Aimless Transition Ensemble Sampling and Analysis (**ATESA**)

Written and published by Tucker Burgin, © 2018
tburgin@umich.edu

Introduction

This document describes the usage of the ATESA python program to automate transition path sampling with the aimless shooting sampling method, using Amber installed on a cluster with a batch system (either PBS or Slurm). The code is designed to function flexibly under varied conditions and to communicate helpfully with the user at all times; however, as with most academic code, it was not created by a professional software developer, and is likely to contain undiscovered and undocumented bugs. Please feel free to contact the developer at the email address above or to raise issues on the project's GitHub page.

As of October 2018, ATESA is publicly visible; however development is NOT complete for either this software itself or for this document. See the section titled "Development Notes" below for details on the current state of ATESA and future plans.

A thorough explanation of aimless shooting (AS) as a technique and its relationship to other methods is beyond the scope of this document, and it is assumed that the reader is acquainted with the relevant theory. If this is not the case, an excellent primer is available from Beckham and Peters 2010. With that said, AS is a method to perform efficient, unbiased sampling of the region of phase space corresponding to a putative transition state. Because by definition the transition state is a local maximum in energy along at least one dimension, this region is difficult to sample using conventional simulations. The AS approach is to leverage one or more "guess" transition state structures (which are obtained by other methods as a prerequisite to beginning AS with ATESA) as a seed, which will be "aimlessly" "shot" through phase space using unbiased initial velocities chosen from the Boltzmann distribution. The resulting trajectory is simultaneously also attempted in reverse (using initial velocities of opposite direction and equal magnitude), and if after simulations the two trajectories converge to different energetic basins (one products, one reactants), then the reactive trajectory connecting them is considered a success. New starting points are daisy-chained from older successful ones by taking an early frame from the reactive trajectory as the initial coordinates, and in this way it is ensured that sampling remains nearby the transition state separatrix.

ATESA automates this process with a system of independent “threads” representing one particular path in the search through phase space. A thread has a given set of initial coordinates, which it repeatedly “shoots” until it finds a successful reactive trajectory, at which point it picks a new shooting point on the reactive trajectory and continues. Because threads run entirely in independently, ATESA scales perfectly so long as sufficient computational resources are available.

Table of Contents

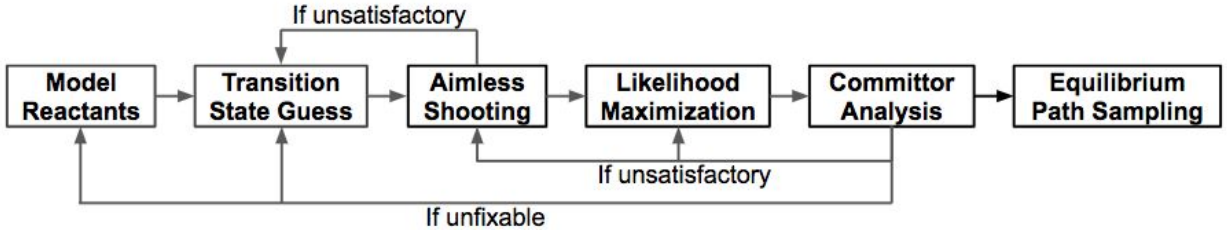
Development Notes

Last Updated October 8th, 2018. Currently, ATESA is functionally complete, in that there are no core features currently intended but not implemented. However, the code has not been thoroughly bug-tested, nor have unit tests with a high degree of coverage been implemented. Finally, integration with installation packages or update pipelines has not been finalized. Collectively, these three goals represent the development priorities for ATESA in the immediate future.

Because ATESA remains so rough around the edges, and because it has not been published in the scientific literature yet, interested users are encouraged to keep an eye on the project until it is more complete, which is expected within the next year. If you are eager to get started, please contact me at the email address given in the header of this document and I would be delighted to work with you on getting ATESA configured and working for you, and to discuss the appropriate citation, if applicable.

Scope of ATESA

ATESA automates a particular Transition Path Sampling (TPS) workflow that uses the flexible length aimless shooting algorithm of Mullen *et al.* as its main workhorse. TPS is an approach to obtaining and analyzing a reaction coordinate that describes a given chemical transformation on a computationally tractable timescale.



The aimless shooting TPS workflow used by ATESA. Of these steps, the first two (Model Reactants and Transition State Guess) are not (at least yet) part of ATESA and must be handled by the user separately.

