PyRETIS: A well-done, medium-sized Python library for rare events

Anders Lervik[*], Enrico Riccardi[†], Titus S. van Erp[‡]

June 30, 2017

Abstract

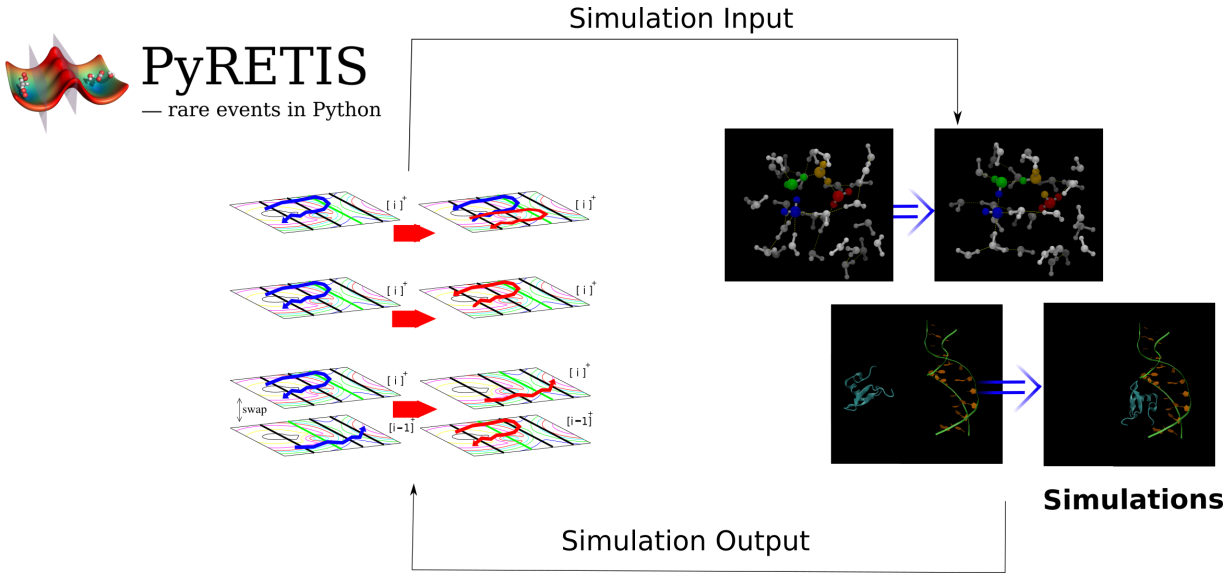
Transition Path Sampling techniques are becoming common approaches in the study of rare events at the molecular scale. More efficient methods, such as transition interface sampling (TIS) and replica exchange transition interface sampling (RETIS), allow the investigation of rare events, e.g. chemical reactions and structural/morphological transitions, in a reasonable computational time. Here, we present PyRETIS, a Python library for performing TIS and RETIS simulations. PyRETIS directs molecular dynamics (MD) simulations in order to sample rare events with unbiased dynamics. PyRETIS is designed to be easily interfaced with any molecular simulation package and in the present release, it has been interfaced with GROMACS and CP2K, for classical and ab initio MD simulations, respectively.

Keywords: rare event, path sampling, Python, transition, reaction

[*]Department of Chemistry, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway, anders.lervik@ntnu.no

[†]Department of Chemistry, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway

[‡]Department of Chemistry, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway



PyRETIS is a Python based code to sample rare events in path space. It can handle different types of dynamics and it can be easily interfaced with different software at different resolution levels. Proton transfer between water molecules and biomolecule-DNA binding are just two examples of the transitions that PyRETIS can describe quantitatively without the implementation of any bias.

1 Introduction

Molecular dynamics (MD) simulations have become common computational experiments in many disciplines. From biology to material science, descriptions at atomic scale of various material properties and processes have been obtained.[1–5] Validations of assumptions and parameter estimation for continuum modeling have also been performed in systems for which lab experiments can not be applied.[6–9] On the other hand, molecular simulations have severe limitations for the accessible time and length scale. Thus, there is still a significant need for more efficient algorithms in order to reach longer time scales and study larger systems without loosing valuable information.

Especially when simulations aim to capture transition events (e.g. chemical reactions, nucleation, protein folding), exceedingly long simulations are required to describe the mechanisms and quantify their rates. Several approaches have recently been developed to increase the simulation efficiency.[10–18] However, the majority of these methods achieve this by disturbing the physically correct dynamics of the system. Transition path sampling (TPS),[19] on the other hand, samples unbiased molecular dynamics trajectories via a Monte Carlo (MC) procedure. Among the methods using a TPS based approach, transition interface sampling (TIS)[17] has grown as one of the most popular ones. Following the TIS development, more advanced algorithms have been invented, e.g. replica exchange transition interface sampling (RETIS)[18] with new MC moves[20] that further increases the sampling efficiency.

The beauty of TIS based methods is that its results are equivalent to the ones that would be obtained by running exceedingly long brute-force molecular simulations, but orders of magnitude faster. The lack of biased dynamics also implies that the TIS based methods can easily be interfaced with any simulation approach (e.g. simulations with different resolutions, like coarse grained, full atomistic, ab initio MD, Langevin dynamics, dissipative particle dynamics, Brownian dynamics, kinetic MC, etc.) and any external MD code. In the present manuscript, we present our library, named PyRETIS, for performing TIS-based rare event simulations. The PyRETIS library handles the rare event algorithms and it employs external simulation packages for running MD simulations. Currently, PyRETIS is interfaced with two very popular simulation codes: CP2K[21] (for ab initio based MD) and GROMACS[22] (for classical MD) and the library is designed so that no modifications of the external simulation package is needed. This means that PyRETIS can, in principle, be interfaced with any free or closed source MD simulation code (presently, we are developing interfaces with LAMMPS,[23] VASP,[24] Espresso,[25,26] and QChem[27]).

While the TIS theory has been well established in the latest years,[17,18,20] its practical usage has been limited by a lack of coordinated computational efforts. To address this demand, we have developed a Python based library to perform rare event simulations. The programming language has been chosen to reach many users and potential developers. Interpreted, dynamic languages such as Python are known for poorer computational efficiency compared to compiled languages (e.g. FORTRAN or C). However, since the MD simulations are by far the most costly part of the path sampling algorithms, the relative poor performance of current Python interpreters is limited since PyRETIS make use of efficient external programs for the actual MD steps. Further, Python is a relatively simple programming language to learn, has a large user base[28] and several popular libraries for scientific computing is available (e.g. Numpy and SciPy[29,30]) which can reduce the programming efforts, especially for beginners.

