Internship Report: Tipsi Tutorial and Documentation

Author: Dylan Goldsborough
Date: July 12, 2016

Internship Goals

The primary goal of this 6 ECTS internship was to produce a tutorial for the TIPSI package, developed by Tsjerk Wassenaar [2]. This application of transition path sampling (TPS) was completed by Tsjerk as a PhD, but documentation was never written.

TPS is a Monte Carlo sampling method of the path space: it samples MD trajectories that connect two stable states of a molecular system, A and B, in an unbiased fashion. TPS selects a point in this trajectory, the shooting point, from which a 'shooting move' is made. In a shooting move, a new MD simulation is started with the shooting point as a starting state. A shooting move can be backward in time or forward in time in the case of one-way shooting, or both in the case of two way shooting. This shooting move replaces the original trajectory in the case of two-way shooting, and the part before or after the shooting point in the case of one-way shooting (depending on the direction of the shooting move). A shooting move can be accepted or rejected. If the new trajectory does not connect the two stable states A and B, the new shooting move is instantly rejected. If the two states are connected, the Metropolis acceptance rule

$$P_{acc} = \min(1, \frac{N^{(o)}}{N^{(n)}}) \tag{1}$$

where P_{acc} denotes the probability of accepting the new path, $N^{(o)}$ is the length of the original path, and $N^{(n)}$ is the length of the new path. To avoid accepting paths that are too long, a second stage is introduced in the acceptance process: a maximum pathlength $N_{\max}$ is defined before the shooting procedure starts, and defined as

$$N_{\max} = \frac{N^{(o)}}{\xi} \tag{2}$$

where ξ is a uniformly distributed random number between 0 and 1. New paths longer than the maximum length are rejected by default. If the shooting move is rejected, a new move is performed.

The two stable states A and B, as well as the transition region I, are defined using order parameters. These order parameters are properties of the molecular system that can be expressed numerically for every timestep. For instance, it is possible to define the stable states using the distance between two specific atom (to test whether a hydrogen bond exists), by examining the radius of gyration to examine the compactness of the molecule, or by tracking a dihedral angle.

Ideally the shooting point is chosen in such a way that the 'committor probability' is 0.5: if you start an MD simulation at this frame, there is a 50% chance to reach both state A and state B. A new shooting point is then picked from the trajectory of the shooting move, and the process is repeated. Generally the shooting point is chosen at random, although methods are available to steer the shooting point towards a point where the committor probability is a half. After several shooting moves, new paths can be constructed that go from A to B, and are decorrelated from the original trajectory. A TPS algorithm that performs a forward and backward shooting move every step is labeled as a two-way shooting algorithm, whereas those that do a forward or backward shooting are labelled one-way shooting algorithms. TIPSI belongs to the latter category, and only does one move per step, randomly picking forward or backward at each attempt. The one-way shooting procedure is used in TIPSI as it improves the acceptance ratio of the shooting moves.

To make the package accessible to use for students and members of the computational chemistry group that did not have hands-on experience with the package yet, I was instructed to write a tutorial that introduces TIPSI through one or more examples, and provide some minimal documentation. The documentation itself is not aimed to have full coverage, but sufficient coverage to allow users to run TIPSI without having to dive into the source code themselves. The tutorial and documentation is to be available on GIT (`https://github.com/dgoldsb/tipsitutorial`).

A secondary goal was to write a few scripts that can help users work with TIPSI. The likely application of these scripts was some form of processing of the output of TIPSI. This goal was considered optional, as the need (or lack thereof) for these tools would become apparent in the process of developing the tutorial.

Realization

The contributions from this intership fall in three categories. Each of these categories will be discussed, to show the methods in developing this product, to discuss what goal this helps fullfill, and to discuss some of the pitfalls and solutions found for these problems. The categories are as follows:

- Bash and Python scripts for processing TIPSI OUTPUT
- A tutorial with accompanying Python script
- Minimal documentation on setting up and running TIPSI

Some miscelleaneous facts that were not widely known in the group were also encountered, which will be mentioned in a final subsection.