ATESA requires first that a simulation-ready model of the reactant state for the desired reaction be obtained by any means. Then, one or more transition state guesses are prepared by stretching the relevant bonds (either manually by directly modifying coordinates, or using biased simulations) to distances between their expected product and reactant states (note that at this stage, the researcher will have injected an assumption about the reaction mechanism into the method. Successful TPS finds *a* reaction coordinate, but it does not guarantee that it is *the* "real" reaction coordinate as it might hypothetically be observable in nature with a really, really good microscope.)

Once the TS guess(es) is/are available, ATESA can be used to automate arbitrarily parallelizable aimless shooting sampling of the transition state ensemble, inertial likelihood maximization to obtain a suitable reaction coordinate, committor analysis to validate that reaction coordinate, and equilibrium path sampling to obtain the energy surface along that reaction coordinate.

Installation

<Insert install instructions here>

Care has been taken to ensure that atesa.py is compatible with Python versions 2 and 3; however, considerably more testing has been performed in Python 3, and as such this is the recommended version. Some older pytraj installations may not support Python 3, and will throw

an ImportError when Python 3 attempts to read them. If this happens, reverting to Python 2 *should* be an acceptable alternative, rather than requiring a more recent version of pytraj.

Quickstart Instructions

Although you are encouraged to read through the Usage section in more detail to get a grasp of what ATESA is capable of, many users will want to make a quick attempt at using the software to gauge its usefulness to them. If this sounds like you, <then what?>

Setting Up the Working Environment

Setting up your working environment to properly supply ATESA with all the necessary files and directories is essential, but is also designed to be as painless as possible. There are four directories that must be considered:

1. The “home” directory, which must contain:
    a. The “templates” directory, and
    b. The “input_files” directory
2. The directory from which ATESA is called

Of these, only one is defined in the input file (see `home_directory` below). The home directory does not need to contain anything other than the two subdirectories named above. By default, home is set to the directory that actually contains the atesa.py script, and its subdirectories are prepackaged there as well for the user to edit as necessary.

The “templates” directory must be named exactly that, and should contain at minimum just one file, named “batch_[batch_system].tpl”, where [batch_system] is either “slurm” or “pbs” (*sans* quotes) in accordance with the batch system type being used. This file is a Jinja2 template file that is used as a template for the code to fill when constructing batch jobs, and the example file included in the installation package can be edited as the user sees fit. Templated spots are denoted by double curly braces (*e.g.*, `{{ example }}`) and spots that the user does not wish to include for whatever reason can be safely deleted without error (or without error from ATESA, anyway — the batch system may complain if essential arguments are not supplied!) At minimum, you will likely need to modify the module dependencies and exact manner of calling sander in accordance with the setup of the batch system you are using. If the groupfile option is included in the input file, then another template appropriate for submitting groupfile jobs and called “batch_[batch_system]_groupfile.tpl” is required instead.

The “input_files” directory must also be named exactly that, and contains at minimum two files for a standard ATESA run: “init.in” and “prod.in”. These are the Amber input files for initialization and production runs, respectively, which refer (again, respectively) to the extremely short job

each shooting move makes starting from the shooting point to select its initial velocities, and the longer jobs in the forward and reverse directions from the initialization run during which commitment to basins A or B is awaited. Since the “prod” runs choose their initial velocities from their input coordinate files, the settings `ntx=5` and `irest=1` are mandatory. If they are omitted or changed, the program may appear to run without issue, but the resulting trajectories will not accurately represent aimless shooting moves. Another input file, “committor_analysis.in”, is required if the `committor_analysis` option is supplied in the ATESA input file. This file should specify simulations that do not read in velocities from their input coordinates (`ntx=1` and `tempi=` whatever `temp0` is set to).

The Amber input files may be left mostly unchanged, but at minimum the user will have to define the appropriate qm_region (and accompanying &qmmm namelist variables) for their system. Advice on choosing the appropriate region, and documentation on Amber input files more broadly, is outside the scope of this document; the user is referred to the Amber documentation for whatever version of the software they are using.

Finally, the directory from which ATESA is called is used as the relative location from which pointers to other locations in the ATESA input file are interpreted. If you use exclusively absolute paths, then this is of no concern.

Once the directories have been set up, the following files are required:

- The input file identified with the `-i` flag in the command line call (see the Usage section for details)
- The topology file for the model, identified with the `topology` option in the input file
- One or more coordinate files to begin aimless shooting from, identified with the `initial_structure` option in the input file

Warning: there are two rules about the naming of the initial structure files. First, the file names cannot contain space (‘ ‘) characters, as these would break certain functions of the software. If this rule is violated, the code will exit with an helpful error message. However, the code will not automatically exit if the other rule is violated: no file name can be equivalent to another with an underscore followed by an integer appended to it. For instance, the following pair of initial coordinate filenames will cause catastrophic failure without a helpful error message: “coordinates.rst7” and “coordinates.rst7_1”. However, the following pair is acceptable: “coordinates.rst7” and “coordinates_1.rst7”.