Currently, there exist several Python packages for performing MD simulations (for instance MMTK[31] and PyOpenMM[32]) and for setting up and analyzing output from MD simulations (for instance, MDAnalysis[33] and MDTraj[34]). This allows straightforward modification of sampling algorithms, order parameters and implementation of additional analysis methods in PyRETIS.

We begin the article by a description of the RETIS algorithm, since this algorithm is the central rare event sampling technique implemented in PyRETIS, before we give an overview of the library with required input and given output. We also show an example of how PyRETIS can be extended by adding a custom order parameter and we report some practical examples of the use of PyRETIS. The purpose of these examples is to get acquainted to the use of the library and most of them can be performed on a simple laptop. Finally, we close the article by describing our plans for future development and how the library can be obtained.

2 Replica Exchange Transition Interface Sampling

Replica Exchange Transition Interface Sampling (RETIS) is a rare event method based on transition interface sampling. To explain its basic concepts, we consider the transition between two stable states, from the reactant (labeled A) to the product (labeled B). These two states are defined by a progress coordinate, λ, where state A is for $\lambda \leq \lambda_{\mathrm{A}}$ and state B for $\lambda \geq \lambda_{\mathrm{B}}$. Here, and in the following, the term progress coordinate is used to denote a particular order parameter which describe the progress of the reaction. The central quantity calculated in RETIS simulations is the rate constant k_{AB} for the transition from state A to state B which can be expressed as

$$k_{\mathrm{AB}} = f_{\mathrm{A}} \mathcal{P}_{\mathrm{A}}(\lambda_{\mathrm{B}}|\lambda_{\mathrm{A}}), \tag{1}$$

where f_{A} is the so-called initial flux (through λ_{A}) and $\mathcal{P}_{\mathrm{A}}(\lambda_{\mathrm{B}}|\lambda_{\mathrm{A}})$ the crossing probability. Formally, the crossing probability is the probability of crossing λ_{B} before λ_{A} given that λ_{A} has just been crossed.

In practice, the RETIS algorithm splits up the calculation of the crossing probability by considering so-called path ensembles. Within each path ensemble the algorithm generates trajectories which sample the progress coordinate space. Mathematically, a path ensemble comprises all possible trajectories that fulfill certain conditions: in RETIS, all path ensembles contain trajectories that start at the foot of reaction barrier from the reactant side (λ_{A}), end in the reactant (λ_{A}) or product region (λ_{B}) and having reached a certain threshold value (λ_i) at some point in the trajectory. The threshold value, λ_i, differs for each path ensemble and we label the corresponding path ensemble by $[i^+]$. Typically, $N+1$ path ensembles are defined by positioning $N+1$ interfaces along the progress coordinate space, e.g. $\{\lambda_{\mathrm{A}} = \lambda_0, \lambda_1, \lambda_2, \ldots, \lambda_N = \lambda_{\mathrm{B}}\}$. This sub-division of the progress coordinate space defines the path ensembles $[0^+]$, $[1^+]$, $\ldots$, $[(N-1)^+]$. In addition, a special path ensemble, $[0^-]$, is considered in order to calculate the initial flux. This path ensemble contains trajectories that explore the reactant state. The overall crossing probability can then be expressed as

$$k_{\mathrm{AB}} = f_{\mathrm{A}} \mathcal{P}_{\mathrm{A}}(\lambda_{\mathrm{B}}|\lambda_{\mathrm{A}}) = f_{\mathrm{A}} \prod_{i=0}^{N-1} \mathcal{P}_{\mathrm{A}}(\lambda_{i+1}|\lambda_i), \tag{2}$$

where the f_{A} is given by

$$f_{\mathrm{A}} = \left(\left\langle t_{\mathrm{path}}^{[0^+]} \right\rangle + \left\langle t_{\mathrm{path}}^{[0^-]} \right\rangle \right)^{-1}, \tag{3}$$

and $\langle t_{\mathrm{path}}^{[0^+]} \rangle$ and $\langle t_{\mathrm{path}}^{[0^-]} \rangle$ are the average lengths of the paths in the $[0^+]$ and $[0^-]$ ensembles, respectively. $\mathcal{P}_{\mathrm{A}}(\lambda_{i+1}|\lambda_i)$ is the probability of a path crossing λ_{i+1} given that it originated from λ_{A}, ended in λ_{A} or λ_{B}, and had at least one crossing with λ_i. This crossing probability is estimated for each path ensemble $[i^+]$ by sampling trajectories which are generated by a set of Monte Carlo (MC) moves. These MC moves generate new trajectories from already accepted trajectories in the different path ensembles. The relevant MC moves for the RETIS algorithm are depicted in Fig. 1. As shown in this figure, the RETIS method uses the

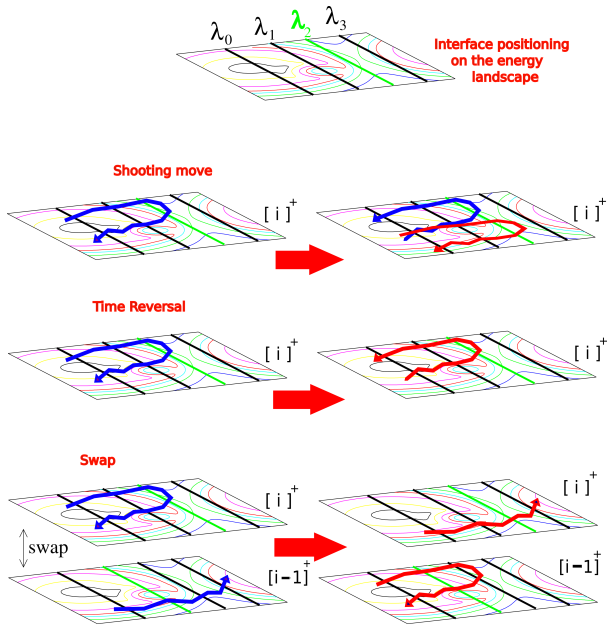


Figure 1: Scheme of the RETIS method for generating trajectories. A contour plot of a hypothetical free energy surface along a progress coordinate and an arbitrary second coordinate is shown, and 4 interfaces (λ_0, λ_1, λ_2, λ_3) have been positioned along the progress coordinate. Three different RETIS moves (shooting, time reversal and swapping) are illustrated for the $[i^+] = [2^+]$ path ensemble. The old paths are in blue and the new paths after (a successful) completion of the MC moves are shown in red.

