to $N = 3n_{\mathrm{atom}}$. It also provides the equilibrium geometry $\mathbf{R}_{\mathrm{eq}}$.

In order to create an initial condition $(\mathbf{R}, \mathbf{v})$, the following procedure is applied. Initially, $\mathbf{R}_0 = \mathbf{R}_{\mathrm{eq}}$ and $\mathbf{v}_0 = 0$. Then, for each normal mode i, two random numbers P_i and Q_i are chosen uniformly from the interval $[-3, 3]$. The value of a ground state quantum Wigner distribution for these values is calculated:

$$W_i = \mathrm{e}^{-P_i^2}\mathrm{e}^{-Q_i^2}. \tag{8.45}$$

W_i is compared to a uniform random r_i number from $[0,1]$. If $W_i > r_i$, then P_i and Q_i are accepted and the coordinates and velocities are updated:

$$\mathbf{R}_i = \mathbf{R}_{i-1} + \frac{Q}{\sqrt{2\nu_i}}\mathbf{n}_i \tag{8.46}$$

$$\mathbf{v}_i = \mathbf{v}_{i-1} + \frac{P\sqrt{\nu_i}}{\sqrt{2}}\mathbf{n}_i \tag{8.47}$$

$$\tag{8.48}$$

The random number procedure and updates are repeated for all normal modes, until $(\mathbf{R}_N, \mathbf{v})_N$ is obtained, which constitutes one initial condition. Finally, the center of mass is restored and translational and rotational components are projected out of $\mathbf{v}$. The harmonic potential energy is given by:

$$E_{\text{pot}} = \frac{1}{2}\sum_i \nu_i Q_i^2 \tag{8.49}$$

8.13 Scaling

The scaling factor (keyword **`scaling`**) applies to all energies and derivatives of energies. Hence, the full Hamiltonian is scaled, and the gradients are scaled. Nothing else is scaled (no dipole moments, non-adiabatic couplings, overlaps, etc).

8.14 Seeding of the RNG

The standard Fortran 90 random number generator is seeded by a sequence of integers of length n, where n depends on the computer architecture. The input of SHARC, however, takes only a single RNG seed, which must reproducibly produce the same sequence of random numbers for the same input.

In order to generate the seed sequence from the single input x, the following procedure is applied:

- Query for the number n,
- Generate a first seed sequence $\mathbf{s}$ with $s_i = x + 37i + 17i^2$,
- Seed with the sequence $\mathbf{s}$,
- Obtain a sequence $\mathbf{r}$ of n random numbers on the interval $[0,1[$,
- Generate a second seed sequence $\mathbf{s}'$ with $s'_i = \text{int}\left(65536(r_i - \frac{1}{2})\right)$,
- Reseed with the sequence $\mathbf{s}'$.

The fifth step will generate a sequence of nearly uncorrelated numbers, distributed uniformly over the full range of possible integer values.

8.15 Selection of gradients and non-adiabatic couplings

In order to increase performance, it is possible to omit the calculation of certain gradients and non-adiabatic couplings. An energy-gap-based algorithm selects at each timestep a subset of all possible gradients and non-adiabatic couplings to be calculated. Given the diagonal energy E_ξ^{diag} of the current active state ξ, the gradient $\mathbf{g}_\alpha^{\text{MCH}}$ of MCH state α is calculated if:

$$\left|E_\xi^{\text{diag}} - E_\alpha^{\text{MCH}}\right| < \varepsilon_{\text{grad}} \tag{8.50}$$

where $\varepsilon_{\text{grad}}$ is the selection threshold.
Similarly, a non-adiabatic coupling vector $\mathbf{K}^{\text{MCH}}_{\beta\alpha}$ is calculated if:

$$\left|E^{\text{diag}}_{\xi} - E^{\text{MCH}}_{\alpha}\right| < \varepsilon_{\text{nac}} \qquad \text{and} \qquad \left|E^{\text{diag}}_{\xi} - E^{\text{MCH}}_{\beta}\right| < \varepsilon_{\text{nac}} \tag{8.51}$$

with selection threshold ε_{nac}.
Neither $\mathbf{g}^{\text{MCH}}_{\alpha}$ nor $\mathbf{K}^{\text{MCH}}_{\beta\alpha}$ are ever calculated if α or β are frozen states.
There is only one keyword (**`eselect`**) to set the selection threshold, so $\varepsilon_{\text{grad}}$ and ε_{nac} are the same in most cases.