Usage

After installation, ATESA is called from the command line like so:

```
atesa [-O] [-i as.in] [-w working_directory]
```

The `-O` flag indicates whether or not the working directory should be overwritten if it already exists. If `-O` is not supplied and the working directory does exist, the program exits without doing anything. This is included as a safety measure to help avoid accidental deletion of existing data. The working directory defaults to `` `pwd`/as_working `` and can also optionally be set using the `working_directory` option in the input file (see below; in the event that both are specified, the input file entry takes precedence). This option is included in the command line to support working directories whose literal names may not be known in advance, such as scratch directories created by a batch system dynamically.

The input file indicated with the `-i` flag is required (although if it is not specified in the command line it will take on the default name `as.in`), and all the settings for the AS run are specified within as described in the rest of this section.

The input file should consist only of lines with the following format (*sans* brackets):

```
[variable_name] = [variable_value]
```

The two spaces flanking the equality sign (=) are mandatory, and no comments or other characters are permitted on the line. Unless otherwise specified, all variable values should be given without quotes, even for strings or names of files, *e.g.*,

```
topology = ts_guess.prmtop
```

The variable names accepted, their possible values, and their defaults are detailed below. If a variable is not specified in the input file, it will take on its default value, but take note that for a normal AS run **`commit_fwd`**, **`commit_bwd`**, and **`candidate_op`** are non-optional and do not have default values. Although most of the default values are likely to be acceptable in most cases, the user is strongly advised to at least skim this list in full before attempting any simulations, to avoid wasted time and/or resources.

`home_directory`

A string identifying the "home" directory of the ATESA run. This is the directory within which the folders named "templates" and "input_files" are stored. Default = the folder that atesa.py is located in

`initial_structure`

The filename corresponding to the initial coordinate file or files. The program will interpret the indicated path(s) relative to the directory from which it is called (not relative to **`home_directory`**). Pattern recognition is supported, iff **`if_glob`** = True. Default = inpcrd

if_glob

Boolean indicating whether or not to interpret **initial_structure** as a glob argument for pattern recognition. Accepted values are 'True' or 'False' (case-insensitive, without quotes). If using this option, matching files are only searched for in the directory from which atesa.py is called (todo: fix this?) Default = False

topology

Filename corresponding to the topology file for the coordinate files. Only a single topology file is supported; if aimless shooting is desired on multiple structures with different topologies, each topology must have its own instance of ATESA. The program will interpret the indicated path relative to the directory from which it is called (not relative to **home_directory**). Default = prmtop

n_adjust

Maximum number of frames from which the initial coordinates of the next shooting point are allowed to deviate from the initial coordinates of the last successful shooting move. The actual number of frames is chosen randomly in the range [1,**n_adjust**]. Default = 50

batch_system

Indicates the type of batch system that is being employed. Accepted values are 'PBS' and 'Slurm' (case-insensitive, without quotes). Warning: if the wrong value is chosen here the program will potentially fail without returning a helpful error message! Default = slurm

working_directory

The directory name to perform all the simulations in and output the log, output, and status files to. This should be a folder name that does not exist yet, which folder will be created in the given location; if it does exist, and the program was called with the -O flag, then it will be overwritten. This option supersedes the working directory given with the -w flag in the command line call, if applicable. Note that for analysis runs using the **resample**, **rc_eval**, or **committor_analysis** options, **working_directory** should point to an existing directory containing the results of a previous ATESA run (and in this case it will not be overwritten). Default = as_working

commit_fwd

A list-formatted object defining commitment to the "fwd", or products basin. The format for this list is as follows:

```
[[mask1A,mask2A...],[mask1B,mask2B...],[dist1,dist2...],['<lt/gt>','<lt/gt>'...]]
```

Each 'mask' should be an Ambermask-formatted string (including quotes; see Amber documentation for mask formatting details) matching the desired atom or

group of atoms (if more than one atom is matched, the center of mass of the selection is used). Each 'dist' value should be a number giving the cutoff distance in Å between the corresponding masks, and finally 'lt' or 'gt' for each column indicates whether the cutoff is passed when the distance is less than or greater than the given value. An example of a correctly formatted **commit_fwd** line is given below:

```
commit_fwd = [['@1','@2'],['@3','@4'],[1.50,2.75],['lt','gt']]
```