TIS MC moves (shooting and time reversal) with, in addition, the swapping move, which significantly increases the sampling efficiency.[18]

The shooting move is adapted from the TPS shooting algorithm,[35,36] to allow variable trajectory lengths. It consists of a Monte Carlo (MC) algorithm in which one of the discrete MD steps of the present path is randomly selected. After modifying the velocities of this phase point (e.g. by randomly drawing new velocities from a Maxwellian distribution), a new trajectory is generated and accepted or rejected according to a detailed balance[20,37] condition. This condition ensures unbiased sampling. If a trajectory does not fulfill the specific ensemble threshold condition, it is rejected.

The shooting move gives a much higher chance to generate a valid trajectory at each trial compared to simply starting from a random phase point within the reactant well. Since trajectories are generated by the shooting move, computationally expensive MD simulation steps are required. In comparison, time reversal is an inexpensive move in path space: by simply changing the time direction of a path, it eventually increases the number of accepted paths of its ensemble.

The swapping move[18] (which is inspired by replica exchange MC moves) acts between different path simulations. If two simulations generate simultaneously two paths that are valid for each other's path ensemble, these two paths can be swapped as shown in Fig. 1. The swapping moves increase with negligible extra computational cost the number of accepted paths in the ensembles and decreases significantly the correlations between the consecutive paths within the same ensemble. There is one notable exception where the swapping move is more computationally demanding: the swap between the $[0^+]$ and $[0^-]$ ensembles requires MD simulation steps.[18] Currently, computational strategies for lowering the cost of this move is under development[38] and planned for future implementation in the PyRETIS library.

The efficiency of TIS, and moreover of RETIS, is relatively insensitive to the choice of progress coordinate. This is advantageous in complex condensed systems where it is difficult to determine, a priori, the most efficient progress coordinate(s).[39] PyRETIS supports both the TIS and RETIS algorithms. The TIS algorithm can trivially be run in parallel, while for the RETIS algorithm this is currently not the case. The RETIS swapping move is applied to the whole set of path ensembles when all path ensembles have completed a full MC step. Therefore, a straightforward parallelization of RETIS in which each path ensemble is for instance treated by a separate computer node, would lead to many computer nodes being idle, each time they complete a short path and are waiting for longer paths to be finished in other path ensembles. Novel algorithmic set-ups to avoid this issue are under development and it is a long-term goal for the PyRETIS project to implement a parallel RETIS algorithm. We note however, that the MD steps can be run in parallel for both TIS and RETIS.

3 Overview of the PyRETIS library

PyRETIS is designed as a Python library and it makes use of both object oriented and procedural programming principles. Simulations can be set up and carried out by explicitly making use of the library in a Python script, or by making use of an input file as described in the next section. Due to the dynamic nature of Python, the PyRETIS library can easily be extended by the end-user, and, by making use of the library, customized simulations can be created. We show an example on how PyRETIS can be extended by adding a custom order parameter in section 6.

In Fig. 2 we illustrate the main simulation loop in a typical rare event PyRETIS simulation. To start a rare event simulation, the user defines the path ensembles to consider by deciding the positions of the interfaces. Optimal placement of these interfaces and the number to consider has to be determined by the user and we give some guidelines for this in section 5.

Further, a valid initial trajectory for each path ensemble (i.e. a trajectory which fulfills the specific ensemble conditions) is required. In PyRETIS this initial path can be either

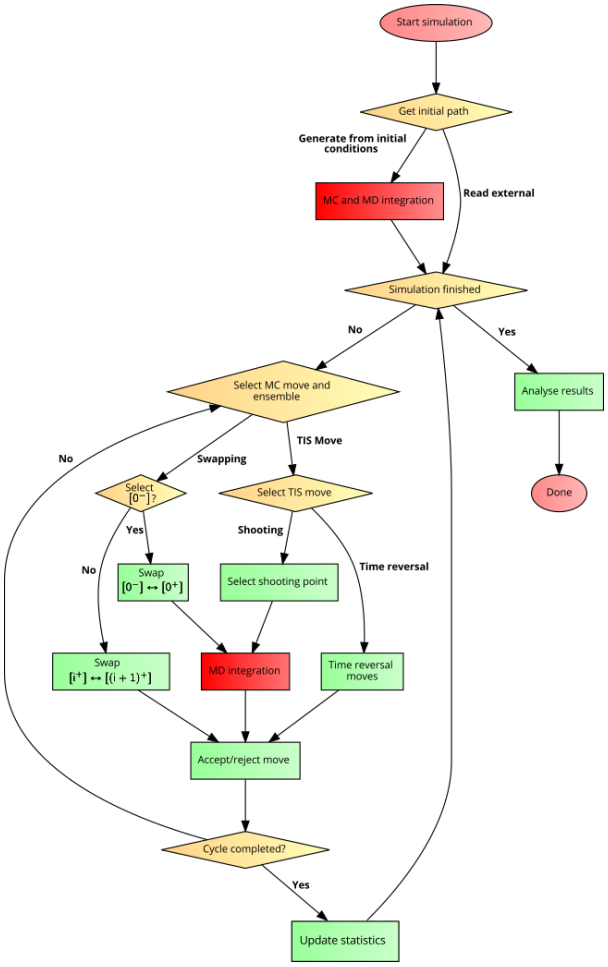


Figure 2: Flowchart of a rare event simulation with PyRETIS. Following generation of the initial paths, the main simulation loop is entered. Here, new trajectories are generated from old ones by randomly selecting a MC move as described in the main text. In the case shown, there are three options and two of these options may lead to an integration of the equations of motion. Both the time reversal and the swapping moves which do not involve $[0^-]$ are inexpensive operations that do not require MD simulation. However, the shooting move and the swapping move in which $[0^-]$ is used require MD simulations for obtaining a new trajectories and these are the most costly operations.

read from an already existing trajectory (for instance generated by an external MD software using meta-dynamics) or it can be generated by PyRETIS. The generation of the initial paths by PyRETIS is performed via a hybrid MC/MD method which search for a crossing with the required interface. This is done in the following way: An initial configuration supplied by the user is read and set as the current configuration. Then the velocities of the current configuration is modified by assigning randomized velocities from a Maxwellian distribution, and this is followed by one MD integration step. The current configuration is then updated if the resulting configuration after the MD step is closer to the interface. Otherwise, the old configuration is kept and the process (assigning velocities followed by one

MD integration step) is performed again. This is repeated until the current configuration and the configuration after the MD step are on different sides of the relevant interface, giving a crossing of the interface. A full trajectory is then generated by integrating the configuration left of the interface backwards in time and the configuration right of the interface forward in time.