8.16 State ordering

The canonical ordering of MCH states of different S and M_S in SHARC is as follows. In the innermost loop, the quantum number is increased; then M_S and finally S. Example:

```
nstates 3 0 3
```

In this example, the order of states is given as:

Number	Label	S	M_S	n
1	S_0	0	0	1
2	S_1	0	0	2
3	S_2	0	0	3
4	T^{-}_1	2	-1	1
5	T^{-}_2	2	-1	2
6	T^{-}_3	2	-1	3
7	T^{0}_1	2	0	1
8	T^{0}_2	2	0	2
9	T^{0}_3	2	0	3
10	T^{+}_1	2	+1	1
11	T^{+}_2	2	+1	2
12	T^{+}_3	2	+1	3

The canonical ordering of states is for example important in order to specify the initial state in the MCH basis (using the **`state`** keyword in the input file).

8.17 Surface Hopping

Given two coefficient vectors $\mathbf{c}^{\text{diag}}(t)$ and $\mathbf{c}^{\text{diag}}(t+\Delta t)$ and the corresponding propagator matrix $\mathbf{R}^{\text{MCH}}(t+\Delta t, t)$, the surface hopping probabilities are given by

$$P_{\beta\to\alpha} = \left(1 - \frac{\left|c^{\text{diag}}_{\beta}(t+\Delta t)\right|^2}{\left|c^{\text{diag}}_{\beta}(t)\right|^2}\right) \times \frac{\Re\left[c^{\text{diag}}_{\alpha}(t+\Delta t) R^{*}_{\alpha\beta}\left(c^{\text{diag}}_{\beta}(t)\right)^{*}\right]}{\left|c^{\text{diag}}_{\beta}(t)\right|^2 - \Re\left[c^{\text{diag}}_{\beta}(t+\Delta t) R^{*}_{\beta\beta}\left(c^{\text{diag}}_{\beta}(t)\right)^{*}\right]}. \tag{8.52}$$

where, however, $P_{\beta\to\beta} = 0$ and all negative $P_{\beta\to\alpha}$ are set to zero.

The hopping procedure itself obtains a uniform random number r from the interval $[0, 1]$. A hop to state α is performed, if

$$\sum_{i=1}^{\alpha-1} P_{\beta\to i} < r \leq P_{\beta\to\alpha} + \sum_{i=1}^{\alpha-1} P_{\beta\to i} \tag{8.53}$$

See section 8.7 for further details on how frustrated hops are handled.

8.18 Velocity Verlet

The nuclear coordinates of atom A are updated according to the Velocity Verlet algorithm [52], based on the gradient of state β at $\mathbf{R}(t)$ and $\mathbf{R}(t + \Delta t)$:

$$\mathbf{a}_A(t) = -\frac{1}{m_A}\nabla_{\mathbf{R_A}} E_\beta(\mathbf{R}(t)) \tag{8.54}$$

$$\mathbf{a}_A(t+\Delta t) = -\frac{1}{m_A}\nabla_{\mathbf{R_A}} E_\beta(\mathbf{R}(t+\Delta t)) \tag{8.55}$$

$$\mathbf{R}_A(t+\Delta t) = \mathbf{R}_A(t) + \mathbf{v}_A(t)\Delta t + \frac{1}{2}\mathbf{a}_A(t)\Delta t^2 \tag{8.56}$$

$$\mathbf{v}_A(t+\Delta t) = \mathbf{v}_A(t) + \frac{1}{2}\left[\mathbf{a}_A(t) + \mathbf{a}_A(t+\Delta t)\right]\Delta t \tag{8.57}$$

Currently, there are no other integrators for the nuclear motion implemented in SHARC.

8.19 Wavefunction propagation

The electronic wavefunction is needed in order to carry out surface hopping. The electronic wavefunction is expanded in the basis of the so-called model space $\mathcal{S}$, which includes the few lowest states $|\psi_\alpha^{\mathrm{MCH}}\rangle$ of the multiplicities under consideration (e.g. the 3 lowest singlet and 2 lowest triplet states).

$$\Psi_{\mathrm{el}}(t) = \sum_{\alpha\in\mathcal{S}} c_\alpha^{\mathrm{MCH}} \left|\psi_\alpha^{\mathrm{MCH}}\right\rangle \tag{8.58}$$

All multiplet components are included explicitly, i.e., the inclusion of an MCH triplet state adds three explicit states to the model space (the three components of the triplet).