Bash and Python scripts

A major challenge in TIPSI is the assembly of the transition paths it generates. Applications of TPS, such as TIPSI, use an input trajectory that describes the transition from a state A to a state B of a molecular system, and generates many more trajectories. To save space, TIPSI only stores the shooting moves, as storing the full trajectories would mean that frames are stored multiple times. Up until this point, no tool had been written to actually assemble the output of TIPSI into the full trajectories.

Assembling the trajectories was an unforeseen challenge, due to some of the properties of the molecular dynamics engine that is used: GROMACS [1]. This software is only designed for simulations that go forward in time, as it was not build with TPS in mind. TIPSI works around this by using a negative timestep in the shooting moves that go backkward in time. However, the resulting trajectory cannot be processed by GROMACS due to the decreasing timestamps. This problem is worsened by the fact that TIPSI produces negative timestamps. Negative numbers are used in the GROMACS command line parameters as a null-indicator, making it impossible to retrieve frames with a negative timestamp. As a result of this, it is not possible to simply turn the order of the frames in backward shooting moves around to process them.

This problem was solved through a Bash script, which has been appended to the TIPSI runscript. This way, the Bash script is automatically ran before the data is copied back from the computing node to the directory of the user. The script follows the following outline in its functioning: This

Algorithm 1 Assembling Transition Paths

```
1: regex ← regex extracting timestamp and framesource
2: for directories with ACCEPTED do
3:     for each line in dat-file do
4:         apply regex to line
5:         overwrite timestamps from framesource
6:         extract frame from framesource
7:         append frame to output
8:     store data on trajectory to csv
9:     save output
10: save csv
```

is a highly simplified version of the script, as finding which frame to extract after overwriting the

timestamps is not straightforward. It was decided to make each complete trajectory start at $t = 0$, and use the same timestep as the original trajectory.

A Python script to analyze the `csv` that this script outputs was also produced. The analysis is primarily meant to give the user a good idea of what the shooting process has looked like, and spot if something went wrong. This script outputs the average length of the transition paths, the number of decorrelated paths, and the ratio between succesfull forward and backward shooting moves. In addition, it draws a plot which visualizes the shooting process using a Python drawing tool. A minor drawback of this visualization method is that this script, unlike all other scripts produced in this internship, cannot be run on a cluster, as this package does not run when monitor output is not available. This should not be a problem, however, as visualization of the trajectory is also done on a personal computer.

Tutorial

A tutorial that contains two examples was written, as well as a brief introduction to the general idea of TPS. The first example is designed to have a low computational time, and be easy to analyze. The seconds example was selecte to be more demanding, but to be an actual example of a rare event. In the repository containing the tutorial, I also included some sample output for students to reference their results with.

For the first example, the molecule alanine dipeptide was chosen. This compound has two distinct states, between which transitions occur. These two states can be described using two dihedral angels, and are visually distinguishable. To give the student the complete picture of what working with TIPSI entails, this example lets the student start from scratch. We start with a file describing the structure of alanine dipeptide, and files containing settings for molecular dynamics simulations. We prepare the simulation by placing the molecule in a solvent, in a periodic box. We do two regular simulations with this system: one at room temperature (298K), and one at high temperature (400K). The former is used to describe the stable states, and to see if these states are stable enough. A Python script which produces plots, from which the ranges between which the system is in a stable state can be read for two order parameters, is included with the tutorial. This information is used to set the state definitions in the TIPSI parameter file. The high temperature simulation is used to provide an initial transition to TIPSI, as a higher temperature can force a transition.

For the second example, a project by two members of the computational chemistry group (Jocelyne Vreede and David Swenson) was used. This project concerned the transition in a DNA basepair from the Watson-Crick configuration to the Hoogsteen configuration. This event is actually rare, as it does not occur in normal molecular dynamics simulations of a length up to microseconds (approximately the upper limit of simulation lengths at this moment). Jocelyne and David have obtained several transitions already by forcing the transition using a metadynamics simulation, and have already defined the states. As such, this example is less work for the student: all that is left is setting up a file that contains the conditions for the simulations TIPSI produces, and setting up TIPSI itself. With this example, students are shown that TIPSI does what it promises. In addition, it shows students what the shooting process is like in a more complex molecular system; in the first example the shooting process is nice and symmetrical, whereas in this example this is not the case due to the increased complexity of the system.

