Hands-On Session

Analysis of Surface Hopping Trajectories obtained with SHARC and OpenMolcas

1 What this exercise is about

- Verifying trajectories using SHARC tools
- Inspection of individual trajectories
- Statistical analysis of electronic populations
- Statistical analysis of nuclear vibrations
- Statistical analysis of time-dependent spectra
- Performing excited-state optimizations using SHARC tools

2 Introduction

We discussed the setup and execution of SHARC trajectories yesterday. Today, we will leap one step forward and actually analyze an ensemble of SHARC trajectories. These trajectories have been pre-computed to reduce the waiting time during the workshop. See below for a discussion of the system and simulations.

This task sheet is adapted from the second half of the official SHARC tutorial.

3 Description of the model system

We are still investigating the same system for which we setup trajectories yesterday—the methaniminium cation ($CH_2NH_2^+$) that was excited to the bright excited state. A swarm of trajectories was previously run for 100 fs by the instructor. The employed quantum chemistry method was CASSCF(6,4)/cc-pVDZ, using the OpenMolcas program, including 4 singlet states and 3 triplet states.

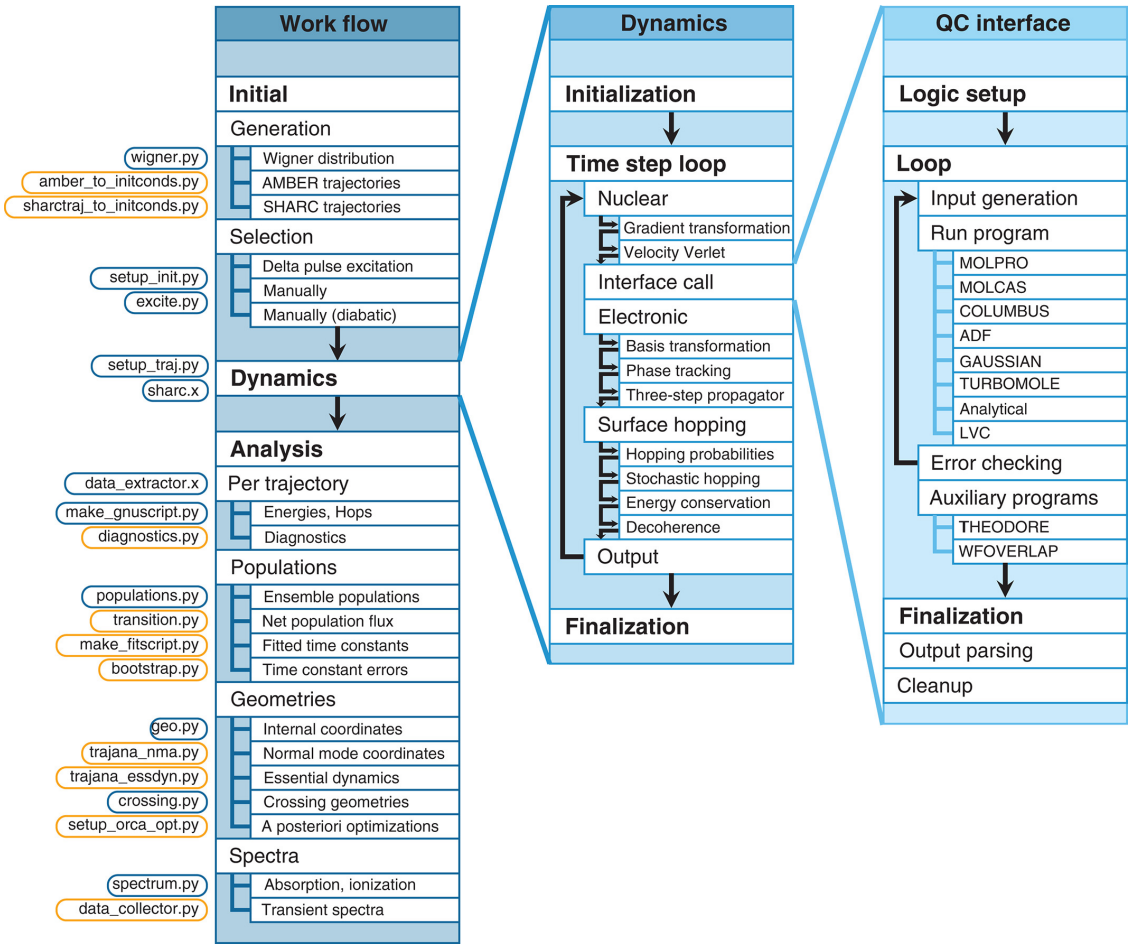

Figure 1: Mai et al., *WIREs Comput. Mol. Sci.*, 8, e1370 (2018).

4 How to use the SHARC suite

Beyond the core dynamics program, the SHARC suite contains a number of Python scripts allowing to perform various types of setup and analysis tasks. There are two types of these scripts.

Non-interactive scripts can be controlled by command-line arguments and options. Every non-interactive script can be called with the command line option `-h` in order to get a description of the functionality and possible options.

Interactive scripts ask the user for information about the task to be conducted (using features like auto-complete and default values), and only perform the task after the input has been completed.

In the tutorial, the input dialogue of the interactive scripts is shown as in this example. **Red bold text** gives the input which the user has to type.

```
Type of calculation: 2
Frequency calculation? [True] <ENTER>

Geometry filename: [geom.xyz] (autocomplete enabled) g<TAB>
Geometry filename: [geom.xyz] (autocomplete enabled) geom.xyz

Enter atom indices: (range comprehension enabled) 1~4
```

During the interactive sessions, square brackets indicate that the question has a default answer, which can be used by just pressing ENTER. If filenames or directory paths need to be entered, auto-complete is active, which can be used by pressing TAB. If a list of integers (e.g., atom indices) needs to be entered, range comprehension is active, and ranges can be entered with the tilde symbol (e.g., `1~4` is equivalent to `1 2 3 4`). Upon completion, every interactive script produces a file `KEYSTROKES.<name>`, which contains all user input for the last run.

Please make sure before starting that `$SHARC` is set to the directory containing the SHARC scripts and executables.

Thus, you should do this before starting with the exercise:

```
(base) user@node: ~> export SHARC=/projects/academic/cyberwksp21/Software/sharc2.1.1/sharc-master/bin/
(base) user@node: ~> export SCRADIR=/panfs/panfs.cbls.ccr.buffalo.edu/scratch/grp-cyberwksp21/$USER
```

Note that in principle one should be able to qualitatively reproduce the results in this tutorial. Note that it is recommended that for sections in which the user is especially interested, they should refer to the corresponding sections in the SHARC Manual.

5 Workflow

This section continues the steps from the hands-on session about setting up trajectories. Consequently, the section numbering will continue after 5.7 "Running the trajectories".

5.8 Analyzing a single trajectory

We will first discuss the analysis of a single trajectory based on the output files. Later (section 5.9) we will also analyze ensemble properties.

If you are not in the directory for the trajectory `TRAJ_00003/`, change to this directory:

```
(base) user@node: ~> cd Singlet_2/TRAJ_00003
```

5.8.1 Data extraction and plotting

The file `output.dat` of each trajectory contains the Hamiltonian, transformation matrix, dipole matrices, coefficients, hopping probabilities, kinetic energy and random number from the surface hopping procedure in a compressed form. The program `data_extractor.x` can be used to generate data tables, which can then be plotted.

```
(base) user@node: ~> module load intel-oneapi-compilers/2021.3.0 intel-oneapi-mkl/2021.3.0
(base) user@node: ~> source $SHARC/sharcvars.sh
(base) user@node: ~> $SHARC/data_extractor.x output.dat
```

The program creates a subdirectory called `output_data/`. Note that the trajectories that are provided by the instructor already have their `output_data/` directory created, so you do not necessarily need to perform this step. However, if you ran your own trajectories, you should now extract their results.

With the default settings, the following files will be created:

- `coeff_diab.out`, `coeff_class_diab.out`, `coeff_mixed_diab.out` contain the coefficients and populations in the diabatic representation (only approximate).
- `coeff_diag.out`, `coeff_class_diag.out`, `coeff_mixed_diag.out` contain the coefficients and populations in the diagonal representation.
- `coeff_MCH.out`, `coeff_class_MCH.out`, `coeff_mixed_MCH.out` contain the coefficients and populations in the MCH representation.
- `energy.out` contains kinetic, current potential, total and potential energy of all excited states.
- `fosc.out` contains the oscillator strengths of the current state and all excited states.
- `fosc_act.out` contains the oscillator strengths of all states relative to the active state.
- `spin.out` contains the total spin expectation values of the current state and all excited states.
- `prob.out` contains the surface hopping random number and the hopping probabilities in the diagonal representation.
- `expec.out` contains the content of `energy.out`, `fosc.out` and `spin.out` in one file (for plotting).
- `expec_MCH.out` is analogue to `expec.out`, except all data is given in the MCH representation (except the active state energy).

In order to plot the content of these files in an efficient manner, `gnuplot` can be used. Use

```
(base) user@node: ~> $SHARC/make_gnuscript.py  4 0 3 0 0 0 0 0 -png > plot.gp
```

to create a `gnuplot` script with the correct state numbering and labeling. Execute

```
(base) user@node: ~> gnuplot plot.gp
```

to plot energies, populations and hopping probabilities. In Figures 2, 4, 5 and 6 exemplary outputs for a trajectory for the first 100 fs are given.

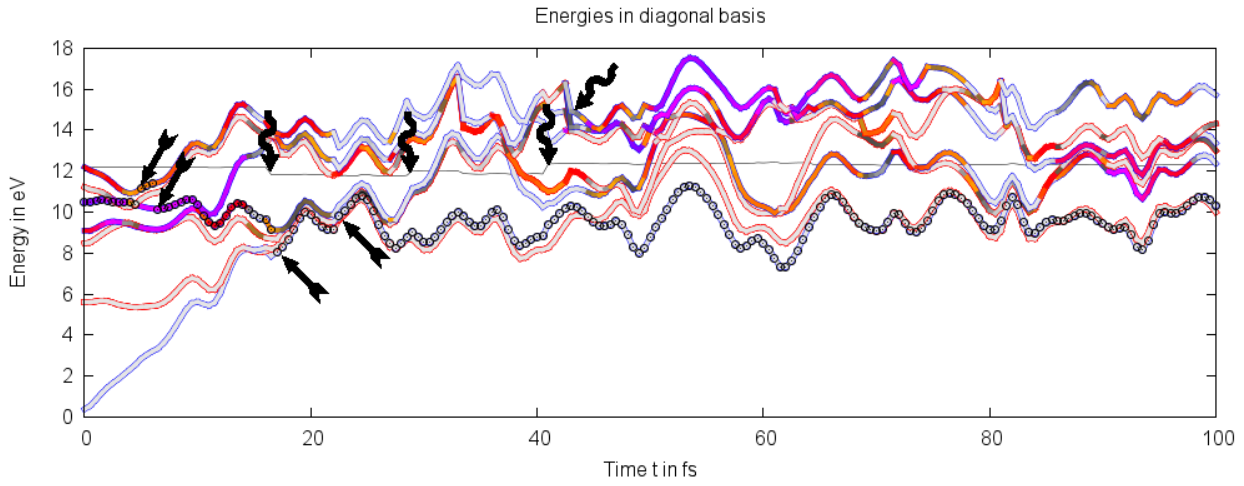

Figure 2: Plot of the potential energies for trajectory `TRAJ_00002/` with 4 singlet and 3 triplet states. Straight arrows indicate hopping events discussed in the text, wiggly arrows indicate problems with energy conservation/continuity.

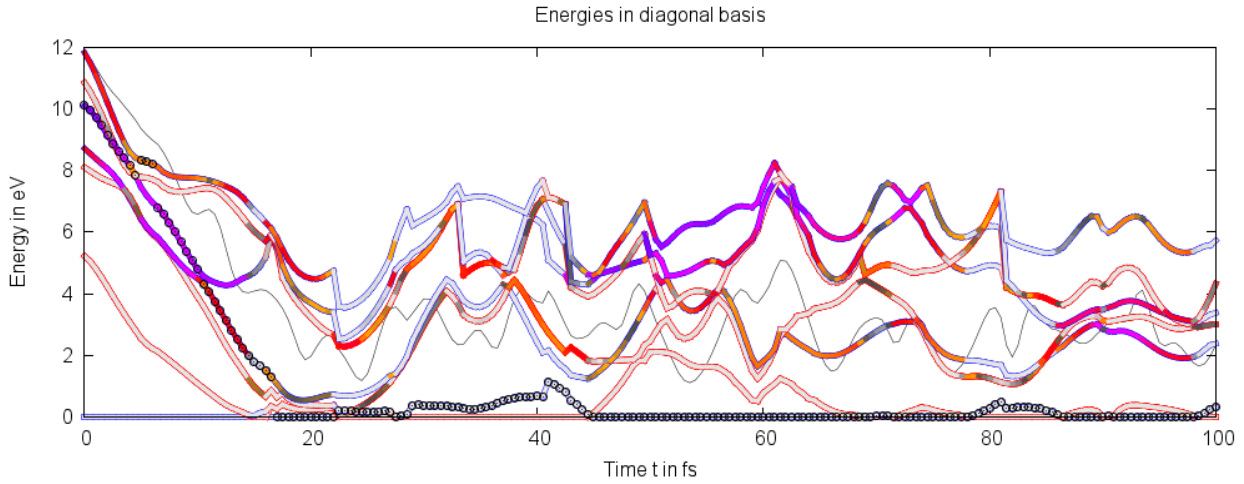

Figure 3: Plot of the *relative* potential energies for trajectory `TRAJ_00002/` with 4 singlet and 3 triplet states.