This commitment criterion is met when the distance between atom 1 and atom 3 is less than 1.5 Å, and the distance between atom 2 and atom 4 is greater than 2.75 Å. If chosen correctly, this variable should correspond to a state that can confidently be said to have commited to proceeding to products, rather than to reactants. For now, only distance-based commitment definitions are supported, but angles, dihedrals, and/or literal pytraj input may be supported in the future (especially if a user specifically requests it, so don't be shy!) There is no default for this variable; it must be supplied manually (can be omitted if **resample** = True, or if **rc_definition** is provided).

commit_bwd

As **commit_fwd**, but this time corresponding to commitment in the "bwd" or reactants direction. There is no default for this variable; it must be supplied manually (can be omitted if **resample** = True, or if **rc_definition** is provided).

candidate_op

A list-formatted object corresponding to the candidate order parameters (OPs) that the reaction coordinate may be constructed from during LMAX. If an OP is not included here, it cannot be taken into account when determining the reaction coordinate! Each of the corresponding distances, angles, and/or dihedrals indicated by this variable will be measured at the end of each shooting move and, along with the corresponding basin of one of the trajectories, will be passed into the output file in the proper format for the LMAX code produced by Baron Peters. The format of this variable is as follows:

```
[[mask1A,mask2A,...],[mask1B,mask2B,...],[mask1C,mask2C,...],[mask1D,mask2D,...]]
```

Each column of this variable corresponds to one OP, with each mask representing one atom (or the center of mass of many atoms) to include in the measurement. If a mask is supplied for each row of a given column (*i.e.*, for masks A through D for the same numerical column), the corresponding dihedral angle is measured; if only three (A through C) are given, it's an angle; and if only two (A and B) are given it's a distance. To omit a mask for any given position, simply use an empty string, ''. An example of a correctly formatted **candidate_op** line is given below:

```
candidate_op = [['@1','@2','@3'],['@4','@5','@6'],['@7','@8',''],['@9','','']]
```

This is interpreted as the dihedral defined by atoms 1,4,7,9; the angle between atoms 2,5,8; and the distance between atoms 3,6. If a row is empty (contains only empty strings) it may be omitted without error. There is no default for this variable; it must be supplied manually.

Alternatively to the explicit definition described above, **candidate_op** can be defined implicitly by giving a mask and a coordinate file as such:

```
candidate_op = [':10<@3.5','ts_guess.rst7']
```

This example will automatically select every second- through fourth-order OP (bonds, angles, and dihedrals) that can be constructed from any of the atoms found within 3.5 angstroms of residue 10 in the coordinate file ts_guess.rst7 (which is interpreted using the topology file given in the **topology** option). Explicit masks are also accepted (that is, they do not have to be formatted to match all atoms within a distance). The user should be aware that for implicit definitions based on even quite modest distances, tens of thousands or even millions of candidate order parameters may be produced, which can significantly slow down resampling and outputting of data. This option provides a means of sampling the full configuration space around the transition state for likelihood maximization without requiring the user to exhaustively curate the definitions of those OPs by other means.

restart_on_crash

A boolean indicating whether or not threads that crash (do not produce a restart file) during the initialization step (that is, during the first simulation step for each shooting move, wherein the initial velocities are chosen) should be allowed to remain dead (False) or resubmitted to the queue (True). Failure of this type typically indicates either errors in the jobs themselves (*e.g.*, a typo in the Sander input file) or insufficient resources (memory or walltime) (in which case you would want to set this option to False), or poor convergence in the QM structure (in which case you may want to set this to True, to automatically reroll the initial velocities in hopes of obtaining a more convergent simulation). Note that this does not influence the behavior of the program when a simulation runs successfully but does not converge to the product or reactant basins. Default = False

max_fails

An integer corresponding to the number of times a thread is allowed to "fail" (that is, the simulations proceed but either do not converge to a product or reactant state (as defined by **commit_fwd** and **commit_bwd**) within the walltime of the corresponding batch jobs, or both converge to the same basin) in a row. If this number of failed runs is met before a successful run, the corresponding thread will be terminated without impacting the others. This is intended to provide a way to eliminate shooting points that have strayed too

far from the separatrix to continue to succeed. This variable can be set to a negative value to remove the limit and continue submitting jobs until the **max_moves** or **max_accept** limit is met (whichever comes first). Default = 10

max_moves

An integer corresponding to the total number of shooting moves that each thread is allowed to attempt before being terminated, regardless of the outcome of any given move. This effectively puts a cap on the total computational expense that ATESA might incur. This variable can be set to a negative value to remove the limit and continue submitting jobs until the **max_fails** or **max_accept** limit is met (whichever comes first). Default = 100