After initiation, PyRETIS generates new trajectories according to the TIS/RETIS algorithms (as shown in Fig. 2 and discussed in the previous section). Two of the generating moves may result in integration of the equations of motion (i.e. a MD simulation). This is, compared to the other operations, the most expensive operation in the algorithm since this operation requires evaluation of the forces. In PyRETIS, the MD simulation is formally handled by a specific *class* which we refer to as MD "engines". The engine is responsible for performing the actual dynamics (including updating the forces) and PyRETIS implements both internal and external engines. The internal engines can be used to run pure PyRETIS simulations for simple models, e.g. a single particle in a user-defined potential or a diatomic molecule in a multi-particle two-dimensional Weeks-Chandler-Andersen system.[17] This is useful for testing and development of new algorithms. The external engine is used to interface PyRETIS with external MD programs such as GROMACS or CP2K. Since the MD simulation required by the rare event methods is the most expensive operation, it is also the most important method to optimize. Presently, there exists several highly optimized MD codes, both for classical and ab initio MD and the external engine in PyRETIS is designed to make use of such codes. In order to be as general as possible PyRETIS does not interface external codes by modifying the MD packages themselves, but it is rather used to control the execution of these codes. This design choice is motivated by having the flexibility to also include closed-source MD packages as external MD engines. The disadvantage of this approach is that there will be additional overhead due to the communication between PyRETIS and the external MD package. This will slow down the execution of the MD package compared to when the MD package is running without being controlled by PyRETIS. In order to avoid this problem, we are currently working towards patching selected open source MD packages in future releases of PyRETIS. Since the external engine is such an important feature of the PyRETIS package, we discuss the interface in more detail in the following.

3.1 Interfacing external MD packages

PyRETIS can interface external MD simulation packages and Fig. 3 presents the scheme for how PyRETIS connects to any external MD code. In the current implementation, PyRETIS is used to direct the external MD software so that trajectories can be generated. PyRETIS will not store the full trajectories (i.e. all positions, velocities etc. as function of time) in memory and will only store the location of the externally generated files containing the trajectories.

PyRETIS directs the execution of an external MD package via a specific Python class which defines the following methods:

- A method which executes the external MD software for generating/extending trajectories.
- A method which can read/write snapshots in given trajectories.

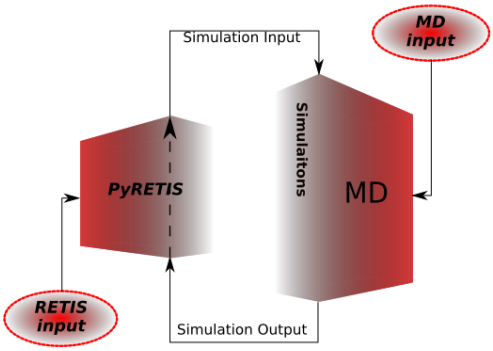


Figure 3: Scheme describing the PyRETIS–external simulation software information flow. PyRETIS communicates with and lead the external software without altering its force field/integration algorithm. PyRETIS selects the initial configuration and determines if the equations of motion are to be propagated forward or backward in time. This is passed to the external engine which then executes the MD steps. After completion of the MD steps, the external engine passes information about the order parameters obtained and the files created which contain the trajectory.

- A method for generating random velocities for a specific configuration using the external MD software. In simple cases (i.e. if there are no constrains between bounded atoms) PyRETIS can be used to generate these velocities.

For executing the external MD software, PyRETIS may need several input files which has to be provided by the user. The type and number of input files will depend on the MD simulation package, but they should define a MD simulation. Typically, these files contain information about the force field, topology, type of dynamics and so on. PyRETIS will make use these files as templates for setting up new MD simulations and only a small subset of the MD settings will be read (and possibly) modified by PyRETIS: the length of the MD simulation (i.e. the number of steps), the time step, temperature settings, and the frequency of output of the trajectory and energies. In some sense, from the point of view of PyRETIS, the external MD engine is treated as a "black box" and PyRETIS does not distinguish between executing classical MD software or ab initio MD software. The external MD software is not modified by PyRETIS and it can be executed as one would normally execute the software (e.g. in parallel when using a cluster). The specific command for executing the external software can be modified and provided by the user in the input to PyRETIS.

Thus, when MD integration is needed in the rare event algorithms, PyRETIS selects a configuration from a trajectory file which is passed to the external engine as an initial configuration. PyRETIS then executes the external MD engine which propagates the equations of motion until a stopping condition is reached. This stopping condition is defined by the TIS/RETIS algorithms and is either that the maximum number of integration steps is reached, or that the leftmost or rightmost interface has been crossed. If the maximum number of integration steps is reached, the external MD engine will exit by itself, while

PyRETIS will stop the external MD engine if the specified interface is crossed in order parameter space. We have currently implemented two variants of the checking of the crossing in order parameter space. In both variants the actual calculation of the order parameter is identical, but they differ in the way they interact with the external MD code. In the first variant, PyRETIS will start the external MD engine and let it run for a some steps, n_{sub}, (defined by the used in the input to PyRETIS) before the order parameter is calculated. If none of the interfaces were crossed, PyRETIS will then continue the simulation for n_{sub} additional steps before checking again. In the second variant, PyRETIS will simply start the execution of the external MD engine and continuously monitor and read new frames from the trajectory as they are written by the external MD engine. The advantage of the second variant over the first is that it avoids the overhead associated with stopping/starting the execution of the external MD package many times. The disadvantage is that it can be more difficult to implement. The first variant is then useful as a benchmark as the two approaches should give identical results (within machine precision) when they are employed by PyRETIS.

The actual calculation of the order parameter may in some cases also introduce a non-negligible overhead for the external MD engine. The order parameters considered in the examples in this paper are relatively simple and are negligible compared to the propagation by the MD engines. However, in some cases the calculation of the order parameter will be expensive, one specific example is the determination of clusters in relation to nucleation events. In some cases, the external MD engine can actually be capable of calculating and outputting the order parameter during the propagation. PyRETIS can the make use of this output if the order parameter, from the point of view of PyRETIS is defined to read this output from the external MD engine.