Discussion of Figure 2 In figure 2, the potential energies of all states depending on time are given. The total energy is given by the thin black line (around 12 eV), the currently occupied state is marked with black circles. Each state is represented double-colored line, with an inner core color and an outer colored contour. The inner color encodes the oscillator strength of the state at each instant of time. Dark states are light grey, while brighter states are given in grey, dark grey, orange, red, magenta or blue, in order of increasing oscillator strength. The outer color encodes the total spin expectation value. Singlets are blue, triplets red and states with mixed singlet-triplet character are green. Since in the methaniminium cation spin-orbit coupling is negligible, in the figure only blue and red contours are visible.

In the figure, the trajectory starts in the very bright singlet state slightly above 10 eV (the S_3). The ground state (grey/blue line), the T_1 and T_2 (grey/red lines), and the S_1 (red/blue line) are at lower energies, whereas the T_3 (grey/red line) and the S_4 (red/blue line) are at higher energies. All important nonadiabatic events occur within the first 25 fs. In the figure, four hopping events are marked with straight arrows (at 5.0 fs, 6.5 fs, 17.0 fs, and 22.5 fs; note that these are not the only hopping events in the figure). It can be seen that the trajectory briefly switches to the S_4 state but quickly returns to S_3. Subsequently, it changes to S_1 and

then to S_0 (at 17.0 fs). The hop at 22.5 fs is due to a crossing of the S_0 with the T_1, and since the T_1 is not populated, the trajectory hops to stay in the singlet state. For the remainder of the simulation time, the trajectory stays in S_0, occasionally performing a hop at S_0/T_1 crossings. The very high potential energy indicates that the trajectory did not simply return to the ground state equilibrium geometry.

Four wiggly arrows indicate potential problems in the trajectory. Three arrows (at 17.0 fs, 29.0 fs, and 41.0 fs) point to time steps where the total energy was not well conserved. This can have different reasons (e.g., wrong gradients, too large time steps, convergence problems), but here all three cases are due to abrupt changes in the active space because the highest singlet state crossed with another state. Often, this problem can be circumvented by a good choice of the active space and the number of roots for state averaging. The fourth wiggly arrow (at 43.0 fs) points to a time step where the same problem happens to the triplet states; note how the T_3 energy suddenly changes. Since in Molcas each multiplicity uses its own active space, this does not affect the singlet states and thus the trajectory. However, in systems with larger spin-orbit couplings, such state switches might lead to unphysical population transfer to the triplet states.

Ultimately, the user is responsible to check for such problematic time steps. In this hands-on, we will ignore these energy conservation problems. Note that this trajectory checking can also be carried out while the trajectories are still running; if a problematic trajectory is encountered, it can be terminated gracefully by creating an empty file called `STOP` in the run directory of that trajectory.

Discussion of Figure 3 In figure 3, the same data as in figure 2 is presented, the only difference being that in figure 3 all energies are relative to the energy of the lowest state. This is useful if there are strong oscillations in the energies of all states that make it difficult to see the energy gaps between the states.

Note that in this plot, the grey line represents the difference between total energy and energy of the lowest state, and hence does not need to be a straight line.

Discussion of Figure 4 In figure 4, the MCH populations are given depending on time. The system starts with 100% of the population in the S_2. Around 5 fs, part of the population is briefly transferred to the S_3. Then, at 10 fs population flows to the S_1, and at 17 fs to the S_0, where it stays for the remainder of the simulation. The population of the triplet states is approximately zero for all time steps.

It is instructive to observe the correlation between the population transfers in figure 4 with the hopping events in figure 2.

Discussion of Figure 5 In figure 5, the diagonal populations are given depending on time. The difference between the MCH and diagonal populations is due to the fact that the diagonal states are strictly ordered according to energy. This can be seen best in the second half of the figure, where population is occasionally exchanged (with 100% efficiency) between state 1 and state 4. These population transfers happen where the S_0 and T_1 states cross (e.g., before the crossing the S_0 is lower than T_1 and hence S_0 is state 1. After the crossing, S_0 becomes state 4, and the population is transferred to state 4 to conserve the spin multiplicity of the total wave function).

Note that in more complicated cases, the diagonal populations are of little use for interpretation purposes, so that most users will prefer to analyze the MCH populations.

Discussion of Figure 6 Figure 6 shows the surface hopping probabilities and the corresponding random numbers depending on time. In a nutshell, a surface hop happens whenever the random number lies within one of the colored bars. The color of the bar corresponds to the state into which the trajectory will hop. In the diagram, there are several hopping probabilities close to unity. This corresponds to the near-complete population transfer during the crossing of the singlet and triplet states.

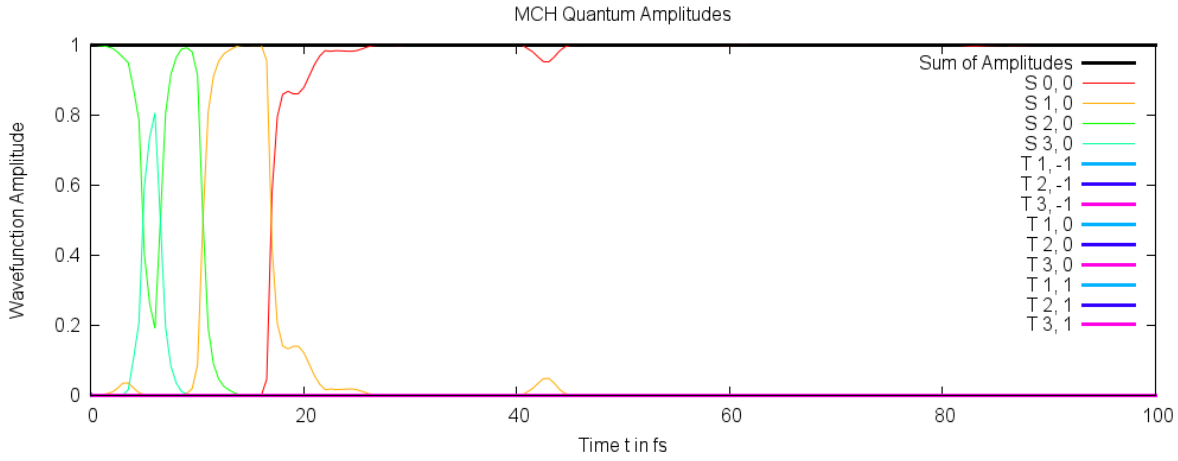

Figure 4: Plot of the excited-state populations in the MCH representation.

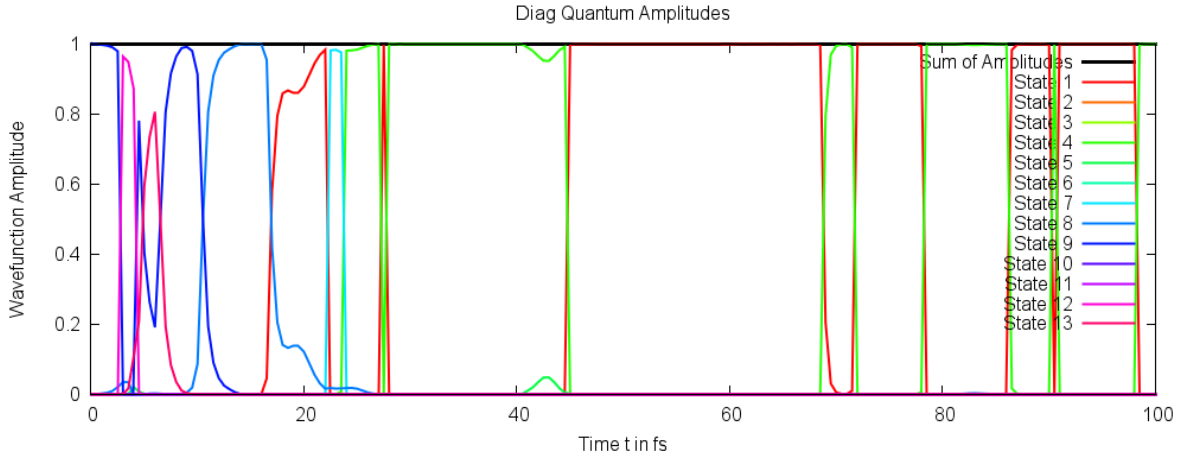

Figure 5: Plot of the excited-state populations in the diagonal representation.

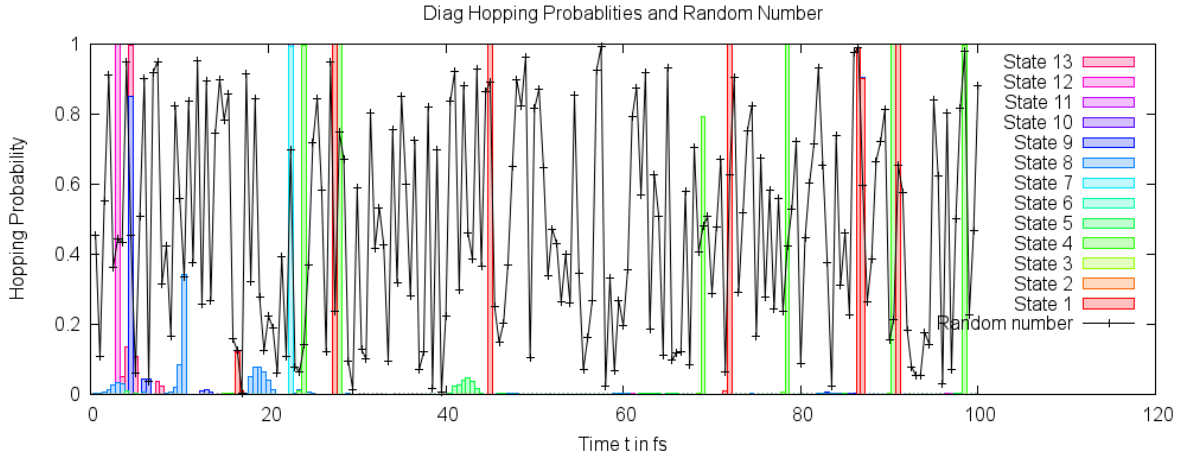

Figure 6: Plot of the hopping probabilities in the diagonal representation. Additionally, the random number for the surface hopping procedure is given.

5.8.2 Analyzing internal coordinates

The file `output.xyz` contains the cartesian coordinates of all timesteps. Oftentimes, one is interested in the variation of certain internal coordinates (like bond lengths, angles, etc.) during the dynamics. The SHARC tool `geo.py` can quickly calculate these values. Invoke the program and enter the internal coordinate specifications:

```
(base) user@node: ~> python2 $SHARC/geo.py -g output.xyz -t 0.5
```

```
Enter the internal coordinate specifications:
r 1 2
d 3 1 2 4
end
Number of internal coordinate requests:   2
Number of geometries:    200
FINISHED!
```

The `-g` option specifies the filename of the input xyz geometry file, while the `-t` option specifies the timestep. `geo.py` writes the results to standard out, so redirect the output to some file:

```
(base) user@node: ~> python2 $SHARC/geo.py -g output.xyz -t 0.5 > Geo.out
```

The file `Geo.out` contains a table with the specified internal coordinates:

```
#                1|                  2|                  3|
#             time|             r  1  2|       d  6  1  2  5|
            0.0000              1.3024              16.2632
            0.5000              1.3135              18.5857
            1.0000              1.3293              20.6828
                 ⋮                   ⋮                   ⋮
```

Use GNUPLOT to plot this table.

```
(base) user@node: ~> gnuplot
```

The file `Geo.out` contains a table with the specified internal coordinates:

```
gnuplot> set term pngcairo
gnuplot> set out "Geo-1-2.png"
gnuplot> p "Geo.out" u 1:2 w l     # Plot column 2 versus 1
gnuplot> set out "Geo-1-3.png"
gnuplot> p "Geo.out" u 1:3 w l     # Plot column 3 versus 1
```

The results are shown in figures 7 and 8.

Discussion of the internal coordinates In figures 7 the C=N bond length is plotted over time. It can be easily seen that after excitation to the $\pi\pi^*$ state, the C=N bond stretches strongly and reaches more than 3 Å after 100 fs. This is a clear sign that the $CH_2NH_2^+$ molecule is dissociating in this trajectory.