Within SHARC, the wavefunction is represented just by the vector $\mathbf{c}^{\mathrm{MCH}}$. The Hamiltonian $\mathbf{H}^{\mathrm{MCH}}$ is represented in matrix form with elements:

$$H_{\beta\alpha}^{\mathrm{MCH}} = \left\langle \psi_\beta^{\mathrm{MCH}} \middle| \hat{H}_{\mathrm{el}}^{\mathrm{total}} \middle| \psi_\alpha^{\mathrm{MCH}} \right\rangle \tag{8.59}$$

From the MCH representation, the diagonal representation can be obtained by unitary transformation within the model space $\mathcal{S}$ ($\mathbf{U}^\dagger\mathbf{H}^{\mathrm{MCH}}\mathbf{U} = \mathbf{H}^{\mathrm{diag}}$ and $\mathbf{U}^\dagger\mathbf{c}^{\mathrm{MCH}} = \mathbf{c}^{\mathrm{diag}}$):

$$\Psi_{\mathrm{el}}(t) = \sum_{\alpha\in\mathcal{S}} c_\alpha^{\mathrm{diag}} \left|\psi_\alpha^{\mathrm{diag}}\right\rangle \tag{8.60}$$

and

$$H_{\beta\alpha}^{\mathrm{diag}} = \left\langle \psi_\beta^{\mathrm{diag}} \middle| \hat{H}_{\mathrm{el}}^{\mathrm{total}} \middle| \psi_\alpha^{\mathrm{diag}} \right\rangle \tag{8.61}$$

The propagation of the electronic wavefunction from time t to $t + \Delta t$ can then be written as the product of a propagation matrix with the coefficients at time t:

$$\mathbf{c}^{\text{diag}}(t + \Delta t) = \mathbf{R}^{\text{diag}}(t + \Delta t, t)\mathbf{c}^{\text{diag}}(t) \tag{8.62}$$

or

$$\mathbf{c}^{\text{diag}}(t + \Delta t) = \underbrace{\mathbf{U}^{\dagger}(t + \Delta t)\mathbf{R}^{\text{MCH}}(t + \Delta t, t)\mathbf{U}(t)}_{\mathbf{R}^{\text{diag}}(t+\Delta t,t)} \mathbf{c}^{\text{diag}}(t) \tag{8.63}$$

In order to calculate $\mathbf{R}^{\text{MCH}}(t + \Delta t, t)$, SHARC uses (unitary) operator exponentials.

8.19.1 Propagation using non-adiabatic couplings

Here we assume that in the dynamics the interaction between the electronic states is described by a matrix of non-adiabatic couplings $\mathbf{K}^{\text{MCH}}(t)$, such that

$$\left(\mathbf{K}^{\text{MCH}}(t)\right)_{\beta\alpha} = \left\langle \psi_\beta(t) \middle| \frac{\partial}{\partial t} \middle| \psi_\alpha(t) \right\rangle \tag{8.64}$$

or

$$\left(\mathbf{K}^{\text{MCH}}(t)\right)_{\beta\alpha} = \frac{\partial \mathbf{R}}{\partial t} \cdot \left\langle \psi_\beta(t) \middle| \frac{\partial}{\partial \mathbf{R}} \middle| \psi_\alpha(t) \right\rangle. \tag{8.65}$$

In equation (8.64), the time-derivative couplings are directly calculated by the quantum chemistry program (use **`coupling ddt`** in the SHARC input), while in (8.65) the matrix $\mathbf{K}^{\text{MCH}}(t)$ is obtained from the scalar product of the nuclear velocity and the non-adiabatic coupling vectors (use **`coupling ddr`** in the input).