In this way, the external MD package is used to generate trajectory segments which is joined by PyRETIS to create full trajectories. If a new generated trajectory satisfies the ensemble requirements, the old trajectory is discarded in favor of the new one. For each trajectory, relevant information of the path are saved and for visualization purposes, and post-processing analysis, the full trajectories can also be saved. This scheme permits a complete computational separation of PyRETIS from the external MD package also allowing the eventual usage of multiple packages or multiple independent (parallel) simulations.

4 Input description

We show short examples of typical input to PyRETIS in Figs. 4, 5 and 6. In general, the input file is structured into sections and in each section, keywords are used to define settings. We refer to the on-line manual[40] for a more detailed description of the recognized sections and keywords. Further, we can classify the input sections into two main categories, defining the rare event method and the dynamics:

The rare event method settings This defines how the actual rare event simulation should be performed and a short example is given in Fig. 4. These settings do not define settings for the dynamics such as selection of the force field, type of dynamics (NVE, NVT, etc.), integration time step, etc.

```
pyretis example RETIS input
===========================

Simulation
----------
task = retis
steps = 20000
interfaces = [-0.9, -0.75, -0.65, -0.4, 1.0]

RETIS settings
--------------
swapfreq = 0.5
relative_shoots = None
nullmoves = True
swapsimul = True

TIS settings
------------
maxlength = 20000
freq = 0.5

Orderparameter
--------------
class = Position
dim = x
index = 0
name = Position
periodic = False
```

Figure 4: Example input file for a PyRETIS RETIS simulation. Here, we show the rare event specific settings which are given by the sections "Simulation", "RETIS settings", "TIS settings" and "Orderparameter". These sections define the rare event simulation, for instance are the positions of the RETIS interfaces given in the "Simulation" section by defining the "interfaces" keyword and the order parameter is defined by the keywords in the "Orderparameter" section. We refer to the on-line manual[40] for an explanation of all sections and keywords.

The MD input If an internal MD engine is used, the MD input includes the selection of the engine and a force field. A short example is given in Fig. 5 where an internal engine and a force field is set up. If an external MD engine is used, then the input to PyRETIS selects the external engine and describe additional input files required. The additional input files are the ones required by the external MD package and this is specific to the package. For instance, GROMACS, needs information about the topology/force field, the initial configuration and how to perform the actual MD. It is assumed that the user provide these, so that PyRETIS can make use of them. Typically, only the file containing the settings for actual MD will be read and used by PyRETIS as a template. In Fig. 6 we show a short input example which makes use of GROMACS.

5 Output description

For each path ensemble in a simulation, PyRETIS stores statistics for the generated trajectories. This includes information on the type of MC move performed, the maximum order

```
pyretis internal MD engine
==========================

Engine
------
class = Langevin
timestep = 0.002
gamma = 0.3
high_friction = False
seed = 0

Forcefield settings
-------------------
description = 1D double well test potential

Potential
---------
class = DoubleWell
a = 1.0
b = 2.0
c = 0.0
```

Figure 5: Example of a PyRETIS input file using an internal MD engine. The engine is selected and defined in the "Engine" section and a force field is set up in the "Forcefield settings" section, consisting of a single potential function as defined in the "Potential" section. The force field in this example consists of a single (internal) potential function which is of type "DoubleWell". In Fig. 7 we show the potential energy of this particular potential function with the given parameters ("a", "b", "c"). The on-line manual[40] contains information about the meaning of these parameters and other potential functions.

```
pyretis external MD engine
==========================

Engine
------
class = gromacs
gmx = gmx
mdrun = gmx mdrun
input_path = gromacs_input
timestep = 0.002
subcycles = 5
```

Figure 6: Example of a PyRETIS input file for using an external MD engine. The GROMACS external MD engine is here specified by defining the commands for executing it and the input files. The input files are stored in a separate folder named "gromacs_input" and is referenced with the "input_path" keyword. In this folder, PyRETIS expects to find the initial configuration, the topology information and the simulation settings for GROMACS. In addition, the number of sub-cycles is defined. This is the number of MD steps the engine will perform before PyRETIS halts the execution and calculates the order parameter. PyRETIS will then decide if additional MD steps are required or not. When using the external GROMACS engine, we do not specify a force field as this is handled by the input files supplied to the engine.

parameter reached, acceptance/rejection and so on. This information is needed to calculate the crossing probability. In addition, information about the trajectories themselves can be stored for each path ensemble. This will typically include the value of the order parameter(s), energies, positions and velocities as function of time on the trajectories. This information is not needed to calculate rates, however they can be very valuable for additional analysis. For large systems the storage of several trajectories can grow very large and the user can select the frequency by which they are written. PyRETIS only requires that the current accepted trajectory is stored and this trajectory is always kept separate from the additional trajectories the user may wish to store. For instance, the user can select to not write any additional trajectories at all and then, only the current accepted trajectory will be kept.

PyRETIS comes with a series of analysis tools to extract the most relevant information from the simulation output in terms of the transition event. After performing an analysis, PyRETIS can gather the results into a short report which is generated using a specified template. This template can be extended and customized by the user to tailor the reporting of the results. Currently PyRETIS is using matplotlib[41] for generating the figures and supports generation of reports in reStructuredText, HTML or LaTeX.

While different order parameters can be considered and implemented, the main PyRETIS analysis is based on the progress coordinate which describes the rare event. During a simulation, information about rejected and accepted trajectories are stored for each ensemble on disk for each simulation cycle. The initial, final, maximum and minimum value of the progress coordinate are saved for each path. For each path ensemble $[i^+]$ the following results are obtained:

(a) The crossing probability as a function of the progress coordinate.

(b) A running average of the crossing probability at λ_{i+1} as a function of the number of cycles.

(c) An error analysis of the crossing probability using block averaging.

(d) Distribution of lengths of the generated paths.

(e) The distribution along the progress coordinate of the shooting points.

Furthermore, PyRETIS reports the relative sampling efficiency of the simulation, the average length of accepted and rejected paths, the acceptance rate and the correlation number of generated paths. The estimated correlation (also called the *statistical inefficiency*), N_{cor}, is obtained as the square of the estimated relative error divided by the square of the block error obtained for a block size of 1. For the RETIS method, the reported errors are indicative as they do not include the effect of correlations due to the swapping move.