In figure 8, the dihedral angle H–C=N–H is plotted in degrees (confined to the interval $[-180°, 180°]$). It can be seen that after excitation the molecule undergoes torsion around the central bond.

In order to confirm these findings, it is recommended that you load `output.xyz` into MOLDEN (or another program) to watch the trajectory as a movie.

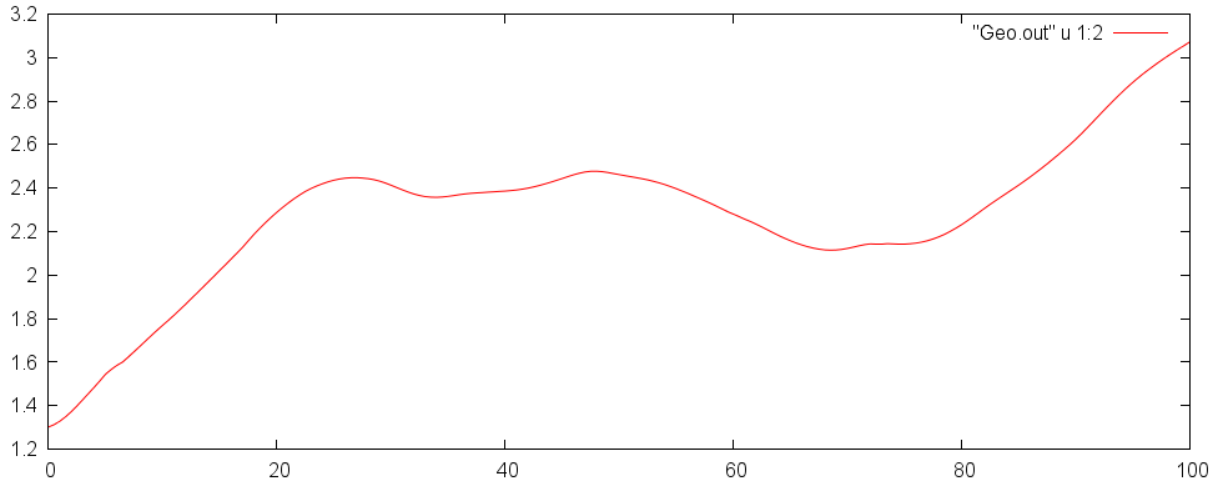

Figure 7: Value of the C=C bond length during the simulation.

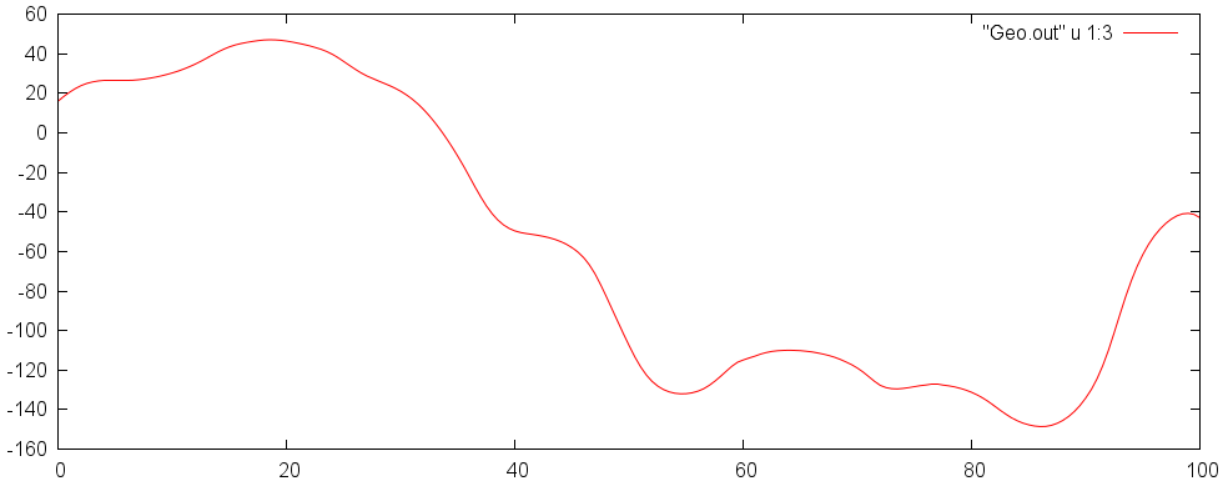

Figure 8: Value of one of the H–C=C–H dihedrals during the simulation.

5.9 Analyzing the Ensemble

For these analysis the tutorial assumes that you use the entire swarm of pre-computed trajectories. These can be found at:

```
/projects/academic/cyberwksp21/Students/smai/Instructors_material/traj/Singlet_2
```

In order to run your own extraction procedures and analysis, copy this directory to your working space with `cp -r`. Note that all trajectories in this directory were already extracted, so subdirectories `output_data` should be present. To save some disk space, the trajectories also had their restart files removed, which leads to warnings by `diagnostics.py` but does not affect your analyses.

Note that the print-outs shown below are exemplary and do not necessarily show results identical to the ones that you will see.

5.9.1 Ensemble Diagnostics

It is always a good idea to inspect the trajectories before starting with the ensemble analysis, because within the large ensemble it might not be possible to spot problems of a single trajectory. There are two ways to inspect the trajectories—either manually checking them as described above, or the ensemble diagnostics tool, `diagnostics.py`. This script performs a number of sanity checks for all trajectories (file existence, consistency, energy conservation, intruder states), and allows automatically marking problematic trajectories to exclude them from the analysis steps.

If you are still in the directory `TRAJ_00002/`, go back to the root directory of the ensemble. Then, execute `diagnostics.py`:

```
(base) user@node: ~> cd ../..
(base) user@node: ~> python2 $SHARC/diagnostics.py
```

Note that `diagnostics.py` will automatically run the data extractor for all trajectories if needed, which might take a while.

```
  ================================================================================
||                                                                              ||
||             Diagnostic tool for trajectories from SHARC dynamics             ||
||                                                                              ||
||                               Author: Sebastian Mai                          ||
||                                                                              ||
||                                    Version:2.1                               ||
||                                     01.09.19                                 ||
||                                                                              ||
  ================================================================================


This script reads output.dat files from SHARC trajectories and checks:
* missing files
* normal termination
* total energy conservation
* total population conservation
* discontinuities in potential and kinetic energy

--------------------Paths to trajectories-------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. Sing_2/ and Sing_3/.
Please enter one path at a time, and type "end" to finish the list.
```

```
Path:  [end] (autocomplete enabled) Singlet_2/
# add more paths here if you want to check them, too
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

'paths': ['Singlet_2/']
Total number of subdirectories: 9

['nstates', '4', '0', '3']
--------------------Diagnostic settings--------------------

Please, adjust the diagnostic settings according to your preferences.
You can use the following commands:
show            Prints the current settings
help            Prints explanations for the keys
end             Save and continue
<key> <value>   Adjust setting.

Current settings:
        missing_output : True
       missing_restart : True
    normal_termination : True
         always_update : False
           etot_window : 0.2
             etot_step : 0.1
             epot_step : 0.7
             ekin_step : 0.7
            pop_window : 1e-07
            hop_energy : 1.0
              intruders : True
        extractor_mode : default

?  [end] <ENTER>
#########################Full input#########################

paths                      ['Singlet_2/']
settings                   {'missing_restart': True,
                            'etot_step': 0.1,
                            'hop_energy': 1.0,
                            'epot_step': 0.7,
                            'ekin_step': 0.7,
                            'intruders': True,
                            'pop_window': 1e-07,
                            'missing_output': True,
                            'normal_termination': True,
                            'etot_window': 0.2}

Do you want to do the specified analysis? [True] <ENTER>

Checking the directories...
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ./TRAJ_00003 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    Output files:     .lis .. .log .. .dat .. .xyz .. OK
    Restart files:    ctrl .. traj .. restart/ ..     OK
    Progress:         [====                    ]     16.0 of 100.0 fs
    Status:                                          CRASHED # crashed after 16 fs (convergence problem?)
```

```
    Energy:                                         OK
    Population:                                     OK
    Intruder states:                                OK

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ./TRAJ_00004 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    Output files:     .lis .. .log .. .dat .. .xyz .. OK
    Restart files:    ctrl .. traj .. restart/ ..     OK
    Progress:         [========================]     100.0 of 100.0 fs
    Status:                                         FINISHED
    Data extractor...                               OK
    Energy:           Large dE during hop           at 13.50 fs    # a hop over >1eV might be suspicious
    Population:                                     OK
    Intruder states:                                OK

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ./TRAJ_00005 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    Output files:     .lis .. .log .. .dat .. .xyz .. OK
    Restart files:    ctrl .. traj .. restart/ ..     OK
    Progress:         [========================]     100.0 of 100.0 fs
    Status:                                         FINISHED
    Data extractor...                               OK
    Energy:           Large step in Epot            at 19.50 fs   # problem due to active space switch
    Population:                                     OK
    Intruder states:                                OK

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ./TRAJ_00006 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    Output files:     .lis .. .log .. .dat .. .xyz .. OK
    Restart files:    ctrl .. traj .. restart/ ..     OK
    Progress:         [========================]     100.0 of 100.0 fs
    Status:                                         FINISHED
    Data extractor...                               OK
    Energy:           Large step in Etot            at 15.00 fs     # time step might be too long here
    Population:                                     OK
    Intruder states:                                OK

...

================================== Summary ==================================

                   Trajectory Files? Status Length  T_use
                                              (fs)   (fs)

          ...
          Singlet_2/TRAJ_00258     OK FINISH  100.0   59.0   [==============----------]
          Singlet_2/TRAJ_00220     OK FINISH  100.0   75.0   [====================------]
          Singlet_2/TRAJ_00207     OK FINISH  100.0   84.5   [======================----]
          Singlet_2/TRAJ_00190     OK FINISH  100.0  100.0   [========================]
          Singlet_2/TRAJ_00145     OK FINISH  100.0  100.0   [========================]


This many trajectories can be used for an analysis up to the given time:
up to  20.0 fs:     52  trajectories
up to  40.0 fs:     11  trajectories
up to  60.0 fs:      4  trajectories
up to  80.0 fs:      3  trajectories
up to  100.0 fs:      2  trajectories
```

```
-------------------- Trajectory Flagging -------------------

You can now flag the trajectories according to their maximum usable time.
In this way, you can restrict the analysis tools to the set of trajectories with sufficient
simulation time.

Do you want to flag the trajectories? [True] no
```

In the output, `diagnostics.py` prints for each directory a summary of the performed checks and their results. For example, for trajectory `Singlet_2/TRAJ_00004/`, the script reports that all output files are there, and that the trajectory ran for 100 out of 100 fs (finished). It also reports that at 13.5 fs there is a problem because during a hop the potential energy changed by a large amount (it prints the first time that any problem occurs, there might be more problems occurring later). Inspecting the corresponding energy plot (analogous to Figure 2), at 13.5 fs the trajectory performed a hop from S_1 back to S_0. In general, such hops across large energy differences might be suspicious because they should be physically unlikely (nonadiabatic coupling becomes stronger the closer two states are).

For trajectory `Singlet_2/TRAJ_00005/`, the script reports that the potential energy changed by a large amount within one step. However, at the indicated time (19.5 fs), the warning comes up simply because the potential energy surface is extremely steep and the time step possibly too long to properly handle this situation. You should generate the energy plot for `Singlet_2/TRAJ_00005/` and inspect this situation. You will also see that at a later time (around 35 fs) there is a serious discontinuity in the singlet potential energy surfaces, likely due to an active space switch.

For trajectory `Singlet_2/TRAJ_00006/`, the script reports that the total energy changed too much within one step. As already discussed for Figure 2, this can happen if the active space composition changes suddenly. Another possible reason could be that the computed gradient was incorrect. It is possible to distinguish between these cases by checking whether potential and total energy both show the sudden change (then it is likely a problem with the energy computation) or whether only the total energy changes while the potential energies are smooth (then it is likely a problem with the gradients).

For trajectories `Singlet_2/TRAJ_00003/`, the script reports `CRASHED`, which is most likely due to convergence problems within Molcas. The most likely situation there is a bad convergence in the coupled-perturbed equation systems that are solved in the MCLR module during gradient evaluation. Bad convergence there is often a sign that the active space is too small and that other orbitals would like to enter the active space.

Note that a large number of reported problems is often a sign that the method is badly chosen, e.g., the active space, basis set, state-averaging, etc. These problems are typically much more common with multi-configurational/multi-reference methods (like CASSCF) than with single-reference methods (TD-DFT, ADC(2)). It is ultimately in the responsibility of the user to check and avoid these problems, or to judge whether these problems can be ignored because they do not affect the conclusions drawn from the simulations.