max_accept

An integer corresponding to the total number of accepted shooting moves that each thread is allowed to make before being terminated. In effect, this sets an upper limit on the number of unique shooting points that a given thread will explore (assuming that **always_new** is set to "False"; see below), and if this is the only active limit, then AS terminates when each thread has explored exactly that number of points. This variable can be set to a negative value to remove the limit and continue submitting jobs until the **max_fails** or **max_moves** limit is met (whichever comes first). Default = 100

degeneracy

An integer corresponding to the number of duplicate (or "degenerate") threads that should be spawned for each of the given initial structures. At a value of 1, only a single thread is produced per structure. This option provides a way to scale up the rate of sampling arbitrarily for a single given structure, without needing to call multiple instances of ATESA. Default = 1

init_nodes

Number of nodes to dedicate to each job during the initialization step. This value fills the {{ nodes }} template spot in the initialization batch file. Default = 1

init_ppn

Number of processors per node to dedicate to each job during the initialization step. This value fills the {{ ppn }} template spot in the initialization batch file. Default = 1

init_walltime

Walltime to permit each initialization step. Should consist of colon-separated integers of format HH:MM:SS (*e.g.*, 01:30:00). This value fills the {{ walltime }} template spot in the initialization batch file. Default = 01:00:00

init_mem

Amount of memory per processor to assign each job during the initialization step. This value fills the {{ mem }} template spot in the initialization batch file, and can be safely ignored if you have removed that spot from your batch template. Default = 4000mb

prod_nodes

Number of nodes to dedicate to each job during the production step. This value fills the {{ nodes }} template spot in the production batch file. Default = 1

prod_ppn

Number of processors per node to dedicate to each job during the production step. This value fills the {{ ppn }} template spot in the production batch file. Default = 1

prod_walltime

Walltime to permit each production step. Should consist of colon-separated integers of format HH:MM:SS (*e.g.*, 01:30:00). This value fills the {{ walltime }} template spot in the production batch file. Because atesa.py implements the flexible length aimless shooting algorithm, each job may not use up the full walltime it is allocated, even if it is unable to complete the number of steps indicated in the Amber input file in that time. Instead, production simulations are dynamically cancelled once they meet the either of the commitment criteria given in the **commit_fwd** and **commit_bwd** options. Default = 01:00:00

prod_mem

Amount of memory per processor to assign each job during the production step. This value fills the {{ mem }} template spot in the production batch file, and can be safely ignored if you have removed that spot from your batch template. Default = 4000mb

fork

Integer indicating the number of threads to spawn from each successful shooting move. If **fork** = 1, each thread simply continues upon success (subject to the termination criteria). For larger values, in addition to the continued thread, each successful move spawns **fork** - 1 new threads (named by appending a new numbered suffix to the name of the parent thread). Threads spawned in this manner do not inherit the histories of their parents; that is, they are treated as new for the purposes of termination criteria. For this reason, if threads would otherwise only terminate based on **max_accept** or **max_moves** (because **max_fails** is turned off or is never reached), then for values of **fork** greater than 1, the only termination criterion is effectively the amount of time for which ATESA is allowed to run. Default = 1

always_new

Boolean dictating whether a new shooting point is obtained from the last successful reactive trajectory only after successful moves (False), or after *every* shooting move regardless of outcome (True). Setting this option to "True" results in the exploration of a wider ensemble of candidate transition states and may improve acceptance ratios (or rather, prevent them from getting worse over time), but it has not been rigorously shown that this is preferable for obtaining a "better" reaction coordinate. Default = False

groupfile

This option indicates that jobs should be submitted together in a groupfile rather than as individual batch jobs. It should be given as an integer equal to the number of simulations per job (where 0 means no groupfiles are used, and 1 means groupfiles with one simulation each are used). This option is for users of computing clusters that don't support shared computation (*i.e.*, multiple different batch jobs sharing one node) and thus must allocate all the cores on each node for each job. Groupfile jobs are submitted to the batch system using the batch job options whose names begin with "prod" (as opposed to "init"). Because the constituent simulations of groupfile jobs cannot be terminated without cancelling the entire job, this option disables flexible length aimless shooting. For this reason, users are encouraged to choose their **prod_walltime** option with extra caution to avoid cutting off potentially convergent jobs early, or else wasting computational resources simulating already-converged structures. The user is encouraged to run a few simulations manually to estimate the walltime required for the system to meet the given definition of commitment, and then add 50% onto that to be safe. Be aware that the current build of atesa.py will sometimes submit groupfiles with **groupfile**−1 simulations when **groupfile** is an odd number; this may change in future versions. Warning: Currently, committor analysis is not supported with **groupfile** > 0. Default = 0

groupfile_max_delay

A time in seconds that groupfiles are permitted to await being "filled" before they will be submitted regardless of size. This option is disabled when set to 0, in which case groupfiles will wait indefinitely for the number of simulations indicated in **groupfile** to populate them before being submitted to the batch system. This option provides a means of compromising on computational efficiency in exchange for decreasing the time that simulations have to wait for others to complete before they can begin. For example, consider the situation in which **groupfile** = 5, and there are nine jobs in the queue (numbered 1-9, respectively):

groupfile_1: contains 1, 2, 3, 4, 5, submitted right away
groupfile_2: contains 6, 7, 8, 9, _, awaiting completion of groupfile_1