The propagation matrix can then be written as

$$\mathbf{R}^{\text{MCH}}(t + \Delta t, t) = \hat{\mathfrak{T}} \exp\left[- \int_t^{t+\Delta t} \left(\frac{\mathrm{i}}{\hbar} \mathbf{H}^{\text{MCH}}(\tau) + \mathbf{K}^{\text{MCH}}(\tau) \right) \mathrm{d}\tau \right] \tag{8.66}$$

with the time-ordering operator $\hat{\mathfrak{T}}$. For small timesteps Δt, $\mathbf{H}^{\text{MCH}}(\tau)$ and $\mathbf{K}^{\text{MCH}}(\tau)$ can be interpolated linearly

$$\mathbf{R}^{\text{MCH}}(t + \Delta t, t) = \exp\left[-\frac{1}{2} \left(\frac{\mathrm{i}}{\hbar} \mathbf{H}^{\text{MCH}}(t) + \frac{\mathrm{i}}{\hbar} \mathbf{H}^{\text{MCH}}(t + \Delta t) + \mathbf{K}^{\text{MCH}}(t) + \mathbf{K}^{\text{MCH}}(t + \Delta t) \right) \Delta t \right] \tag{8.67}$$

And in order to have a sufficiently small timestep for this to work, the interval $(t, t + \Delta t)$ is further split into subtimesteps $\Delta\tau = \frac{\Delta t}{n}$.

$$\mathbf{R}^{\text{MCH}}(t + \Delta t, t) = \prod_{i=1}^{n} \mathbf{R}_i \tag{8.68}$$

$$\mathbf{R}_i = \exp\left[- \left(\frac{\mathrm{i}}{\hbar} \mathbf{H}_i + \mathbf{K}_i \right) \Delta\tau \right] \tag{8.69}$$

$$\mathbf{H}_i = \mathbf{H}^{\text{MCH}}(t) + \frac{i}{n} \left(\mathbf{H}^{\text{MCH}}(t + \Delta t) - \mathbf{H}^{\text{MCH}}(t) \right) \tag{8.70}$$

$$\mathbf{K}_i = \mathbf{K}^{\text{MCH}}(t) + \frac{i}{n} \left(\mathbf{K}^{\text{MCH}}(t + \Delta t) - \mathbf{K}^{\text{MCH}}(t) \right) \tag{8.71}$$

8.19.2 Propagation using overlap matrices

In many situations, the non-adiabatic couplings in $\mathbf{K}^{\mathrm{MCH}}$ are very localized on the potential hypersurfaces. If this is the case, in the dynamics very short timesteps are necessary to properly sample the non-adiabatic couplings. If too large timesteps are used, part of the coupling may be missed, leading to wrong population transfer. The local diabatization algorithm gives more numerical stability in these situations. It can be requested with the line **`coupling overlap`** in the input file.

Within this algorithm, the change of the electronic states between timesteps is described by the overlap matrix $\mathbf{S}^{\mathrm{MCH}}(t, t+\Delta t)$

$$\left(\mathbf{S}^{\mathrm{MCH}}(t, t+\Delta t)\right)_{\beta\alpha} = \left\langle \psi_\beta(t) \middle| \psi_\alpha(t+\Delta t) \right\rangle \tag{8.72}$$

With this, the propagator matrix can be written as

$$\mathbf{R}^{\mathrm{MCH}}(t+\Delta t, t) = \mathbf{S}^{\mathrm{MCH}}(t, t+\Delta t)^\dagger \prod_{i=1}^{n} \mathbf{R}_i \tag{8.73}$$

$$\mathbf{R}_i = \exp\left[-\frac{\mathrm{i}}{\hbar}\mathbf{H}_i \Delta\tau\right] \tag{8.74}$$

$$\mathbf{H}_i = \mathbf{H}^{\mathrm{MCH}}(t) + \frac{i}{n}\left(\mathbf{H}^{\mathrm{MCH}}_{\mathrm{tra}} - \mathbf{H}^{\mathrm{MCH}}(t)\right) \tag{8.75}$$

$$\mathbf{H}^{\mathrm{MCH}}_{\mathrm{tra}} = \mathbf{S}^{\mathrm{MCH}}(t, t+\Delta t)\mathbf{H}^{\mathrm{MCH}}(t+\Delta t)\mathbf{S}^{\mathrm{MCH}}(t, t+\Delta t)^\dagger \tag{8.76}$$

Bibliography

[1] M. Kasha: "Characterization of electronic transitions in complex molecules". *Discuss. Faraday Soc.*, **9**, 14 (1950).

[2] C. M. Marian: "Spin-orbit coupling and intersystem crossing in molecules". *WIREs Comput. Mol. Sci.*, **2**, 187–203 (2012).

[3] I. Wilkinson, A. E. Boguslavskiy, J. Mikosch, D. M. Villeneuve, H.-J. Wörner, M. Spanner, S. Patchkovskii, A. Stolow: "Non-adiabatic and intersystem crossing dynamics in SO_2 I: Bound State Relaxation Studied By Time-Resolved Photoelectron Photoion Coincidence Spectroscopy". *J. Chem. Phys.*, **140**, 204 301 (2014).