For the tutorial, we will from here ignore most problems by rerunning `diagnostics.py` with relaxed check thresholds.

```
(base) user@node: ~> python2 $SHARC/diagnostics.py
```

```
  ================================================================================
||                                                                              ||
||             Diagnostic tool for trajectories from SHARC dynamics             ||
||                                                                              ||
||                                Author: Sebastian Mai                         ||
```

```
||                                                                              ||
||                                  Version:2.1                                 ||
||                                   01.09.19                                   ||
||                                                                              ||
  ==============================================================================


This script reads output.dat files from SHARC trajectories and checks:
* missing files
* normal termination
* total energy conservation
* total population conservation
* discontinuities in potential and kinetic energy

--------------------Paths to trajectories-------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. Sing_2/ and Sing_3/.
Please enter one path at a time, and type "end" to finish the list.
Path:  [end] (autocomplete enabled) Singlet_2/
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'plot.gp', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

'paths': ['Singlet_2/']
Total number of subdirectories: 9

['nstates', '4', '0', '3']
--------------------Diagnostic settings--------------------

Please, adjust the diagnostic settings according to your preferences.
You can use the following commands:
show            Prints the current settings
help            Prints explanations for the keys
end             Save and continue
<key> <value>   Adjust setting.

Current settings:
        missing_output : True
       missing_restart : True
    normal_termination : Truee
         always_update : False
           etot_window : 0.2
             etot_step : 0.1
             epot_step : 0.7
             ekin_step : 0.7
            pop_window : 1e-07
            hop_energy : 1.0
             intruders : True
        extractor_mode : default

?  [end] etot_window 3.0   # extremely large thresholds
?  [end] etot_step 3.0     # to ignore most problems,
?  [end] epot_step 3.0     # don't do this in a real project
?  [end] ekin_step 3.0     #
?  [end] hop_energy 3.0    #
?  [end]
```

```
#########################Full input##########################

paths                      ['Singlet_2/']
settings                   {'missing_restart': True,
                            'etot_step': 3.0,
                            'hop_energy': 3.0,
                            'epot_step': 3.0,
                            'ekin_step': 3.0,
                            'intruders': True,
                            'pop_window': 1e-07,
                            'missing_output': True,
                            'normal_termination': True,
                            'etot_window': 3.0}

Do you want to do the specified analysis? [True]

Checking the directories...

# ... omitting trajectory summaries ...

================================== Summary ==================================

                   Trajectory Files? Status Length  T_use
                                              (fs)   (fs)

          Singlet_2/TRAJ_00010     OK  CRASH   20.5   20.0   [=====                ]
          Singlet_2/TRAJ_00009     OK  CRASH   71.5   71.0   [=================    ]
          Singlet_2/TRAJ_00004     OK FINISH  100.0  100.0   [=========================]
          Singlet_2/TRAJ_00005     OK FINISH  100.0  100.0   [=========================]
          Singlet_2/TRAJ_00006     OK FINISH  100.0  100.0   [=========================]
          Singlet_2/TRAJ_00007     OK FINISH  100.0  100.0   [=========================]
          Singlet_2/TRAJ_00002     OK FINISH  100.0  100.0   [=========================]
          Singlet_2/TRAJ_00003     OK FINISH  100.0  100.0   [=========================]
          Singlet_2/TRAJ_00008     OK FINISH  100.0  100.0   [=========================]

This many trajectories can be used for an analysis up to the given time:
up to  20.0 fs:      9  trajectories
up to  40.0 fs:      8  trajectories
up to  60.0 fs:      8  trajectories
up to  80.0 fs:      7  trajectories
up to  100.0 fs:      7  trajectories

-------------------- Trajectory Flagging --------------------

You can now flag the trajectories according to their maximum usable time.
In this way, you can restrict the analysis tools to the set of trajectories with sufficient
simulation time.

Do you want to flag the trajectories? [True] <ENTER>
Threshold for T_use (fs): [100.0] <ENTER>

Flagged  7 trajectories for analysis.
Excluded 2 trajectories from analysis.
```

With the relaxed thresholds, only very few trajectories are reported to have problems. Nevertheless, 40 trajectories are shorter than 100 fs because they crashed before. Now one has two choices—either analyze all trajectories, but only to the length of the shortest one, or neglecting trajectories so that a longer simulation time can be analyzed.

The choice suggested by `diagnostics.py` is essentially the latter, because a large number of trajectories reached at least 99 fs, giving the best possible total simulated time. All shorter trajectories are then marked by `diagnostics.py` by creating a file called `DONT_ANALYZE` in their directories. All other analysis scripts will then ignore those trajectories. With this, the ensemble is prepared for the ensemble analysis procedures.

5.9.2 Ensemble Populations

Among the main results of a SHARC simulation are the time-dependent excited-state populations within the simulated ensemble. In order to obtain these populations, the populations of all trajectories have to be summed up and normalized to the number of trajectories.

The script `populations.py` can be used to calculate various excited-state populations. There are several concepts, e.g.:

- Count, for each timestep, the number of trajectories in each classical state. These are the "classical" populations.
- For each timestep, calculate the sum of the squares of the quantum amplitudes of each state. These sums are called the "quantum" populations.
- Count, for each timestep, the number of trajectories whose expectation values are within a certain interval. This can be used to obtain populations which correspond to certain classes of states (e.g. count all trajectories with large oscillator strength to find the approximate $\pi\pi^*$ population).

Note that the rigorous computation of electronic populations including changes of representation is a complex topic, which explains the large number of possible population modes offered by `populations.py`. The modes used below (mode 3 and 9) should work in most simple cases. When spin-orbit mixing is strong or when looking into diabatic populations, consider using the Wigner-transformed populations (modes 14, 15, 22) after reading the relevant section in the manual.

In the following, an example is given on the usage of `populations.py`, and subsequently the results of using the different concepts are discussed.

```
(base) user@node: ~> python2 $SHARC/populations.py
```

```
  ================================================================================
||                                                                              ||
||                    Reading populations from SHARC dynamics                   ||
||                                                                              ||
||                             Author: Sebastian Mai                            ||
||                                                                              ||
||                                  Version:2.1                                 ||
||                                   01.09.19                                   ||
||                                                                              ||
  ================================================================================

This script reads output.lis files and calculates ensemble populations
(e.g. based on the classically occupied state or based on the quantum amplitudes).

--------------------Paths to trajectories-------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. Sing_2/ and Sing_3/.
Please enter one path at a time, and type "end" to finish the list.
Path:  [end] (autocomplete enabled) Singlet_2/
# here you can include results from other users as well by adding the paths
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'plot.gp', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

Total number of subdirectories: 9
```

```
-----------------------Analyze Mode-----------------------

This script can analyze the classical populations in different ways:
1  Number of trajectories in each diagonal state                                  from output.lis
2  Number of trajectories in each (approximate) MCH state                         from output.lis
3  Number of trajectories in each (approximate) MCH state (multiplets summed up)  from output.lis
4  Number of trajectories whose total spin value falls into certain intervals     from output.lis
5  Number of trajectories whose dipole moment falls into certain intervals        from output.lis
6  Number of trajectories whose oscillator strength falls into certain intervals  from output_data/fosc.out

It can also sum the quantum amplitudes:
7  Quantum amplitudes in diagonal picture                                  from output_data/coeff_diag.out
8  Quantum amplitudes in MCH picture                                       from output_data/coeff_MCH.out
9  Quantum amplitudes in MCH picture (multiplets summed up)                from output_data/coeff_MCH.out

It can also transform the classical diagonal populations to MCH basis:
12  Transform diagonal classical populations to MCH
                                                                           from output_data/coeff_class_MCH.out
13  Transform diagonal classical populations to MCH (multiplets summed up)
                                                                           from output_data/coeff_class_MCH.out
14  Wigner-transform classical diagonal populations to MCH
                                                                           from output_data/coeff_mixed_MCH.out
15  Wigner-transform classical diagonal populations to MCH (multiplets summed up)
                                                                           from output_data/coeff_mixed_MCH.out

It can also compute diabatic populations:
20 Quantum amplitudes in diabatic picture                                 from output_data/coeff_diab.out
21 Transform diagonal classical populations to diabatic                   from output_data/coeff_class_diab.out
22 Wigner-transform classical diagonal populations to diabatic            from output_data/coeff_mixed_diab.out
Analyze mode: 3

----------------------Number of states----------------------

Please enter the number of states as a list of integers
e.g. 3 0 3 for three singlets, zero doublets and three triplets.
Number of states: [4 0 3] <ENTER>


------------------------Normalization-----------------------

Normalize the populations? [True] <ENTER>

-----------------------Simulation time----------------------

Up to which simulation time should the analysis be performed? (Trajectories which are shorter are
continued with their last values.)
Simulation time (in fs):  [1000.0] 100

------------------Setup for bootstrapping?------------------

The population data can be analyzed by fitting with a kinetic model (via make_fitscript.py).
In order to estimate errors for these time constants (via bootstrapping),
additional data needs to be saved here.
Save data for bootstrapping? [False] yes
Directory for data? [bootstrap_data/] (autocomplete enabled) <ENTER>

-----------------------Gnuplot script-----------------------

Gnuplot script? [False] yes
```

```
Gnuplot script filename? [populations.gp] (autocomplete enabled) pop_class.gp

#########################Full input##########################

normalize                 True
paths                     ['Singlet_2/']
gnuplot_out               pop_class.gp
bootstrap                 False
gnuplot                   True
run_extractor             False
states                    [4, 0, 3]
statemap                  {1: [1, 1, 0.0, 1],
                           2: [1, 2, 0.0, 2],
                           3: [1, 3, 0.0, 3],
                           4: [1, 4, 0.0, 4],
                           5: [3, 1, -1.0, 5],
                           6: [3, 2, -1.0, 6],
                           7: [3, 3, -1.0, 7],
                           8: [3, 1, 0.0, 5],
                           9: [3, 2, 0.0, 6],
                           10: [3, 3, 0.0, 7],
                           11: [3, 1, 1.0, 5],
                           12: [3, 2, 1.0, 6],
                           13: [3, 3, 1.0, 7]}
run_extractor_full        False
mode                      3
maxtime                   100.0
nstates                   7
nmstates                  13

Do you want to do the specified analysis? [True] <ENTER>

Checking the directories...
⋮          ⋮          ⋮          ⋮          ⋮          ⋮
Number of trajectories: 7                                  # only 7 trajectories left
Found dt=0.500000, nsteps=201, nstates=7
⋮          ⋮          ⋮          ⋮          ⋮          ⋮
Shortest trajectory: 100.000000
Longest trajectory: 100.000000
Number of trajectories: 7

Writing to pop.out ...
Gnuplot script written to "pop_class.gp"
Writing to bootstrap_data/ ...
```

The incoherent sum of the quantum amplitudes can be calculated with mode 9. Rerun `populations.py`.

```
(base) user@node: ~> $SHARC/populations.py
```

```
⋮          ⋮          ⋮          ⋮          ⋮          ⋮          ⋮

------------------------Analyze Mode------------------------

⋮          ⋮          ⋮          ⋮          ⋮          ⋮          ⋮
```

```
Analyze mode: 9

Run data_extractor.x for each trajectory prior to performing the analysis?
For many or long trajectories, this might take some time.
Run data_extractor.x? [True] <ENTER>
Run data_extractor.x only if output.dat newer than output_data/ [True] <ENTER>

⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮

------------------Setup for bootstrapping?------------------

The population data can be analyzed by fitting with a kinetic model (via make_fitscript.py).
In order to estimate errors for these time constants (via bootstrapping),
additional data needs to be saved here.
Save data for bootstrapping? [False] <ENTER>

-----------------------Gnuplot script-----------------------

Gnuplot script? [False] yes
Gnuplot script filename? [populations.gp] (autocomplete enabled) pop_quant.gp

⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮

Overwrite pop.out?  [False] <ENTER>

Please enter the output filename:  (autocomplete enabled) pop_quant.out
Writing to pop_quant.out ...
```

Third, we obtain the number of trajectories whose oscillator strength falls into one of these intervals: $0 < f_{\mathrm{osc}} < 10^{-4}$, $10^{-4} < f_{\mathrm{osc}} < 1^{-1}$ and $10^{-1} < f_{\mathrm{osc}}$. Rerun `populations.py` again.

```
(base) user@node: ~> $SHARC/populations.py
```

```
⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮

-----------------------Analyze Mode-----------------------

⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮

Analyze mode: 6

Run data_extractor.x for each trajectory prior to performing the analysis?
For many or long trajectories, this might take some time.
Run data_extractor.x? [True] no          # Was already done above

⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮

-------------------------Intervals-------------------------

Please enter the interval limits, all on one line.
Interval limits:  1e-4  1e-1          # Outer limits 0 and infinity are automatically assumed

⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮
```

```
------------------Setup for bootstrapping?------------------

The population data can be analyzed by fitting with a kinetic model (via make_fitscript.py).
In order to estimate errors for these time constants (via bootstrapping),
additional data needs to be saved here.
Save data for bootstrapping? [False] <ENTER>

-----------------------Gnuplot script-----------------------

Gnuplot script? [False] yes
Gnuplot script filename? [populations.gp] (autocomplete enabled) pop_fosc.gp

⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮

Overwrite pop.out?  [False] <ENTER>

Please enter the output filename:  (autocomplete enabled) pop_fosc.out
Writing to pop_fosc.out ...
```

Use the produced GNUPLOT scripts to plot the obtained populations.

```
(base) user@node: ~> gnuplot pop_class.gp
(base) user@node: ~> gnuplot pop_quant.gp
(base) user@node: ~> gnuplot pop_fosc.gp
```

This will create the files `pop_class.gp.png`, `pop_quant.gp.png` and `pop_fosc.gp.png`. They are shown in figures 9, 10 and 11. In 9, the classical populations are given. In figure 10, the incoherent sum of the quantum amplitudes is given (obtained by using mode 9 in `populations.py`). In figure 11, the 5 trajectories are classified depending on their oscillator strengths.