As an illustrative example, the results (a)–(c) are shown below in Fig. 8 for a RETIS simulation of a single particle moving in a 1D potential, identical to the potential shown in Fig. 7. The RETIS simulation was performed as previously reported,[42] with a temperature taken equal to 0.12 (dimensionless units) and interfaces $\lambda_0 = -0.90$, $\lambda_1 = -0.75$, $\lambda_2 = -0.65$, $\lambda_3 = -0.40$, $\lambda_4 = 1.0$) defining 5 path ensembles ($[0^-]$, $[0^+]$, $[1^+]$, $[2^+]$, $[3^+]$).

The crossing probabilities as functions of the progress coordinate (a) are shown in the first columns of Fig. 8. The fractional crossing probability for each ensemble of crossing the next

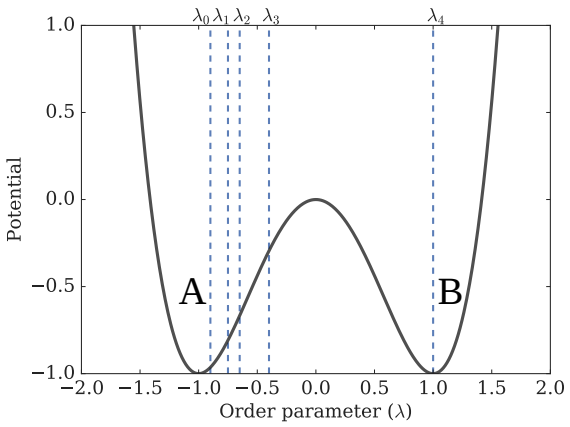


Figure 7: The 1D double well potential defined by the settings given in the example in Fig. 5. The reactant and product states are identified (labeled "A" and "B" respectively) and interfaces for a RETIS simulation (λ_i) are shown with vertical dashed lines. In this particular example, the order parameter is taken as the position in the potential.

interface is explicitly considered in the plots of crossing probability versus number of MD cycles (b) in Fig. 8, central column. The plots are essential indicators of the computational convergence of the computed crossing probability. From a comparative analysis, the most difficult regions to sample in the path space can be identified (in this example, the third ensemble has the slowest convergence). A third plot (Fig. 8, right column), reports the relative error as a function of the averaging block size (c) used to compute the crossing probability and the correlation between paths. Depending on the behavior of the system (fluctuations or slow drift), it provides a criterion to determine an appropriate average block length to quantify the local probability and the correlation between paths. Plots (a)-(e) are automatically generated for each ensemble, enabling a quantitative visualization of the crossing probabilities. Dedicated modification, or simply more intensive computations can, therefore, be directly addressed to the most demanding interval in the progress coordinate space.

An optimal interface positioning minimizes the statistical error while maximizing the computational efficiency in the estimation of the rate of the transition event. As a general indication, an interface should be positioned such that $\mathcal{P}_{\mathrm{A}}(\lambda_{i+1}|\lambda_i) \approx 0.2$.[43] The first interface also defines the limit of state A. A general approach to define its location is to perform a regular MD simulation starting within state A, and track, step by step, the value of the order parameter (assuming that the transition start from a stable state, which is desirable but not strictly required). From this simulation, a graph of residence time versus order parameter can be drawn. The first interface can then be located at the order parameter value such that only the 20% of the simulation time is spent at larger order parameter values. Thereafter, an arbitrary number of interfaces can be positioned as an initial attempt and a TIS/RETIS simulation is performed. After this first exploration of the path space, following the 0.2 crossing probability guideline, the interfaces can be changed (both in position and number) for a better sampling efficiency (eventually discarding the previous results). TIS/RETIS provides the same results regardless the number and position of the interfaces for infinitely

Table 1: Summary of crossing probabilities (P_{cross}) for the path ensembles as generated by the PyRETIS analysis tool for the 1D potential described in the main text. The reported absolute error and relative error are obtained from the block error analysis.

Ensemble	P_{cross}	Error	Rel. error (%)
$[0^+]$	0.275527	0.003722	1.350769
$[1^+]$	0.302107	0.005891	1.949818
$[2^+]$	0.040280	0.002657	6.596495
$[3^+]$	0.084479	0.005571	6.594036

long simulations. A judicious approach is required, therefore, in selecting the number and location of interfaces, and even their eventual repositioning, to maximize the sampling efficiency. In the example reported in Fig. 8, it can be noted (first column of graphs) that the first interfaces (dashed lines) are positioned following approximately the 0.2 probability guideline while in third row ($[2^+]$), the interface position gives a lower probability. For higher probability values, new trajectories have little contribution to reduce the statistical error while the opposite is true for lower crossing probability values which require a larger number of trajectories to reduce the relative error. This is reflected in the results shown for the crossing probabilities and the relative errors (see table 1). For the given case, an additional interface positioned around -0.5 would have reduced the error in the second last ensemble, increasing the sampling efficiency.

From Fig. 8 it can be assumed that the computations have converged, but longer simulations might be needed to reduce the statistical errors. Numerical values for the crossing probabilities, with error estimates, are also reported in tables as exemplified in table 1. In addition, the *overall probability* is also reported and the rate constant is obtained as exemplified in Fig. 9. The matching of the histograms is here done by matching on a single point. This implies that the crossing probabilities shown in Fig. 8 are scaled such that they do not start with the value 1, but at a value equivalent to the product of the previous crossing probabilities. This implies for this example (see Fig. 8 and table 1) that the crossing probability of $[1^+]$ is rescaled with a factor 0.275527, the crossing probability of $[2^+]$ is rescaled with a factor 0.275527×0.302107 etc. A somewhat more accurate evaluation of the overall crossing probability can be made using the weighted histogram analysis method (WHAM)[44–46] as was done in Refs. 47 and 48 The implementation of WHAM in PyRETIS is presently only available in a development version, but will soon become available in an open release.

6 Extending PyRETIS: Adding a new order parameter

PyRETIS has been designed so that it is easy to extend the program with new progress coordinates, force fields or to interface it with other external MD packages. Due to the dynamic nature of Python, such extensions can be added without a recompilation of the PyRETIS library. Further, these extensions can be written in Python or other languages. In this section we briefly discuss how a new progress coordinate can be added.