In this case, four of the nine threads will be waiting at any given time, unless a **groupfile_max_delay** setting is included. This option also provides a

failsafe in the event that fewer simulations are queued than the groupfile is waiting for (in which case if this option is set to 0, the program will continue indefinitely without doing anything). If **groupfile** = 0, this option does nothing. Default = 3600

The following commands have to do with performing analysis on existing ATESA data, rather than generating new data. In each case, **working_directory** should point to an existing directory containing ATESA data from a previous run. Warning: you are advised to ensure that your calls to ATESA do not include the -o flag when attempting analysis runs, to avoid accidentally deleting data instead of analyzing it!

resample

Boolean to indicate that this run should not perform any new simulations, instead building a new "as.out" file from existing files using a new value of **candidate_op**. This option allows the user to modify the order parameters that are considered during LMAX, without the need to rerun the simulations. Resampling is performed inside the working directory identified in the input file, and requires the presence of the log file produced in the course of the simulations in that directory. When this option is set to "True", the otherwise-mandatory options **commit_fwd** and **commit_bwd** can be omitted from the input file. Default = False

rc_definition

If this option is supplied, then this run will not perform any new simulations, and will instead create a new file (rc_eval.out) that provides a sorted list of the values of the reaction coordinate (based on the definition given here, as described below) at each shooting point. This information is of value in performing committor analysis and equilibrium path sampling. Computation is performed inside the working directory identified in the input file, and requires the presence of the log file produced in the course of the simulations in that directory. The reaction coordinate equation is supplied as a python-interpretable line of code, where each of the order parameters defined in **candidate_op** can be used as a variable, named OP# (where # is the index of the column in **candidate_op** that defines the order parameter (starting from 1)). Unlike in other options, space characters (' ') are tolerated. An example of a correctly formatted **rc_definition** is:

```
rc_definition = 2.44*OP3 + 1.21*OP1 - 3.56
```

where OP3 and OP1 refer to the coordinates given by the third and first columns of **candidate_op**, respectively. Functions of the OPs can also be used in place of the OPs themselves using the standard python mathematical operators (+, -, *, /, **, %) and any mathematical functions interpretable by the numpy library (*e.g.*, numpy.sin(), numpy.log(), *etc.*), such as in this example:

```
rc_definition = 3.53*(OP3+OP2) - 0.99*(4*numpy.sin(OP1)**2) + 2.17
```

This string is interpreted using the python function eval(), which means that you can feed ATESA arbitrary code to execute; for this reason among others, 'os.system('rm -rf /')' makes a poor reaction coordinate. Note that if you supply this option, **rc_minmax** must also be supplied (unless you want to use raw values, see below). There is no default value for this option.

rc_minmax

A list object that defines the minimum and maximum observed values for the OPs defined in **candidate_op**. This allows them to be converted to reduced variables, as is standard when using an RC definition obtained from likelihood maximization. This is formatted as a two-row nested list of minimum and then maximum values whose row corresponds to the OPs in **candidate_op**, *e.g.*:

```
rc_minmax = [[3.3,'',1.41],[5.2,'',3.9]]
```

In this example, OP1 is stated to have a range of 3.3 to 5.2, and OP3 is between 1.41 and 3.9. No values are given for OP2, which is acceptable iff **rc_definition** does not contain OP2. Furthermore, if there are more than three OPs, those whose indices are larger than the largest included in **rc_definition** may be omitted (so *e.g.* maybe in the above example OP4 is also defined, but is omitted in **rc_minmax** because it does not appear in **rc_definition**). If you want to use the raw values of the OPs instead, give **rc_minmax** = '' or else omit it entirely. Default = ''

include_qdot

committor_analysis

List of variables to define committor analysis. If this option is included, no new aimless shooting simulations are performed and instead the program performs committor analysis to analyze whether the putative reaction coordinate defined by **rc_definition**, **rc_minmax**, and **candidate_op** is suitable (thus, these options must be supplied alongside this one). The given **working_directory** must contain existing ATESA data, along with the corresponding as.log file. As with aimless shooting, a detailed description of committor analysis is outside the scope of this document, so be sure you understand what you're doing before attempting to use this option. The correct format of this option is as follows (note that as usual, no spaces (' ') are allowed):

```
committor_analysis = [n_shots,rc_ts,rc_tol,min_points,min_dist]
```