Discussion of Figure 9 In figure 9 it can be seen that between 0 and 30 fs, all trajectories changed from the initial S_2 state through the S_1 state to the S_0 ground state. The triplet states remain completely unpopulated.

Discussion of Figure 10 In figure 10 the quantum populations are shown. For sufficiently large ensembles, figure 9 should closely follow figure 10. Consistency between the classical and quantum populations can be improved by using the decoherence correction (input option in `setup_traj.py`).

Discussion of Figure 11 In figure 11 the ensemble population was classified according to the oscillator strength of the classically populated state. The chosen interval limits were 0.0001 and 0.1, giving three classes of states (below 0.0001, between 0.0001 and 0.1, and above 0.1). Initially, all trajectories are in the bright $\pi\pi^*$ state and are thus classified into the third class. During the dynamics, the dissociations/torsions/hops reduce the oscillator strength, so that the trajectories are classified into the intermediate class. Later, the trajectories decay to the ground state, which by definition has an oscillator strength of zero (since f_{osc} is proportional to the excitation energy). Note that in this example the ground state cannot be distinguished from the triplet state, which has negligible oscillator strength and thus would also be classified into the first class. In general, however, classifying the population according to oscillator strength sometimes allows to approximately obtain populations of $\pi\pi^*$ and $n\pi^*$ states.

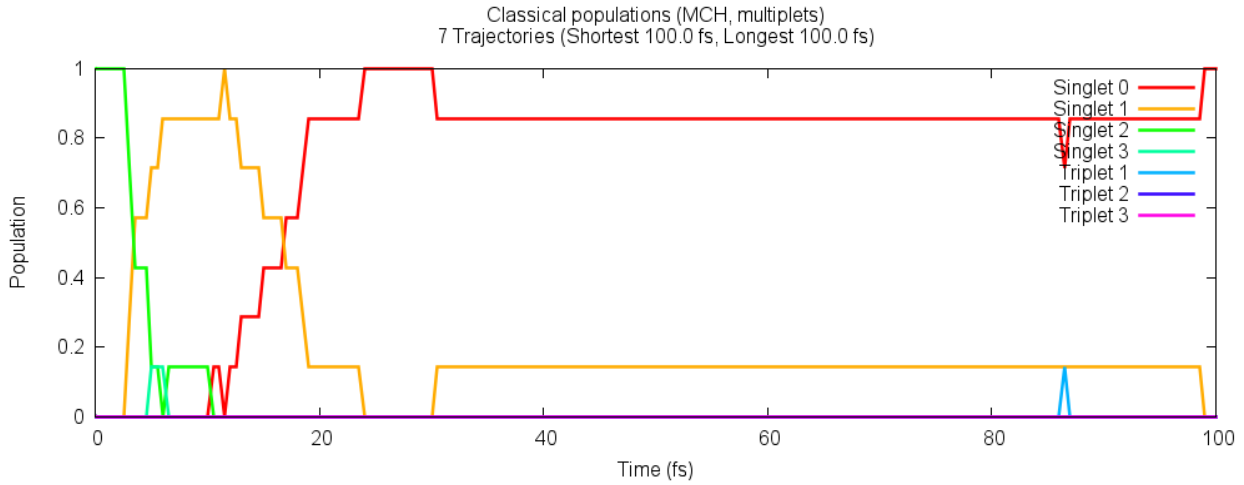

Figure 9: Classical populations for an ensemble of 7 trajectories.

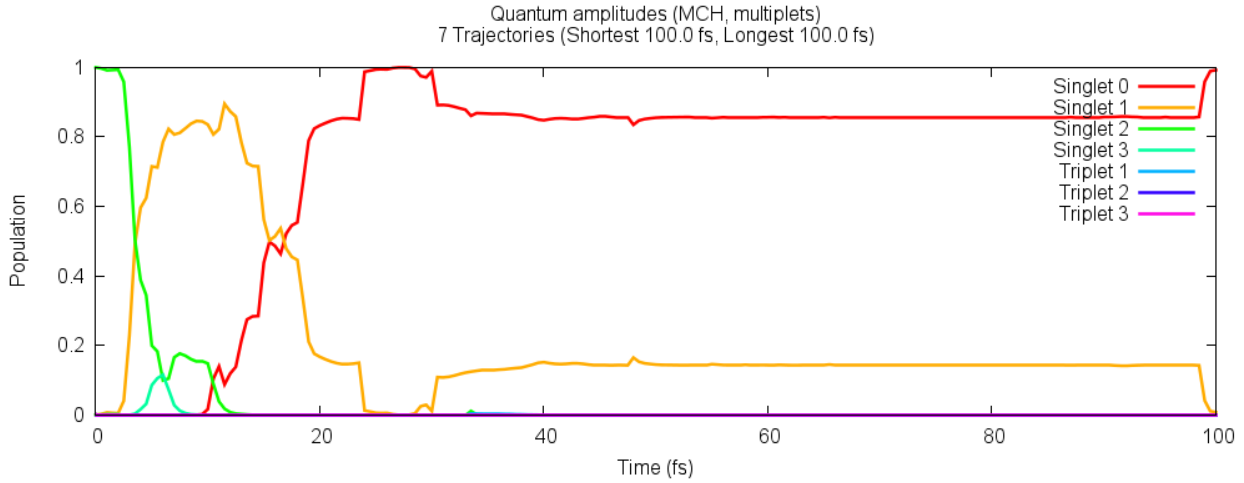

Figure 10: Quantum populations for an ensemble of 7 trajectories.

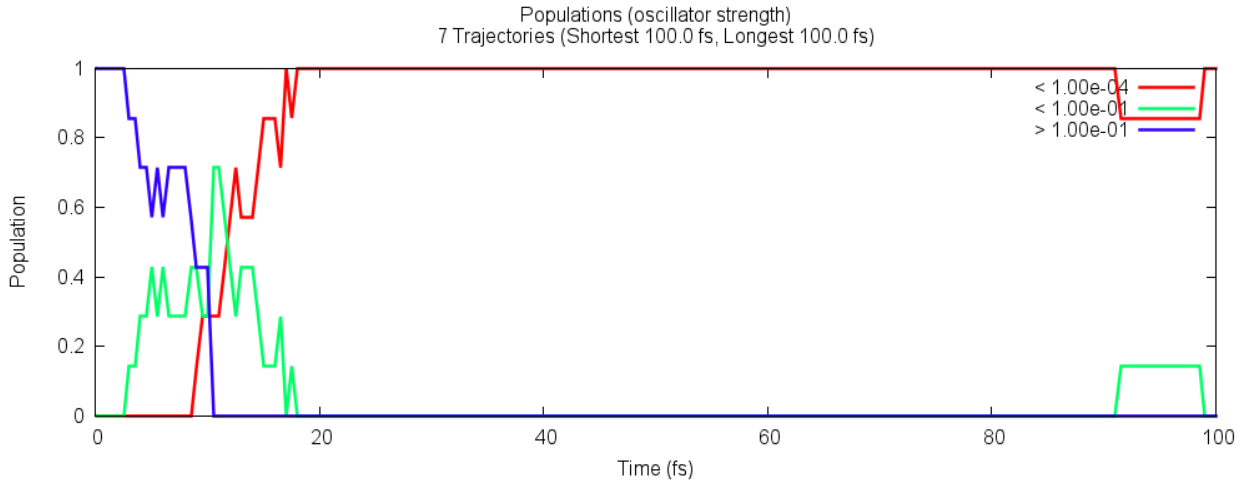

Figure 11: Populations classified based on oscillator strength for an ensemble of 7 trajectories.

5.9.3 Ensemble Populations Flow

In figure 9 it can be seen that, generally, population flows from the S_2 to the S_1 to the S_0. In order to quantify this population flow, one can use `transition.py`. This script counts the number of hops in all trajectories.

```
(base) user@node: ~> python2 $SHARC/transition.py
```

```
  ================================================================================
||                                                                              ||
||                  Counting hopping events from SHARC dynamics                 ||
||                                                                              ||
||                             Author: Sebastian Mai                            ||
||                                                                              ||
||                                  Version:2.1                                 ||
||                                   01.09.19                                   ||
||                                                                              ||
  ================================================================================

This script reads output.lis files files and counts all hopping events
to produce a matrix with the transition counts.

--------------------Paths to trajectories-------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. S_2 and S_3.
Please enter one path at a time, and type "end" to finish the list.
Path:  [end] (autocomplete enabled) Singlet_2/
# here you can include results from other users as well by adding the paths
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'plot.gp', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

Total number of subdirectories: 9

------------------------Analyze Mode------------------------

This script finds the transition matrix:
1        In MCH basis                                                  from output.lis
2        In MCH basis (ignoring hops within one multiplet)             from output.lis

This script can also print the transition matrix for each timestep:
3        In MCH basis                                                  from output.lis
4        In MCH basis (ignoring hops within one multiplet)             from output.lis
5        In MCH basis [cumulative]                                     from output.lis
6        In MCH basis [cumulative] (ignoring hops within one multiplet)  from output.lis

Analyze mode: 2

----------------------Number of states----------------------

Please enter the number of states as a list of integers
e.g. 3 0 3 for three singlets, zero doublets and three triplets.
Number of states: [4 0 3] <ENTER>

----------------------Simulation time----------------------

Up to which simulation time should the analysis be performed?
```

```
Simulation time (in fs):  [1000.0] 100


#########################Full input##########################

paths                     ['Singlet_2/']
run_extractor             False
states                    [4, 0, 3]
mode                      2
maxtime                   100.0
nstates                   7
nmstates                  13

Do you want to do the specified analysis? [True] <ENTER>

Checking the directories...
⋮           ⋮           ⋮           ⋮           ⋮           ⋮           ⋮
Number of trajectories: 164
Number of steps: 201

**************************Results**************************
Full transition matrix:
        |      S0      S1      S2      S3      T1      T2      T3
--------+-----------------------------------------------------------
  S0    |   16593     194       0       0       8       3       0
  S1    |      71    8103     200       2       0       4       0
  S2    |       0      41    3243      12       0       0       1
  S3    |       0       2      12      94       0       0       0
  T1    |       8       0       0       0      26       2       0
  T2    |       2       6       0       0       1      37       0
  T3    |       0       0       1       0       0       0       1

Sum transition matrix:
        |      S0      S1      S2      S3      T1      T2      T3
--------+-----------------------------------------------------------
  S0    |   16593     265       0       0      16       5       0
  S1    |       0    8103     241       4       0      10       0
  S2    |       0       0    3243      24       0       0       2
  S3    |       0       0       0      94       0       0       0
  T1    |       0       0       0       0      26       3       0
  T2    |       0       0       0       0       0      37       0
  T3    |       0       0       0       0       0       0       1

Difference transition matrix:
        |      S0      S1      S2      S3      T1      T2      T3     Sum
--------+-------------------------------------------------------------------
  S0    |       0     123       0       0       0       1       0     124
  S1    |    -123       0     159       0       0      -2       0      34
  S2    |       0    -159       0       0       0       0       0    -159
  S3    |       0       0       0       0       0       0       0       0
  T1    |       0       0       0       0       0       1       0       1
  T2    |      -1       2       0       0      -1       0       0       0
  T3    |       0       0       0       0       0       0       0       0
  Sum   |    -124     -34     159       0      -1       0       0       0
```

The most important matrix in the output is the difference transition matrix (the last one), which shows the "net" hops. In the matrix, the previous state is given in the columns and the new state in the rows. In our example, it shows that there were 159 net hops from the S_2 to the S_1 and 123 net hops from the S_1 to the S_0. No net hops occurred directly between the S_2 and S_0, and only very few hops involved the triplet states (2 from S_1 to T_2, one from T_2 to T_1, and one from T_2 to S_0). The S_3 and T_3 states were not involved at all. In summary, the population flow in the ensemble is clearly $S_2 \rightarrow S_1 \rightarrow S_0$.