[4] S. Mai, P. Marquetand, L. González: "Non-Adiabatic Dynamics in SO_2: II. The Role of Triplet States Studied by Surface-Hopping Simulations". *J. Chem. Phys.*, **140**, 204 302 (2014).

[5] C. Lévêque, R. Taïeb, H. Köppel: "Communication: Theoretical prediction of the importance of the $^3B_2$ state in the dynamics of sulfur dioxide". *J. Chem. Phys.*, **140**, 091101 (2014).

[6] T. J. Penfold, R. Spesyvtsev, O. M. Kirkby, R. S. Minns, D. S. N. Parker, H. H. Fielding, G. A. Worth: "Quantum dynamics study of the competing ultrafast intersystem crossing and internal conversion in the "channel 3" region of benzene". *J. Chem. Phys.*, **137**, 204310 (2012).

[7] R. A. Vogt, C. Reichardt, C. E. Crespo-Hernández: "Excited-State Dynamics in Nitro-Naphthalene Derivatives: Intersystem Crossing to the Triplet Manifold in Hundreds of Femtoseconds". *J. Phys. Chem.*, **117**, 6580–6588 (2013).

[8] C. E. Crespo-Hernández, B. Cohen, P. M. Hare, B. Kohler: "Ultrafast Excited-State Dynamics in Nucleic Acids". *Chem. Rev.*, **104**, 1977–2020 (2004).

[9] M. Richter, P. Marquetand, J. González-Vázquez, I. Sola, L. González: "Femtosecond Intersystem Crossing in the DNA Nucleobase Cytosine". *J. Phys. Chem. Lett.*, **3**, 3090–3095 (2012).

[10] L. Martínez-Fernández, L. González, I. Corral: "An Ab Initio Mechanism for Efficient Population of Triplet States in Cytotoxic Sulfur Substituted DNA Bases: The Case of 6-Thioguanine". *Chem. Commun.*, **48**, 2134–2136 (2012).

[11] S. Mai, P. Marquetand, M. Richter, J. González-Vázquez, L. González: "Singlet and Triplet Excited-State Dynamics Study of the Keto and Enol Tautomers of Cytosine". *ChemPhysChem*, **14**, 2920–2931 (2013).

[12] C. Reichardt, C. E. Crespo-Hernández: "Ultrafast Spin Crossover in 4-Thiothymidine in an Ionic Liquid". *Chem. Commun.*, **46**, 5963–5965 (2010).

[13] J. C. Tully, R. K. Preston: "Trajectory Surface Hopping Approach to Nonadiabatic Molecular Collisions: The Reaction of H^+ with D_2". *J. Chem. Phys.*, **55**, 562–572 (1971).

[14] J. C. Tully: "Molecular dynamics with electronic transitions". *J. Chem. Phys.*, **93**, 1061–1071 (1990).

[15] M. Barbatti: "Nonadiabatic dynamics with trajectory surface hopping method". *WIREs Comput. Mol. Sci.*, **1**, 620–633 (2011).

[16] N. L. Doltsinis: *Molecular Dynamics Beyond the Born-Oppenheimer Approximation: Mixed Quantum-Classical Approaches*, volume 31 of *NIC Series*. John von Neuman Institut for Computing (2006).

[17] N. L. Doltsinis, D. Marx: "First Principles Molecular Dynamics Involving Excited States And Nonadiabatic Transitions". *J. Theor. Comput. Chem.*, **1**, 319–349 (2002).

[18] S. Hammes-Schiffer, J. C. Tully: "Proton transfer in solution: Molecular dynamics with quantum transitions". *J. Chem. Phys.*, **101**, 4657–4667 (1994).

[19] G. Granucci, M. Persico: "Critical appraisal of the fewest switches algorithm for surface hopping". *J. Chem. Phys.*, **126**, 134 114 (2007).

[20] M. Thachuk, M. Y. Ivanov, D. M. Wardlaw: "A semiclassical approach to intense-field above-threshold dissociation in the long wavelength limit". *J. Chem. Phys.*, **105**, 4094–4104 (1996).

[21] B. Maiti, G. C. Schatz, G. Lendvay: "Importance of Intersystem Crossing in the S(3P, 1D) + H2 → SH + H Reaction". *J. Phys. Chem. A*, **108**, 8772–8781 (2004).