Of these values, rc_ts and rc_tol are floats or integers, and the rest are strictly integers. The output of a committor analysis run is a file named

committor_analysis.out, containing the data for a histogram of p_B values for n_shots simulations starting from each of the shooting points in the given **working_directory** with a reaction coordinate value within rc_tol of rc_ts. If any two shooting points are from the same thread and have at least min_dist shooting moves separating one another, only the earlier of the two is included (this option is intended to help ensure decorrelation between analyzed shooting points; it can be set to 1 to ignore it). If there are fewer than min_points eligible shooting points, the difference is made up by equilibrium path sampling-style confined sampling using the options in **eps_settings**, but on a single window centered on rc_ts (regardless of the n_windows setting in **eps_settings**). The simulations performed in the course of committor analysis use the options whose names begin with "prod". The working directory for the simulations will be a new subfolder within the as_working directory called committor_analysis, and that is also where the results are outputted to. Default = []

eps_settings

List of variables to define equilibrium path sampling. If this option is included, no new aimless shooting simulations are performed and instead the program performs equilibrium path sampling to measure the free energy profile (a.k.a. the potential of mean force, PMF) along the reaction coordinate defined by **rc_definition** and **candidate_op**. As with aimless shooting, a detailed description of equilibrium path sampling is outside the scope of this document, so be sure you understand what you're doing before attempting to use this option. At least one initial coordinate file from each of the RC windows should be defined in **initial_structure** in order to produce a complete PMF, but if any are missing the program will continue with a warning. The correct format of this option is as follows (note that as usual, no spaces (' ') are allowed):

```
eps_settings = [n_windows,k_beads,rc_min,rc_max,traj_length,overlap]
```

n_windows is an integer number of evenly-spaced EPS windows into which to divide the space along the reaction coordinate between rc_min and rc_max (both floats or integers). Similarly, k_beads is an integer number of evenly-spaced beads or "nodes" into which to divide each trajectory of traj_length steps. Finally, overlap is an integer or float indicating the additional width along the RC to append to either side of each RC window. This helps ensure overlap between windows when backing out the free energy profile from the histogram data.

The input file for each EPS production runs is templated from "eps_in.tpl" (located in the templates directory; "input_files/init.in" is still used for EPS initialization runs), in which the only templated slots are {{ nstlim }} and {{ ntwx }}. Note that traj_length must be divisible by k_beads. Just as in aimless shooting, EPS runs consist of both initialization and production runs (and their corresponding input file options.) Indeed, with the exception of

n_adjust, the fwd and bwd commitment definitions, **resample**, and **always_new**, all of the above options apply to equilibrium path sampling in the same way they applied to aimless shooting (including the termination criteria!) If either of the boolean options mentioned above are included in the input file along with **eps_settings**, the program will exit with an informative error message, while the three non-boolean options are simply ignored. Also note that while **eps_settings** and **committor_analysis** are not incompatible, if both are supplied only a committor analysis run will be performed (using settings from **eps_settings** as necessary, as described in the documentation for that option). Warning: it is advisable to set EPS to run in a different working directory than any previous ATESA runs, to avoid accidental overwriting of files or misinterpretation of pre-existing data as new data by the EPS code. Default = []

eps_dynamic_seed

An integer or list of integers of length "n_windows" from **eps_settings**. Including this option indicates that empty EPS windows should be dynamically seeded using the results of other EPS shooting moves. When this option is omitted, windows that do not contain any points initially (*i.e.*, those into which none of the coordinate files matched by **initial_structure** fall) will throw a warning and then remain empty throughout the EPS run. When this option is instead set to an integer, new threads are spawned inside of empty windows when those windows are visited by trajectories centered in other windows, up to that integer value of new threads. Finally, if a list is given, then the value corresponding to each window (starting from the left with smaller RC values) indicates the number of threads to spawn in that window. With this option, the full range of EPS windows defined in **eps_settings** can be sampled from even if the supplied initial structures do not span the full range of those windows (so long as at least one initial structure is given that does, in fact, occupy a window to which a non-zero number of threads have been assigned with this option). Note that **degeneracy** still duplicates the given initial structures as usual when this option is given, and that each of the resulting threads counts towards the total for the given window. If more initial structures than the given **eps_dynamic_seed** value for that window are supplied manually, they will still all be used. Note also that in practice it may take extremely long for some windows to be entered in this way depending on where the structures that are supplied fall; for this reason, it is advisable to begin with structures known to have a reaction coordinate value near the transition state(s). Default = ''

Troubleshooting

Although every effort has been made to provide helpful error messages to correspond to every foreseeable issue, the fact remains that not every issue is, in fact, foreseeable. This section is

designed to help the user troubleshoot should they run into issues not accompanied by a custom error message.