5.9.4 Fitting Ensemble Populations including Error Estimation

The results in Figure 9 show that internal conversion is an ultrafast process in $CH_2NH_2^+$. However, in order to facilitate comparison to experiments it is useful to obtain a time constant for the relaxation processes. SHARC allows fitting population data to combinations of unimolecular reactions, using `make_fit.py`. Here, we will fit the population data to the kinetic model $S_2 \rightarrow S_1 \rightarrow S_0$, which was the result of the population flow analysis.

Besides simply performing the fit, we will also obtain error estimates of the fitted time constants with the same script using the bootstrapping method. In this method, based on the original ensemble (of N trajectories), "resample" ensembles are generated by randomly drawing *with replacement* trajectories from the original ensemble (here, drawing N random trajectories). Each resample is then fitted in exactly the same way as the original ensemble, and after many resamples a distribution of possible fitting parameters is obtained. From this distribution, one can then find error measures for the fitting parameters.

In order to start the script, run:

```
(base) user@node: ~> conda activate py27
(base) user@node: ~> python2 $SHARC/make_fit.py
```

```
  ================================================================================
||                                                                              ||
||                     Direct fitting for SHARC populations                     ||
||                                                                              ||
||                              Author: Sebastian Mai                           ||
||                                                                              ||
||                                   Version:2.1                                ||
||                                    01.09.19                                  ||
||                                                                              ||
  ================================================================================

############################################################
######################Kinetics Model########################
############################################################

-----------------------Model Species-----------------------

First, please specify the set of species used in your model kinetics.

Possible input:
+ <label> <label> ...   Adds one or several species to the set
- <label> <label> ...   Removes one or several species from the set
show                    Show the currently defined set of species
end                     Finish species input

Each label must be unique. Enter the labels without quotes.

Input: [end] + S0 S1 S23     # multiple labels can be added
  Species 'S0' added!
  Species 'S1' added!
  Species 'S23' added!                              # sum of S2 and S3 for simplicity
Input: [end] <ENTER>

Final species set:  ['S0', 'S1', 'S23']


-----------------Model Elementary Reactions-----------------

Second, please specify the set of elementary reactions in your model kinetics.
```

```
Possible input:
+ <species1> <species2> <rate_label>  Add a reaction from species1 to species2 with labelled rate constant
- <rate_label>                        Remove the reaction with the given rate constant
show                                  Show the currently defined set of reactions (as adjacency matrix)
end                                   Finish reaction input

Each rate label must be unique.

Input: [end] + S23 S1 k21    # one reaction at a time
  Reaction from 'S23' to 'S1' with rate label 'k21' added!
Input: [end] + S1 S0 k10     # one reaction at a time
  Reaction from 'S1' to 'S0' with rate label 'k10' added!
Input: [end] <ENTER>

Final reaction network:
     |  S0   S1  S23
-----+---------------
  S0 |   .    .    .
  S1 | k10    .    .
 S23 |   .  k21    .

(Initial species: rows; Final species: columns)

------------------Model Initial Conditions------------------

Third, please specify species with non-zero initial populations.

Possible input:
+ <species>       Declare species to have non-zero initial population
- <species>       Remove species from the set of non-zero initial populations
show              Show the currently defined non-zero initial populations
end               Finish initial condition input

Input: [end] + S23         # initial population is in S2
  Species 'S23' added!
Input: [end] <ENTER>

Final initial species set:  ['S23']

############################################################
###################### Fitting Data ########################
############################################################


----------------------Operation mode----------------------

This script can work with the following output:
* pop.out (file from populations.py)
* bootstrap_data/ (directory from populations.py)
Using only the pop.out allows fitting and obtaining time constants.
Using the bootstrap data instead additionally allows for realistic error estimates.

Do you want to use bootstrap data? [False] yes
How many bootstrap samples? [100] <ENTER>


--------------------Population data file--------------------

Please specify the path to the bootstrap data directory (as generated by populations.py).
```

```
Bootstrap data directory: [bootstrap_data/] (autocomplete enabled) <ENTER>
  Detected maximal time of   100.0 fs and 8 columns (time plus 7 data columns).

Do you want to write fitting curves for all bootstrap cycles? [False] <ENTER>

------------Population-to-Species Mapping for Fit-----------

Please specify which model species should be fitted to which data file columns.
For example, you can fit the label 'S0' to column 2:
  S0 = 2
You can also fit the sum of two species to a column:
  T1 T2 = 5
You can also fit a species to the sum of several columns:
  T_all = 5 6 7
You can even fit a sum of species to a sum of columns:
  T1 T2 = 5 6 7

On the right side, "~" can be used to indicate ranges:
  T1 T2 = 5~9

Possible input:
<species1> <species2> ... = <columns1> <column2> ...            Set one mapping
show                                                            Show mapping
end                                                             Finish mapping input
reset                                                           Redo the mapping input

Each species label must be used at most once.
Each column number (except for '1', which denotes the time) must be used at most once.

Set of species:        ['S0', 'S1', 'S23']
Set of column numbers: [2, 3, 4, 5, 6, 7, 8]
Input: [end] S0 = 2       # fit S0 to column 2 data
Input: [end] S1 = 3       # fit S1 to column 3 data
Input: [end] S23 = 4 5    # fit S23 to sum of column 4 and 5
Input: [end] <ENTER>
Final mappings:
     S0  =  2
     S1  =  3
     S23  =  4  5

############################################################
#################### Fitting procedure #####################
############################################################

-------------Initial guesses------------

Please check the initial guesses for the parameters

Possible input:
label = value     Set an initial guess (detects type automatically and computes k=1/t for rates)
show              Show the currently defined non-zero initial populations
end               Finish initial condition input

  time constant ( k10          ):     100.0000 fs
  time constant ( k21          ):     120.0000 fs
  initial pop   ( S23          ):       1.0000

Input: [end] <ENTER>
Final guess parameters:
```

```
  time constant ( k10          ):     100.0000 fs
  time constant ( k21          ):     120.0000 fs
  initial pop   ( S23          ):       1.0000

------Optimize initial populations------

Do you want to optimize the initial populations (otherwise only the rates)? [True] no

--------Constrained optimization--------

Do you want to restrict all rates/initial populations to be non-negative? [True] <ENTER>

#########################Full input##########################

bootstrap_cycles           100
bounds                     True
columns_groups             [[2], [3], [4, 5]]
data                       [ ... ]
do_bootstrap               True
initial                    [2]
initset                    set(['S23'])
maxtime                    100.0
ncol                       8
ngroups                    3
ninitial                   1
nrates                     2
nspec                      3
ntraj                      7
opt_init                   False
p0                         [0.01, 0.008333333333333333]
popfile                    /user/mai/NewSHARC/SHARC_2.0/TUTORIAL/2_full/Tutorial/traj/bootstrap_data
rank                       0
rate_matrix                [['', '', ''], ['k10', '', ''], ['', 'k21', '']]
ratemap                    0: 'k10', 1: 'k21', 'k10': 0, 'k21': 1
rates                      [[(1, 0)], [(2, 1)]]
rateset                    set(['k10', 'k21'])
species_groups             [['S0'], ['S1'], ['S23']]
specmap                    0: 'S0', 1: 'S1', 2: 'S23', 'S1': 1, 'S0': 0, 'S23': 2
summation                  [[0], [1], [2]]
write_bootstrap_fits       False
y0                         [1.0]

Do you want to continue? [True] <ENTER>

########################## Fitting ##########################

-------------- Iterations --------------

   Iteration     Total nfev        Cost      Cost reduction    Step norm     Optimality
       0              1         6.5508e+01                                    4.24e+03
       1              2         3.0090e+01      3.54e+01       1.30e-02       1.67e+03
       2              3         7.1759e+00      2.29e+01       2.54e-02       4.03e+02
       3              4         2.7432e+00      4.43e+00       2.21e-02       1.06e+02
       4              5         1.4844e+00      1.26e+00       2.22e-02       2.94e+01
       5              6         1.2319e+00      2.53e-01       1.57e-02       9.33e+00
       ⋮              ⋮                ⋮              ⋮              ⋮              ⋮
      11             12         1.2019e+00      4.46e-09       3.17e-06       4.66e-04
`ftol` termination condition is satisfied.
Function evaluations 12, initial cost 6.5508e+01, final cost 1.2019e+00, first-order optimality 4.66e-04.
```

```
----------- Final parameters -----------

time constant ( k10          ):      33.0313 fs +/-       0.4253 fs (   1.29 %)
time constant ( k21          ):       9.5498 fs +/-       0.2122 fs (   2.22 %)
initial pop   ( S2           ):       1.0000


These time constants include errors that assume that the population data
is free of uncertainty. Bootstrapping analysis is following now.

Raw data and fitted functions written to "fit_results.txt".

GNUPLOT script written to "fit_results.gp".

###################### Bootstrapping ######################

 Cycle          k10          k21          S23  Time
     1      33.1210       4.2232       1.0000  0:00:00.573773
     2      16.7930       3.7431       1.0000  0:00:00.681325
     3      10.7393       5.2129       1.0000  0:00:00.600873
     4      27.8368       3.4376       1.0000  0:00:00.804838
     5      11.7184       4.6316       1.0000  0:00:00.920739
     :            :            :            :               :
   100      49.7747       3.8927       1.0000  0:00:00.634471

>>>>>>>>>>>>> Finished the bootstrapping cycles ...

-------------------------------- Analysis for time constant "k10" --------------------------------

    Arithmetic analysis:               33.0876 +/-       2.9930
                                            ( +/-       9.05 %)

    Geometric analysis:                32.9550  +        3.1022  -        2.8353
                                            (  +        9.41 %  -        8.60 %)

    Minimum and maximum:               26.0917      and            41.1558

    Histogram:
    ==========
                                  #                              | 27
                                  #                              | 24
                                  #                              | 22
                                  #                              | 20
                                  #     #                        | 18
                            #     #     #     #                  | 15
                            #     #     #     #                  | 13
                            #     #     #     #                  | 11
                            #     #     #     #                  | 9
                      #     #     #     #     #                  | 6
                      #     #     #     #     #           #      | 4
                      #     #     #     #     #     #     #      | 2
        |     |     |     |     |     |     |     |     |     |
    25.16 26.56 28.03 29.58 31.22 32.96 34.78 36.71 38.75 40.90
```

```
-------------------------------- Analysis for time constant "k21" --------------------------------

    Arithmetic analysis:                9.6652 +/-       0.9573
                                            ( +/-       9.90 %)

    Geometric analysis:                 9.6180  +        1.0103  -        0.9143
                                            (  +       10.50 %  -        9.51 %)

    Minimum and maximum:                7.2120      and            12.3217

    Histogram:
    ==========
                                  #                             | 27
                                  #                             | 24
                                  #                             | 22
                                  #     #                       | 20
                                  #     #     #                 | 18
                                  #     #     #                 | 15
                                  #     #     #                 | 13
                            #     #     #     #                 | 11
                            #     #     #     #                 | 9
                            #     #     #     #     #           | 6
                #     #     #     #     #     #     #           | 4
                #     #     #     #     #     #     #           | 2
        |     |     |     |     |     |     |     |     |     |
    7.128 7.568 8.035 8.532 9.058 9.618 10.21 10.84 11.51 12.22


-------------------------------- Analysis for initial population "S23" --------------------------------

    Arithmetic analysis:                1.0000 +/-       0.0000
                                            ( +/-       0.00 %)

    Geometric analysis:                 1.0000  +        0.0000  -       -0.0000
                                            (  +        0.00 %  -       -0.00 %)

    Minimum and maximum:                1.0000      and             1.0000

    Histogram:
    ==========
     #                                                          | 100
     #                                                          | 91
     #                                                          | 83
     #                                                          | 75
     #                                                          | 66
     #                                                          | 58
     #                                                          | 50
     #                                                          | 41
     #                                                          | 33
     #                                                          | 25
     #                                                          | 16
     #                                                          | 8
        |     |     |     |     |     |     |     |     |     |
    1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000 1.000
```

```
----------- Final parameters -----------

time constant ( k10          ):      33.0313 fs +/-       2.9930 fs (   9.06 %)
time constant ( k21          ):       9.5498 fs +/-       0.9573 fs (  10.02 %)
initial pop   ( S2           ):       1.0000

Output (analysis and full fitted data) written to "fit_bootstrap.txt".
```

Unlike the fitting scripts used in the previous SHARC release, `make_fit.py` directly carries out the kinetic model fit in one program, without calling external software to solve differential equations or perform the non-linear fits. It is therefore much faster, and additionally allows fitting of models that are too complex to solve analytically.

The main results of the script can be found under `Final parameters`, obtained after 11 iterations. The table presents the fitted values of all parameters. It can be seen that the $S_2 \longrightarrow S_1$ time constant is 9.5 fs, and the time constant for $S_1 \longrightarrow S_0$ is 33.0 fs.

For visual inspection of the fit results, the script writes two new files, `fit_results.txt` and `fit_results.gp`. The former is a simple text file that contains three columns. The first column is the time axis of the global fit, i.e., the time axis of the data, but continued to accommodate all data sets to be fitted (in the present case, there are three data sets to be fitted as defined in the Population-to-species mapping). The second column is the data and the third column is the fitted kinetic model functions. This file can be conveniently plotted with GNUPLOT:

```
(base) user@node: ~> gnuplot fit_results.gp
```

The result of this plot is shown in Figure 12.