PyRETIS internally computes the progress coordinate and additional order parameters for each system. These quantities are in general functions of all positions and velocities in

the system. In PyRETIS, new order parameters can be added by creating a custom order parameter *class* which is then referenced in the input to PyRETIS. At run time, new order parameters are dynamically loaded by PyRETIS and used in the simulation. In Fig. 10 we give a short example on how a custom order parameter is created. We note that the order parameter can be implemented relatively freely in the sense that it can be implemented in C/C++/FORTRAN as well as in Python. For the former options, a Python wrapper is needed in order for PyRETIS to be able to use the order parameter. It can also make use of scientific Python libraries such as Numpy[30] or SciPy.[29] In principle, since Python can call external programs, a custom order parameter can also make use of external programs for the actual computation if this is convenient for the user. Again we refer to the on-line manual[40] for specific examples.

7 PyRETIS examples

To show some of PyRETIS features and capabilities, we briefly describe four different applications. In the first case, the internal MD engine has been used to simulate a particle moving in a 2D potential surface.[20] The second and third cases are examples of using CP2K[21] for simulating hydrogen splitting and proton transfer in water respectively. Finally, we show an example of using GROMACS[22] for studying the diffusion of a methane molecule into a water hydrate. We note that the aim of this section is to illustrate different features of PyRETIS and not to report on extensive results and PyRETIS input settings for the case studies we show here. For the latter, we refer to the on-line documentation[40] where we describe these examples in more detail together with additional examples.

7.1 Internal MD: A 2D potential with hysteresis

The PyRETIS internal MD engine can run simple MD simulations with user defined potential functions. Here we consider a 2D potential, $V(x, y)$, given by

$$\begin{aligned} V(x,y) =& (x^2+y^2)^2 - 10\mathrm{e}^{-30(x-0.2)^2-3(y-0.4)^2} \\ & -10\mathrm{e}^{-30(x+0.2)^2-3(y+0.4)^2}, \end{aligned} \tag{4}$$

and depicted in Fig. 11. Depending on how the progress coordinate is selected, rare event methods will have different efficiencies. Specifically free energy based methods like umbrella sampling and thermodynamic integration will suffer from hysteresis when the y-coordinate is chosen as progress coordinate. TIS/RETIS is not suffering from this hysteresis problem,[39] but still a drop in the acceptance of the shooting move is expected which makes it more difficult to effectively sample reactive trajectories. This particular potential has been used as a test case in the development of new algorithms[20] based on RETIS which are able to alleviate this problem.

Via the implementation of such simple potentials and case studies, students and developers can have direct access to simulation cases suitable for computational tests and method development. The internal engine constitutes an educational tool providing a simple interface to understand and gain experience with PyRETIS and the rare event algorithms

without considering expensive simulations or high-dimensional and complex systems. Such case studies have a relatively low computational cost which means that small and portable computers can also be used to perform test cases.

7.2 Simulations interfacing CP2K

A second case of study exemplifies the capability of PyRETIS to interface with ab initio MD codes, such as CP2K. CP2K[21] is a quantum chemistry and solid state physics software that can perform atomistic simulations using DFT with the mixed Gaussian and plane waves basis sets.

We first report, as a minimal application of the CP2K–PyRETIS interface, the splitting of H_2 (Fig. 12). The dissociation of H_2 has been followed with a progress coordinate equal to the distance between the hydrogen atoms. It should be noted that this case of study is here included purely as an educational and computationally accessible example that can be even launched on a laptop (even ignoring spin issues to increase the computational speed).

The second application reported here consists of proton transfer in water. We refer to van Erp et al. for simulation details of the proton transfer study in a gas-phase water cluster of 8 water molecules.[48] This study has been extended to liquid water systems, composed of 32 water molecules, which requires extensive computations on computer clusters. An illustration of the system is shown in Fig. 13. In this case of study, the progress coordinate was defined using all distances between oxygen and hydrogen atoms. Each hydrogen atom was assigned to the closest oxygen atom. This allowed classification of the molecules as H_2O, H_3O^+ or OH^-. If the system only contains H_2O molecules, the progress coordinate is defined as the longest hydrogen-oxygen bond length. If the system contains H_3O^+ and OH^- species, the order parameter is taken as the shortest distance from the oxygen in OH^- to a hydrogen in H_3O^+. Within the RETIS method, definition of such progress coordinates is relatively straightforward. The RETIS algorithm provided the means to generate hundreds of proton transfer reactions from which the rate constant could be estimated.

Further application of RETIS with CP2K have been already completed and detailed in the studies of silica oligomerization[49] and water autoionization.[48]

7.3 Simulations interfacing GROMACS

GROMACS[22] is currently one of the most widely used packages to perform classical MD simulations. The large user base has been obtained thanks to its computational speed and relative efficient scaling on large clusters, allowing the simulation of millions of particles for relative long times. We have therefore prioritized GROMACS and included it as one of the supported external packages in PyRETIS.

A study on the methane diffusion between water cages of the sI hydrate (see Fig. 14), has been performed via RETIS and GROMACS. In the simulation, the progress coordinate has been defined as the distance of the methane molecule to the center of the water ring, along the vector normal to the water ring plane. The study allows the determination of the rate of transport of methane and its relation to movements of the water molecules composing the rings.

8 Conclusions and future work

In this article, we have presented PyRETIS a library for performing rare event simulations based on TIS and RETIS. PyRETIS has been explicitly interfaced with CP2K and GROMACS in order to efficiently sample trajectories. This interface can be extended to other molecular simulation packages. As noted, while our current strategy for interfacing external MD packages is flexible and generic (so that any free or closed-source package can be interfaced), it leads to additional overhead due to the communication between PyRETIS and the external package. We are currently working towards patching of open source MD packages, which will allow us to more efficiently generate trajectories using external programs. This will be included in future releases of the library.

Further, we aim to introduce new efficient algorithms based on RETIS and new analysis methods which we are currently developing. It is also a long-term goal for the PyRETIS project to implement a parallel RETIS algorithm.

The PyRETIS project is a collaborative open source project and new developers are welcome to participate and contribute to the project.

9 Availability

PyRETIS is free (released under a LGPLv2.1+ license) and can be obtained as described at `http://www.pyretis.org/user/install.html`. The latest release can be installed via the Python Package Index[50] and the source code and development version is accessible at: `https://gitlab.com/pyretis/pyretis` The website gives a detailed description of the usage of PyRETIS and show example calculations that can be carried out with the package.

10 Acknowledgments

The authors thank the Research Council of Norway for funding (QuanTIS, project number 237423) and the Olav Thon foundation for the support in the development of interactive visualization tools to facilitate the teaching of rare event methods. A.L. thanks the Research Council of Norway, project number 250875 for support. This research was supported in part with computational resources at NTNU provided by NOTUR, http://www.sigma2.no, project number NN9254K. Magnus Heskestad Waage is thanked for providing the methane–water sI clathrate system and commenting on the manuscript. Mahmoud Moqadam is thanked for providing the system used for autoionization of water.