Issues running ATESA will fall into one of three categories, based on where in the pipeline the issue arises:

1. The local shell or the batch system
2. Amber
3. ATESA itself

Here, I will attempt to describe how to identify the category of error being encountered, and provide suggestions as to where to look for the culprit.

The local shell or the batch system

Errors encountered by these systems will appear in one of two places. If the shell encounters an error, it will be reported directly in the command line after ATESA is called. This likely indicates an issue with the installation of the software, or of Python itself (ATESA is written in Python 3 and has not been extensively tested in Python 2, so check your version). Command line errors will also arise if the batch system does not recognize the formatting of the batch files submitted by ATESA; double-check that you have set the input variable batch_system correctly, and that your batch templates are of the correct format. Also, note that if ATESA is called from inside a batch job (as is recommended when performing many simulations), the command line output will be piped to the batch output file for that job.

Errors encountered by the batch system will also appear in batch output files, but in this case they will appear in the individual batch job output files in the working directory. These errors are likely to do with the formatting of the batch template files used by ATESA to create its job batch files. For example, if the user has removed the line in the batch template indicating the nodes to be assigned to the job, ATESA will not complain (as omitting variables in the templates is supported in general), but the batch system will (because this particular variable is mandatory). Other batch system errors include those not specific to ATESA, such as insufficient funds on the user account; to check whether a given error is of this type, try submitting a job that does not involve ATESA.

Amber

Amber has its own output file for each job, usually given by the suffix ".out" or ".mdout" in the working directory (note: ATESA has an output file named with the ".out" suffix in this directory, as well.) If an error arises here, it likely indicates an issue with the formatting of the input coordinate or topology files, or else with the input file passed to Amber. Another common source

of error is the absence of the necessary quantum mechanics files (located in `$(AMBERHOME)/dat/slko/`; if these are missing they may need to be installed for you by a user with root access) for the chosen level of theory, which is DFTB in the default input file. ATESA has been successfully tested on Amber 14, 16, and 18, although I did encounter an unusual error in the ability of Amber 14 to perform some quantum mechanical simulations.

ATESA

Errors encountered by ATESA will be outputted into the command line in the event that the program has been called directly, or else into the batch output file in the event that it has been called in a batch job. If an error message appears beginning with "Error:" without a corresponding Python traceback, then it is a custom error and I hope that the accompanying message is clear (note that internal errors from pytraj (*e.g.,* "Error: NetCDF file has no frames.") are normal, and can usually be ignored). If a Python traceback appears, it indicates an error that I did not foresee. Conceivably, errors of this type could have to do with improperly installed dependencies, especially Jinja2 or pytraj. One possible troubleshooting option is to retry the job with the default settings to see if the error persists — if it does not, add on custom settings one-by-one (starting with those you are least confident in) until the culprit is found.

Although the user is of course permitted under the license to attempt to debug the code themselves (and to use the modified code in whatever manner they see fit), no one finds it easy to read someone else's code, especially if they are not especially fluent in Python (as I admit I am not). If you encounter an issue with ATESA that you cannot resolve, I encourage you to raise an issue on the corresponding GitHub page, or to send me an email directly. And if you encounter an error that you *did* resolve by modifying the code, I would love to hear about that, too!

Not happy with the results?

One final type of issue is the case where the software has functioned as intended, but has not provided results that satisfy you. Perhaps your acceptance ratios are very poor, or even zero, or perhaps LMAX was unable to confidently determine a reaction coordinate from the as.out file. In such cases, you are encouraged to look to the chemistry. A chronically poor acceptance ratio, even across many degenerate threads or slightly different input coordinate files, probably indicates that your initial coordinates are not as close to the reaction separatrix as you might have hoped. Even if they are very close, an extremely steep energetic landscape makes aimless shooting difficult, and depending on the context of your study may indicate an incorrect putative reaction mechanism. You should also consider making `n_adjust` smaller; although a larger value can explore phase space more efficiently, it also increases the risk of a thread straying too far from the separatrix and being unable to climb back up, resulting in poor acceptance ratios after that step. This effect can be mitigated by setting `always_new` to "`True`".

A high value of `fork` will offset this risk, as well, but be mindful of the ballooning computational cost.

If acceptance is good but LMAX is struggling, consider resampling your existing simulation files (using `resample = True`) with more order parameters included; it's possible that you left something out.