Discussion of Figure 12 From the figure, it can be seen that the $S_2 \longrightarrow S_1$ time constant is 4.2 fs, and the time constant for $S_1 \longrightarrow S_0$ is 16.5 fs. Note that the fit is not very good, because the number of trajectories is low and the relaxation processes in $CH_2NH_2^+$ are so fast that first-order reactions are not optimal to describe them. For slower processes in larger molecules, these kinetic model fits might work better.

Discussion of the bootstrapping results Because we provided bootstrapping data to the script, after the main data fit the script automatically continues the run and performs the requested number of bootstrap cycles. In each cycle, it will write the parameters fitted for the current cycle. Note that if the bootstrapping iterations take too long and the results seem to be converged already, pressing `Ctrl+C` allows skipping the remaining iterations and directly leads to the final analysis.

The result of the bootstrapping procedure is presented in a summary for each fitting parameter. Note that by default also the initial populations are treated as fitting parameters, even if they are fixed in the shown example.

The results show that the $S_2 \longrightarrow S_1$ relaxation process in the trajectories had a time constant of 9.7±1.0 fs (using the arithmetic analysis). Most fits yielded a time constant between 9 and 11 fs.

For the $S_1 \longrightarrow S_0$ process, the result is 33.1±3.0 fs. According to the histogram, most of the fits yielded a value between 30 and 37 fs, although there were many fits with values below 30 fs and even some above 40 fs. This narrow distribution leads to the small error found. Clearly, the errors are affected by the number of trajectories. Also note that the distribution of this parameter is only slightly skewed (with a right tail), leading to a good agreement of the fitted value from the main data (33.0 fs) and the average of the bootstrap results (33.1 fs). If these values deviate, one might want to combine the main data value with the error from bootstrapping (33.0±3.0 fs) or use the results of the geometric analysis (which gives $33.0^{+3.1}_{-2.8}$ fs).

Generally, the errors get smaller as more trajectories are employed, and hence, the fitting errors are a good tool to judge whether enough trajectories were computed. For some processes, it might also be necessary

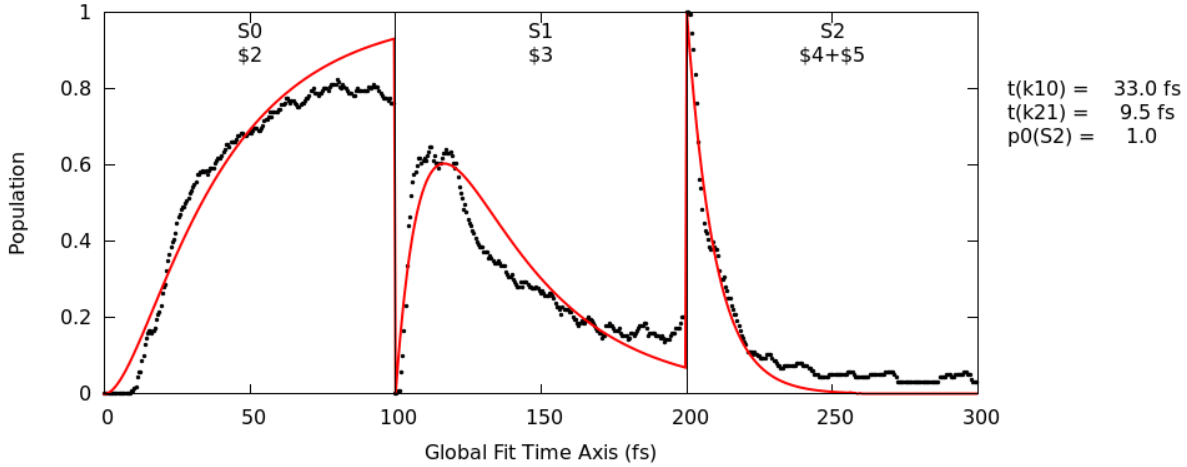

Figure 12: Kinetic model fit of the classical populations.

to run the trajectories for longer time to reduce the errors. In any case, the obtained errors tell nothing about the inherent method error—using surface hopping in combination with a given quantum chemistry method. It is not possible to quantify this method error with `make_fit.py`; only through comparison with reference data or experiment can the method error be judged.

5.9.5 Hopping Geometries

Another aspect one might be interested in are certain critical geometries from the trajectories. `crossing.py` is a script that collects those geometries from all trajectories where a surface hop between two specified states occurred. Its usage is comparable to `populations.py`.

```
(base) user@node: ~> python2 $SHARC/crossing.py
```

```
  ================================================================================
||                                                                              ||
||                 Reading hopping geometries from SHARC dynamics               ||
||                                                                              ||
||                              Author: Sebastian Mai                           ||
||                                                                              ||
||                                   Version:2.1                                ||
||                                    01.09.19                                  ||
||                                                                              ||
  ================================================================================


This script reads output.lis files and output.xyz files to produce a list of
all geometries where certain surface hops (or other events) occured.

--------------------Paths to trajectories-------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. S_2 and S_3.
Please enter one path at a time, and type "end" to finish the list.
Path:  [end] (autocomplete enabled) Singlet_2/
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'plot.gp', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

Total number of subdirectories: 9

------------------------Analyze Mode------------------------

This script can find geometries where:
1        A change of MCH state occured (ignoring hops within one multiplet)              from output.lis

Analyze mode: 1

----------------------Number of states----------------------

Please enter the number of states as a list of integers
e.g. 3 0 3 for three singlets, zero doublets and three triplets.
Number of states: [4 0 3] <ENTER>


---------------States involved in surface hop---------------

In this analysis mode, all geometries are fetched where a trajectory switches from a given MCH state
to another given MCH state.

Please enter the old MCH state involved as "mult state", e.g., "1 1" for S0, "1 2" for S1, or "3 1" for T1:
State 1: 1  2      # Only hops from S1
```

```
Please enter the new MCH state involved (mult state):
State 2: 1  1      # to S0

Direction:
1       Forwards      # Only S1 -> S0
2       Backwards     # Only S0 -> S1
3       Two-way       # Both S1 -> S0 and S0 -> S1

Direction mode: [3] 1

#########################Full input##########################

paths                       ['Singlet_2/']
tostates                    [[1, 1], [1, 2]]
run_extractor               False
states                      [4, 0, 3]
mode                        1
dirmode                     1
nstates                     7
fromstates                  [[1, 1], [1, 2]]
nmstates                    13

Do you want to do the specified analysis? [True] <ENTER>

Checking the directories...
       ⋮

Writing to crossing.xyz ...
```

The script writes a files called `crossing.xyz`, which contains all geometries (9 geometries in this example) where a hop from the S_1 to the S_0 occurred. This file can in turn be analyzed with `geo.py` in order to calculate internal coordinates (e.g., to find whether a bond or angle controls access to the S_1/S_0 crossing seam, or how many different pathways allow this transition). If you do this, you should find that most S_1/S_0 hops occurred at rather long C=N bond lengths and with a twisted molecule, where some of the HCNH dihedrals are close to ±90°.

5.9.6 Ensemble Motion Plots

In addition to the statistical analysis of the nuclear motion (`trajana_essdyn.py` and `trajana_nma.py`), it is often helpful to plot some bond length, angle, internal coordinate, etc. for all trajectories, in what could be called "hair figures" or heat maps. For such plots (and many others), `data_collector.py` can merge the corresponding per-trajectory data into files which can then be conveniently plotted.

In the following, we will collect the time-dependent C=N bond length from all trajectories and plot them. The first step is to compute this bond length for each trajectory. This can be achieved with `geo.py` and a simple Bash loop:

```
(base) user@node: ~> echo "r 1 2" > Geo.inp
(base) user@node: ~> for i in Singlet_2/TRAJ_000*;
(base) user@node: ~> python2 $SHARC/geo.py -t 0.5 -g $i/output.xyz < Geo.inp > $i/Geo.out;
(base) user@node: ~> done
```

This will create a file `Geo.out` for each trajectory with the bond length between atoms 1 and 2.

Then, start the `data_collector.py` to merge the data into a single file (note that the excluded trajectories are ignored):

```
(base) user@node: ~> python2 $SHARC/data_collector.py
```

```
  ================================================================================
||                                                                              ||
||                    Reading table data from SHARC dynamics                    ||
||                                                                              ||
||                             Author: Sebastian Mai                            ||
||                                                                              ||
||                                  Version:2.1                                 ||
||                                   01.09.19                                   ||
||                                                                              ||
  ================================================================================


This script collects table data from SHARC trajectories, smooths them, synchronizes them,
convolutes them, and computes averages and similar statistics.

--------------------Paths to trajectories------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. Sing_2/ and Sing_3/.
Please enter one path at a time, and type "end" to finish the list.
Path:  [end] (autocomplete enabled) Singlet_2/
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'plot.gp', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

Total number of subdirectories: 9

Checking the directories...
Number of trajectories: 7
Checking for common files...

List of files common to the trajectory directories:

 Index Number of appearance   Relative file path
```

```
------------------------------------------------------------
     0                    7   ./Geo.out
     1                    7   ./nma_nma.txt
     2                    7   ./nma_nma_av.txt
     3                    7   ./nma_nma_std.txt
     4                    7   ./output.lis
     5                    7   output_data/coeff_MCH.out
     6                    7   output_data/coeff_diab.out
     7                    7   output_data/coeff_diag.out
     8                    7   output_data/energy.out
     9                    7   output_data/expec.out
    10                    7   output_data/expec_MCH.out
    11                    7   output_data/fosc.out
    12                    7   output_data/fosc_act.out
    13                    7   output_data/prob.out
    14                    7   output_data/spin.out

Please give the relative file path of the file you want to collect:
File path or index: [0] ./Geo.out

------------------------Data columns------------------------

Number of columns in the file:   2

Please select the data columns for the analysis:
For T column:
  only enter one (positive) column index.
  If 0, the line number will be used instead.
For X column:
  enter one or more column indices.
  If 0, all entries of that column will be set to 1.
  If negative, the read numbers will be multiplied by -1.
For Y column:
  enter as many column indices as for X.
  If 0, all entries of that column will be set to 1.
  If negative, the read numbers will be multiplied by -1.

T column (time): [1] <ENTER>
X columns: [2] (range comprehension enabled) <ENTER>
Y columns: [0] (range comprehension enabled) <ENTER>
Selected columns:
T: 1     X: [2]    Y: [0]

----------------------Analysis procedure----------------------

Show possible workflow options? [True] no

---------------1 Smoothing--------------

Do you want to apply smoothing to the individual trajectories? [False] <ENTER>

-------------2 Synchronizing------------

Do you want to synchronize the data? [True] <ENTER>


----------3 Convoluting along X---------

Do you want to apply convolution in X direction? [False] yes
```

```
Choose one of the following convolution kernels:
1  Gaussian function
2  Lorentzian function
3  Rectangular window function
4  Log-normal function
Choose one of the functions: [1] <ENTER>
Choose width of the smoothing function (in units of the X columns): [1.0] 0.2
Size of the grid along X: [25] 50

Choose minimum and maximum of the grid along X:
Enter either a single number a (X grid from  xmin-a*width  to  xmax+a*width)
        or two numbers a and b (X grid from  a  to  b)
Xrange: [1.5] <ENTER>

------------6 Sum over all Y------------

Do you want to sum up all Y values? [False] <ENTER>

-----------7 Integrate along X----------

Do you want to integrate in X direction? [False] <ENTER>

----------8 Convoluting along T---------

Do you want to apply convolution in T direction? [False] <ENTER>

----------9 Integrating along T---------

Do you want to integrate in T direction? [False] <ENTER>

-------10 Convert to Type2 dataset------

If you performed integration along X, the data might be better formatted as Type2 dataset.
Do you want to output as Type2 dataset? [False] <ENTER>


#########################Full input##########################

paths                     ['Singlet_2/']
statistics                {}
allfiles                  ['Singlet_2//TRAJ_00002/./Geo.out',
                           'Singlet_2//TRAJ_00003/./Geo.out',
                           'Singlet_2//TRAJ_00004/./Geo.out',
                           'Singlet_2//TRAJ_00005/./Geo.out',
                           'Singlet_2//TRAJ_00006/./Geo.out',
                           'Singlet_2//TRAJ_00007/./Geo.out',
                           'Singlet_2//TRAJ_00008/./Geo.out']
colX                      [2]
filepath                  ./Geo.out
averaging                 {}
colT                      1
colY                      [0]
synchronizing             True
smoothing                 {}
nX                        1
nY                        1
convolute_T               {}
ncol                      2
type3_to_type2            False
```

```
integrate_X               {}
sum_Y                     False
integrate_T               False
convolute_X               {'function': <__main__.gauss instance at 0x1a7ed88>,
                           'npoints': 50, 'xrange': [1.5]}

Do you want to do the specified analysis? [True] <ENTER>

>>>>>>>>>>>>>>>>>>>> Started data analysis

Collecting the data ...
  Progress: [==================================================] 100%
>>>> Writing output to file "collected_data_1_2_0.type1.txt"...

Synchronizing temporal data ...
  Progress: [==================================================] 100%
>>>> Writing output to file "collected_data_1_2_0_sy.type2.txt"...

Convoluting data (along X column) ...
  Progress: [==================================================] 100%
>>>> Writing output to file "collected_data_1_2_0_sy_cX.type3.txt"...
```

This run of the script produces three output files:

- `collected_data_1_2_0.type1.txt`,
- `collected_data_1_2_0_sy.type2.txt`, and
- `collected_data_1_2_0_sy_cX.type3.txt`.