[22] G. A. Jones, A. Acocella, F. Zerbetto: "On-the-Fly, Electric-Field-Driven, Coupled Electron-Nuclear Dynamics". *J. Phys. Chem. A*, **112**, 9650–9656 (2008).

[23] R. Mitrić, J. Petersen, V. Bonačić-Koutecký: "Laser-field-induced surface-hopping method for the simulation and control of ultrafast photodynamics". *Phys. Rev. A*, **79**, 053 416 (2009).

[24] G. Granucci, M. Persico, G. Spighi: "Surface hopping trajectory simulations with spin-orbit and dynamical couplings". *J. Chem. Phys.*, **137**, 22A501 (2012).

[25] B. F. E. Curchod, T. J. Penfold, U. Rothlisberger, I. Tavernelli: "Local Control Theory using Trajectory Surface Hopping and Linear-Response Time-Dependent Density Functional Theory". *Chimia*, **67**, 218–221 (2013).

[26] G. Cui, W. Thiel: "Generalized trajectory surface-hopping method for internal conversion and intersystem crossing". *J. Chem. Phys.*, **141**, 124 101 (2014).

[27] J. Pittner, H. Lischka, M. Barbatti: "Optimization of mixed quantum-classical dynamics: Time-derivative coupling terms and selected couplings". *Chem. Phys.*, **356**, 147 – 152 (2009).

[28] M. Richter, P. Marquetand, J. González-Vázquez, I. Sola, L. González: "SHARC: ab Initio Molecular Dynamics with Surface Hopping in the Adiabatic Representation Including Arbitrary Couplings". *J. Chem. Theory Comput.*, **7**, 1253–1258 (2011).

[29] S. Mai, M. Richter, M. Ruckenbauer, M. Oppel, P. Marquetand, L. González: "SHARC: Surface Hopping Including Arbitrary Couplings – Program Package for Non-Adiabatic Dynamics". `sharc-md.org` (2014).

[30] M. Richter, P. Marquetand, J. González-Vázquez, I. Sola, L. González: "Correction to SHARC: Ab Initio Molecular Dynamics with Surface Hopping in the Adiabatic Representation Including Arbitrary Couplings?" *J. Chem. Theory Comput.*, **8**, 374–374 (2012).

[31] J. J. Bajo, J. González-Vázquez, I. Sola, J. Santamaria, M. Richter, P. Marquetand, L. González: "Mixed Quantum-Classical Dynamics in the Adiabatic Representation to simulate molecules driven by strong laser pulses". *J. Phys. Chem. A*, **116**, 2800–2807 (2011).

[32] P. Marquetand, M. Richter, J. González-Vázquez, I. Sola, L. González: "Nonadiabatic ab initio molecular dynamics including spin-orbit coupling and laser fields". *Faraday Discuss.*, **153**, 261–273 (2011).

[33] S. Mai, M. Richter, P. Marquetand, L. González: "Excitation of Nucleobases from a Computational Perspective II: Dynamics". In M. Barbatti, A. C. Borin, S. Ullrich (editors), *Photoinduced Phenomena in Nucleic Acids*, Topics in Current Chemistry, Springer Berlin Heidelberg (2014).

[34] L. González, P. Marquetand, M. Richter, J. González-Vázquez, I. Sola: "Ultrafast laser-induced processes described by ab initio molecular dynamics". In R. de Nalda, L. Bañares (editors), *Ultrafast Phenomena in Molecular Sciences*, volume 107 of *Springer Series in Chemical Physics*, 145–170, Springer (2014).

[35] G. Granucci, M. Persico, A. Toniolo: "Direct semiclassical simulation of photochemical processes with semiempirical wave functions". *J. Chem. Phys.*, **114**, 10 608–10 615 (2001).

[36] F. Plasser, G. Granucci, J. Pittner, M. Barbatti, M. Persico, H. Lischka: "Surface hopping dynamics using a locally diabatic formalism: Charge transfer in the ethylene dimer cation and excited state dynamics in the 2-pyridone dimer". *J. Chem. Phys.*, **137**, 22A514–22A514–13 (2012).

[37] J. P. Dahl, M. Springborg: "The Morse oscillator in position space, momentum space, and phase space". *J. Chem. Phys.*, **88**, 4535–4547 (1988).