Documentation

For documentation, three different pieces were written. Two of these are aimed at students in the biomolecular simulations course, and contain a 'cheat sheet' for Linux and GROMACS commands. This was included some that took the course in the current academic year where unfamiliar with one or both. The third piece is documentation for TIPSI, which documents its use and options.

First, it is discussed in this documentation what the running parameters of TIPSI are. This provides some insight in how to change the number of shooting moves made, for instance, or how to include a GROMACS index file. It is also described how to set up the parameter file, the second part of the input that TIPSI requires. In this file, the two states are defined according to some order

parameters. Specific atoms are referred to by their atom number, as defined in the `gro`-file used to set up the simulation.

In this part, a major indexing issue in TIPSI was encountered. This issue cannot properly be resolved, but the turotial clearly warn users, as this has to be kept in mind while writing the parameter file. If not, the TPS process is guaranteed to fail. Declaring groups of atoms on the parameter file is done using numpy-arrays. Python, as most programming languages, is zero-indexed, meaning that arrays start counting at zero. This means that, when the list of all atoms are stored in Python, the first atom is labeled '0'. GROMACS, however, is one-indexed, meaning that the first atom in `gro`-files is labeled '1'. To add to the confusion, there is the possibility to read in groups of atoms from index files. In this case the atom numbers are one-indexed as well. As such, to make sure that the atom numbers match the correct atom, 1 has to be subtracted whenever entering atom numbers directly into the parameter file.

Finally, the section of the TIPSI source code that does order parameter calculations was analyzed. From this analysis, a comprehensive list of all the possible functions that can be used to define states was compiled. In each case the highest level function to define an order parameter was included, as it was found that in some of the existing parameter files functions were used that were not meant to be called directly. For instance, `pdist` is an often-used order parameter, as it finds the distance in a periodic box. However, it turns out that the function `dist` can be called with a running parameter to use a periodic box, which will make it utilize the `pdist` function. In addition to a specifying a periodic box, other running parameters can be added then as well, such as the `what` parameter, which allows you to use Euclidean or Manhattan distances.

A minor pitfall in the calculator functions is the radius of gyration. Most of the calculator functions return the same value as the corresponding GROMACS alternatives, within a reasonable rounding error. However, the radius of gyration is calculated in a different manner: GROMACS uses the atomic mass of each atom to weigh their contribution to the radius of gyration, whereas TIPSI does not. As such, this parameter can only be used if TIPSI is used to define the states. This is possible, however, as running TIPSI without state definitions will simply make it track the values of the order parameters throughout a trajectory.

Miscelleaneous

While working on this project, it was found that TIPSI runs the backward shooting moves by using a negative timestep. This was not widely known in the group, some thought that TIPSI worked by multiplying the velocities with -1.

Summary

To summarize, a tutorial on TIPSI to be used by students and researchers in the group that want to get hands-on experience with the package was written. This tutorial includes minimal documentation, that ensures that TIPSI can be used without forcing the user to read the sourcecode. It also includes cheatsheets for students that are less familiar with Linux and GROMACS. The repository that contains all input files for the tutorial also has a Bash and Python script that assemble the output of TIPSI into a form that can be analyzed by researchers.

To conclude the project, overview of the work done was presented to the computational chemistry group on the 8th of July, 2016. In this half-hour presentation, both examples in the tutorial were discussed, results were shown, the documentation was outlined, and the Bash and Python scripts to process TIPSI output were explained.

References

[1] M.J. Abraham, D. van der Spoel, E. Lindahl, and B. Hess. Gromacs user manual version 4.5.4, 2010.

[2] Tsjerk A. Wassenaar. Transition interface/path sampling identifier, 2014.