You can plot the contained data using Gnuplot:

```
gnuplot> p for [i=2:165] "collected_data_1_2_0_sy.type2.txt" u 1:i w l lc rgb "black"
```

For the 3D data in `collected_data_1_2_0_sy_cX.type3.txt`, use:

```
gnuplot> set view map
gnuplot> sp "collected_data_1_2_0_sy_cX.type3.txt" u 1:2:3 w pm3d
```

The results of these two commands are shown in Figures 13 and 14.

Using `data_collector.py`, it is also possible to compute the mean and standard deviations of the bond lengths. In order to do so, do not apply convolution in X direction, and then ask for an averaging and/or statistics analysis.

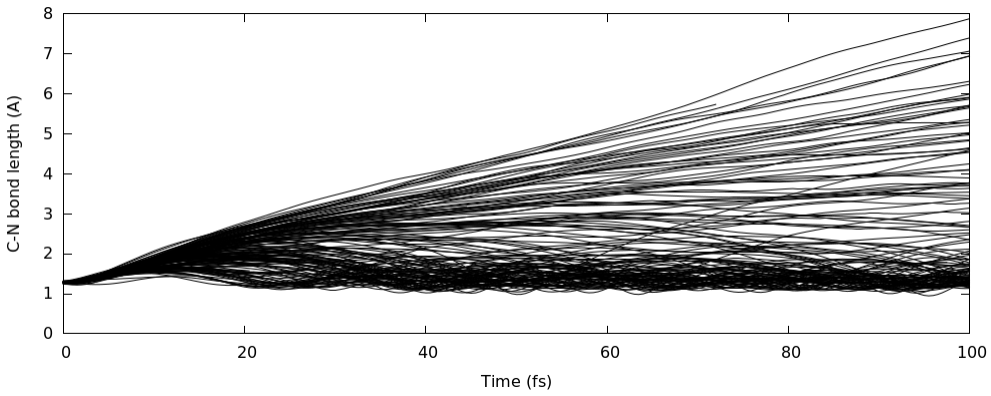

Figure 13: All C=N bond lengths from the 7 trajectories.

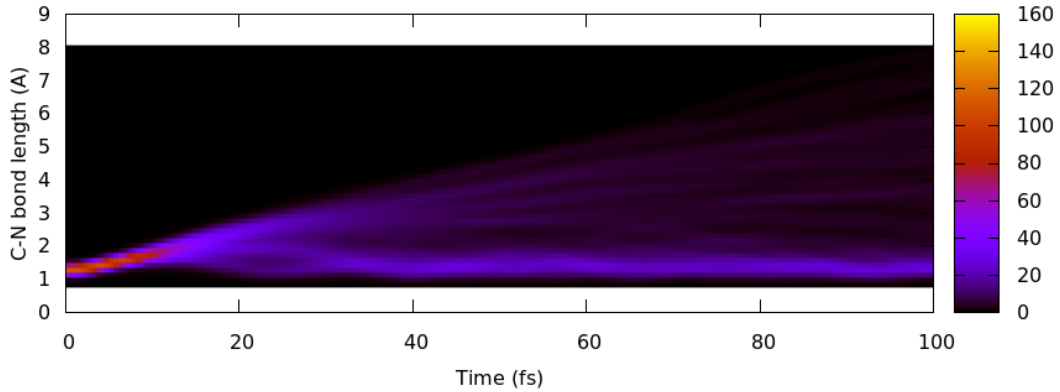

Figure 14: Convolution of the C=N bond lengths from the 7 trajectories.

5.9.7 Transient Spectra

Using `data_collector.py`, it is also possible to compute transient absorption spectra, given that enough states were included in the simulations. Here, we will simulate the transient spectrum for $CH_2NH_2^+$, although probably the spectrum will be incomplete due to the missing of higher excited states.

Again, start the `data_collector.py` to merge and post-process the data:

```
(base) user@node: ~> $SHARC/data_collector.py
```

```
  ================================================================================
||                                                                              ||
||                    Reading table data from SHARC dynamics                    ||
||                                                                              ||
||                             Author: Sebastian Mai                            ||
||                                                                              ||
||                                  Version:2.1                                 ||
||                                   01.09.19                                   ||
||                                                                              ||
  ================================================================================


This script collects table data from SHARC trajectories, smooths them, synchronizes them,
convolutes them, and computes averages and similar statistics.

--------------------Paths to trajectories-------------------

Please enter the paths to all directories containing the "TRAJ_0XXXX" directories.
E.g. Sing_2/ and Sing_3/.
Please enter one path at a time, and type "end" to finish the list.
Path:  [end] (autocomplete enabled) Singlet_2/
['TRAJ_00005', 'TRAJ_00004', 'TRAJ_00008', 'plot.gp', 'TRAJ_00009', 'TRAJ_00002',
'TRAJ_00006', 'TRAJ_00010', 'TRAJ_00007', 'TRAJ_00003']
Found 9 subdirectories in total.

Path:  [end] (autocomplete enabled) <ENTER>

Total number of subdirectories: 9

Checking the directories...
Number of trajectories: 7
Checking for common files...

List of files common to the trajectory directories:

 Index Number of appearance   Relative file path
------------------------------------------------------------
     0                    7   ./Geo.out
     1                    7   ./nma_nma.txt
     2                    7   ./nma_nma_av.txt
     3                    7   ./nma_nma_std.txt
     4                    7   ./output.lis
     5                    7   output_data/coeff_MCH.out
     6                    7   output_data/coeff_diab.out
     7                    7   output_data/coeff_diag.out
     8                    7   output_data/energy.out
     9                    7   output_data/expec.out
    10                    7   output_data/expec_MCH.out
    11                    7   output_data/fosc.out
    12                    7   output_data/fosc_act.out
```

```
    13                    7   output_data/prob.out
    14                    7   output_data/spin.out

Please give the relative file path of the file you want to collect:
File path or index: [0] output_data/fosc_act.out

------------------------Data columns------------------------

Number of columns in the file:   27

Please select the data columns for the analysis:
For T column:
  only enter one (positive) column index.
  If 0, the line number will be used instead.
For X column:
  enter one or more column indices.
  If 0, all entries of that column will be set to 1.
  If negative, the read numbers will be multiplied by -1.
For Y column:
  enter as many column indices as for X.
  If 0, all entries of that column will be set to 1.
  If negative, the read numbers will be multiplied by -1.

T column (time): [1] <ENTER>
X columns: [2] (range comprehension enabled) 2~14
Y columns: [0 0 0 0 0 0 0 0 0 0 0 0] (range comprehension enabled) 15~27
Selected columns:
T: 1     X: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
         Y: [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]

----------------------Analysis procedure----------------------

Show possible workflow options? [True] no

---------------1 Smoothing--------------

Do you want to apply smoothing to the individual trajectories? [False] <ENTER>

-------------2 Synchronizing------------

Do you want to synchronize the data? [True] <ENTER>

----------3 Convoluting along X---------

Do you want to apply convolution in X direction? [False] yes

Choose one of the following convolution kernels:
1  Gaussian function
2  Lorentzian function
3  Rectangular window function
4  Log-normal function
Choose one of the functions: [1] <ENTER>
Choose width of the smoothing function (in units of the X columns): [1.0] <ENTER>
Size of the grid along X: [25] 50

Choose minimum and maximum of the grid along X:
Enter either a single number a (X grid from  xmin-a*width  to  xmax+a*width)
        or two numbers a and b (X grid from  a  to  b)
Xrange: [1.5] <ENTER>
```

```
------------6 Sum over all Y------------

Do you want to sum up all Y values? [False] yes

-----------7 Integrate along X----------

Do you want to integrate in X direction? [False] <ENTER>

----------8 Convoluting along T---------

Do you want to apply convolution in T direction? [False] <ENTER>

----------9 Integrating along T---------

Do you want to integrate in T direction? [False] <ENTER>

-------10 Convert to Type2 dataset------

If you performed integration along X, the data might be better formatted as Type2 dataset.
Do you want to output as Type2 dataset? [False] <ENTER>


########################Full input#########################

paths                     ['Singlet_2/']
statistics                {}
allfiles                  ['Singlet_2//TRAJ_00002/output_data/fosc_act.out',
                           'Singlet_2//TRAJ_00003/output_data/fosc_act.out',
                           'Singlet_2//TRAJ_00004/output_data/fosc_act.out',
                           'Singlet_2//TRAJ_00005/output_data/fosc_act.out',
                           'Singlet_2//TRAJ_00006/output_data/fosc_act.out',
                           'Singlet_2//TRAJ_00007/output_data/fosc_act.out',
                           'Singlet_2//TRAJ_00008/output_data/fosc_act.out']
colX                      [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
filepath                  output_data/fosc_act.out
averaging                 {}
colT                      1
colY                      [15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
synchronizing             True
smoothing                 {}
nX                        13
nY                        13
convolute_T               {}
ncol                      27
type3_to_type2            False
integrate_X               {}
sum_Y                     True
integrate_T               False
convolute_X               {'function': <__main__.gauss instance at 0x28e0c20>,
                           'npoints': 50, 'xrange': [1.5]}

Do you want to do the specified analysis? [True] <ENTER>



>>>>>>>>>>>>>>>>>>>> Started data analysis

Collecting the data ...
  Progress: [==================================================] 100%
```

```
>>>> Writing output to "collected_data_1_234567891011121314_151617181920212222324252627.type1.txt"...

Synchronizing temporal data ...
  Progress: [==================================================] 100%
>>>> Writing output to "collected_data_1_234567891011121314_151617181920212222324252627_sy.type2.txt"...

Convoluting data (along X column) ...
  Progress: [==================================================] 100%
>>>> Writing output to "collected_data_1_234567891011121314_151617181920212222324252627_sy_cX.type3.txt"...

Summing all Y values ...
  Progress: [==================================================] 100%
>>>> Writing output to "collected_data_1_234567891011121314_151617181920212222324252627_sy_cX_sY.type3.txt"...
```

The last of the four produced output files contains the total transient absorption spectrum. You can print it in the same way as the convoluted bond lengths in Figure 14. The corresponding `gnuplot` code is:

```
set xlabel "Time (fs)"
set ylabel "Energy (eV)"
set cblabel "Intensity"
set palette defined (-1 "red", 0 "white", 1 "blue")
set cbrange [-5:5]
set view map
unset key
set term pngcairo size 1000,400
set out "spec.png"
sp "collected_data_1_234567891011121314_151617181920212222324252627_sy_cX_sY.type3.txt"  u 1:2:3 w pm3d
```

The simulated transient absorption spectrum is presented in Figure 15. As you can see, there is a strong excited-state emission at early times, where many trajectories are in the excited state, with an energy that decreases with time due to nuclear motion. At longer times, the trajectories are back in the ground state and absorb in a broad energy range, because the molecules have a lot of energy and thus vibrate/dissociate.

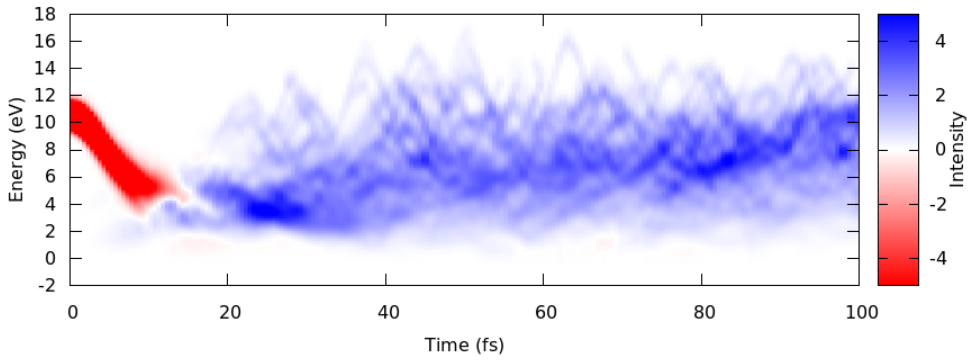

Figure 15: Simulated transient absorption spectrum of $CH_2NH_2^+$. Ground state bleech and excited-state fluorescence (negative absorption) are shown in red, and excited-state absorption (positive absorption) is shown in blue.