[38] R. Schinke: *Photodissociation Dynamics: Spectroscopy and Fragmentation of Small Polyatomic Molecules*. Cambridge University Press (1995).

[39] M. Barbatti, G. Granucci, M. Ruckenbauer, F. Plasser, J. Pittner, M. Persico, H. Lischka: "NEWTON-X: a package for Newtonian dynamics close to the crossing seam, version 1.2". `www.newtonx.org` (2011).

[40] D. Cremer, J. A. Pople: "General definition of ring puckering coordinates". *J. Am. Chem. Soc.*, **97**, 1354–1358 (1975).

[41] J. C. A. Boeyens: "The conformation of six-membered rings". *J. Cryst. Mol. Struct.*, **8**, 317 (1978).

[42] H. Lischka, T. Müller, P. G. Szalay, I. Shavitt, R. M. Pitzer, R. Shepard: "Columbus – a program system for advanced multireference theory calculations." *WIREs Comput. Mol. Sci.*, **1**, 191–199 (2011).

[43] H. Lischka, R. Shepard, I. Shavitt, R. M. Pitzer, M. Dallos, T. Müller, P. G. Szalay, F. B. Brown, R. Ahlrichs, H. J. Böhm, A. Chang, D. C. Comeau, R. Gdanitz, H. Dachsel, C. Ehrhardt, M. Ernzerhof, P. Höchtl, S. Irle, G. Kedziora, T. Kovar, V. Parasuk, M. J. M. Pepper, P. Scharf, H. Schiffer, M. Schindler, M. Schüler, M. Seth, E. A. Stahlberg, J.-G. Zhao, S. Yabushita, Z. Zhang, M. Barbatti, S. Matsika, M. Schuurmann, D. R. Yarkony, S. R. Brozell, E. V. Beck, , J.-P. Blaudeau, M. Ruckenbauer, B. Sellner, F. Plasser, J. J. Szymczak: "COLUMBUS, an ab initio electronic structure program, release 7.0" (2012), `http://www.univie.ac.at/columbus/docs_COL70/documentation_main.html`.

[44] S. Yabushita, Z. Zhang, R. M. Pitzer: "Spin-Orbit Configuration Interaction Using the Graphical Unitary Group Approach and Relativistic Core Potential and Spin-Orbit Operators". *J. Phys. Chem. A*, **103**, 5791–5800 (1999).

[45] S. Mai, T. Müller, P. Marquetand, F. Plasser, H. Lischka, L. González: "Perturbational Treatment of Spin-Orbit Coupling for Generally Applicable High-Level Multi-Reference Methods". *J. Chem. Phys.*, **141**, 074 105 (2014).

[46] H. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz: "Molpro: a general-purpose quantum chemistry program package". *WIREs Comput. Mol. Sci.*, **2**, 242–253 (2012).

[47] H.-J. Werner, P. J. Knowles, G. Knizia, F. R. Manby, M. Schütz, et al.: "MOLPRO, version 2012.1, a package of ab initio programs". www.molpro.net (2012).

[48] G. Karlström, R. Lindh, P.-Å. Malmqvist, B. O. Roos, U. Ryde, V. Veryazov, P.-O. Widmark, M. Cossi, B. Schimmelpfennig, P. Neogrady, L. Seijo: "MOLCAS: a program package for computational chemistry". *Comput. Mater. Sci.*, **28**, 222 – 239 (2003), proceedings of the Symposium on Software Development for Process and Materials Design.

[49] F. Aquilante, L. De Vico, N. Ferré, G. Ghigo, P.-Å. Malmqvist, P. Neogrády, T. B. Pedersen, M. Pitoňák, M. Reiher, B. O. Roos, L. Serrano-Andrés, M. Urban, V. Veryazov, R. Lindh: "MOLCAS 7: The Next Generation". *J. Comput. Chem.*, **31**, 224–247 (2010), www.molcas.org.

[50] G. Granucci, M. Persico, A. Zoccante: "Including quantum decoherence in surface hopping". *J. Chem. Phys.*, **133**, 134 111 (2010).

[51] P. Marquetand: *Vectorial properties and laser control of molecular dynamics*. Ph.D. thesis, University of Würzburg (2007), http://opus.bibliothek.uni-wuerzburg.de/volltexte/2007/2469/.

[52] L. Verlet: "Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules". *Phys. Rev.*, **159**, 98–103 (1967).