References

[1] D. Marx, M. E. Tuckerman, J. Hutter, and M. Parrinello, Nature **397**, 601 (1999).

[2] M. Karplus and J. A. McCammon, Nat. Struct. Mol. Biol. **9**, 646 (2002).

[3] S. P. Adiga and D. W. Brenner, Nano Lett. **5**, 2509 (2005).

[4] R. O. Dror *et al.*, Annu. Rev. Biophys. **41**, 429 (2012).

[5] A. C. Pan, D. W. Borhani, R. O. Dror, and D. E. Shaw, Drug Discov. Today **18**, 667 (2013).

[6] J. Li, D. Liao, and S. Yip, Phys. Rev. E **57**, 7259 (1998).

[7] E. Riccardi and A. I. Liapis, J. Sep. Sci. **32**, 4059 (2009).

[8] E. Riccardi, J.-C. Wang, and A. I. Liapis, J. Chem. Phys. **140**, 084901 (2014).

[9] E. Riccardi, M. C. Böhm, and F. Müller-Plathe, Eur. Phys. J. E **37**, 103 (2014).

[10] M. Iannuzzi, A. Laio, and M. Parrinello, Phys. Rev. Lett. **90**, 238302 (2003).

[11] A. Laio and F. L. Gervasio, Rep. Prog. Phys. **71**, 126601 (2008).

[12] A. F. Voter, J. Chem. Phys. **106**, 4665 (1997).

[13] A. F. Voter, Phys. Rev. Lett. **78**, 3908 (1997).

[14] A. F. Voter and M. R. Sørensen, Mat. Res. Soc. Symp. Proc. **538**, 427 (1999).

[15] H. Grubmüller, Phys. Rev. E **52**, 2893 (1995).

[16] F. G. Wang and D. P. Landau, Phys. Rev. Lett. **86**, 2050 (2001).

[17] T. S. van Erp, D. Moroni, and P. G. Bolhuis, J. Chem. Phys. **118**, 7762 (2003).

[18] T. S. van Erp, Phys. Rev. Lett. **98**, 268301 (2007).

[19] C. Dellago, P. G. Bolhuis, F. S. Csajka, and D. Chandler, J. Chem. Phys. **108**, 1964 (1998).

[20] E. Riccardi, O. Dahlen, and T. S. van Erp, **Submitted**, (2017).

[21] J. Hutter, M. Iannuzzi, F. Schiffmann, and J. VandeVondele, Wiley Interdiscip. Rev. Comput. Mol. Sci. **4**, 15 (2014).

[22] M. J. Abraham *et al.*, SoftwareX **1–2**, 19 (2015).

[23] S. Plimpton, J. Comput. Phys. **117**, 1 (1995).

[24] G. Y. Sun *et al.*, J. Mol. Struc.-Theochem **624**, 37 (2003).

[25] A. Arnold *et al.*, in *Meshfree Methods for Partial Differential Equations VI*, edited by M. Griebel and M. A. Schweitzer (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013), pp. 1–23.

[26] H. J. Limbach, A. Arnold, B. A. Mann, and C. Holm, Comp. Phys. Comm. **174**, 704 (2006).

[27] Y. Shao *et al.*, Mol. Phys. **113**, 184 (2015).

[28] TIOBE - The Software Quality Company, TIOBE Index, `http://www.tiobe.com/tiobe-index/`, accessed: 07-04-2017.

[29] E. Jones *et al.*, SciPy: Open source scientific tools for Python, `http://www.scipy.org`, 2001–, accessed: 07-04-2017.

[30] S. van der Walt, S. C. Colbert, and G. Varoquaux, Comput. Sci. Eng. **13**, 22 (2011).

[31] K. Hinsen, J. Comput. Chem. **21**, 79 (2000).

[32] P. Eastman *et al.*, J. Chem. Theory. Comput. **9**, 461 (2013).

[33] N. Michaud-Agrawal, E. J. Denning, T. B. Woolf, and O. Beckstein, J. Comp. Chem. **32**, 2319 (2011).

[34] R. McGibbon *et al.*, Biophys. J. **109**, 1528 (2015).

[35] C. Dellago, P. G. Bolhuis, F. S. Csajka, and D. Chandler, J. Chem. Phys. **108**, 1964 (1998).

[36] P. G. Bolhuis, D. Chandler, C. Dellago, and P. L. Geissler, Annu. Rev. Phys. Chem. **53**, 291 (2002).

[37] D. Frenkel and B. Smit, *Understanding molecular simulation, 2nd ed.* (Academic Press, San Diego, CA, 2002).

[38] A. Lervik and T. S. van Erp, J. Chem. Theory Comput. **11**, 2440 (2015).

[39] T. S. van Erp, J. Chem. Phys. **125**, 174106 (2006).

[40] A. Lervik, E. Riccardi, and T. S. van Erp, The PyRETIS manual, `http://www.pyretis.org`, accessed: 07-04-2017.

[41] J. D. Hunter, Comput. Sci. Eng. **9**, 90 (2007).

[42] T. S. Van Erp, in *Kinetics and Thermodynamics of Multistep Nucleation and Self-Assembly in Nanoscale Materials: Advances in Chemical Physics Volume 151*, edited by G. Nicolis and D. Maes (John Wiley & Sons, Inc., New Jersy, USA, 2012), pp. 27–60.

[43] T. S. van Erp and P. G. Bolhuis, J. Comput. Phys. **205**, 157 (2005).

[44] A. M. Ferrenberg and R. H. Swendsen, Phys. Rev. Lett. **63**, 1195 (1989).

[45] S. Kumar *et al.*, Journal of Computational Chemistry **13**, 1011 (1992).

[46] B. Roux, Comput. Phys. Commun. **91**, 275 (1995).

[47] J. Rogal *et al.*, J. Chem. Phys. **133**, 174109 (2010).

[48] T. S. van Erp, M. Moqadam, E. Riccardi, and A. Lervik, J. Chem. Theory Comput. **12**, 5398 (2016).

[49] M. Moqadam *et al.*, J. Chem. Phys. **143**, 184113 (2015).

[50] The Python Software Foundation, PyPI - the Python Package Index, `https://pypi.python.org/pypi`, accessed: 07-04-2017.

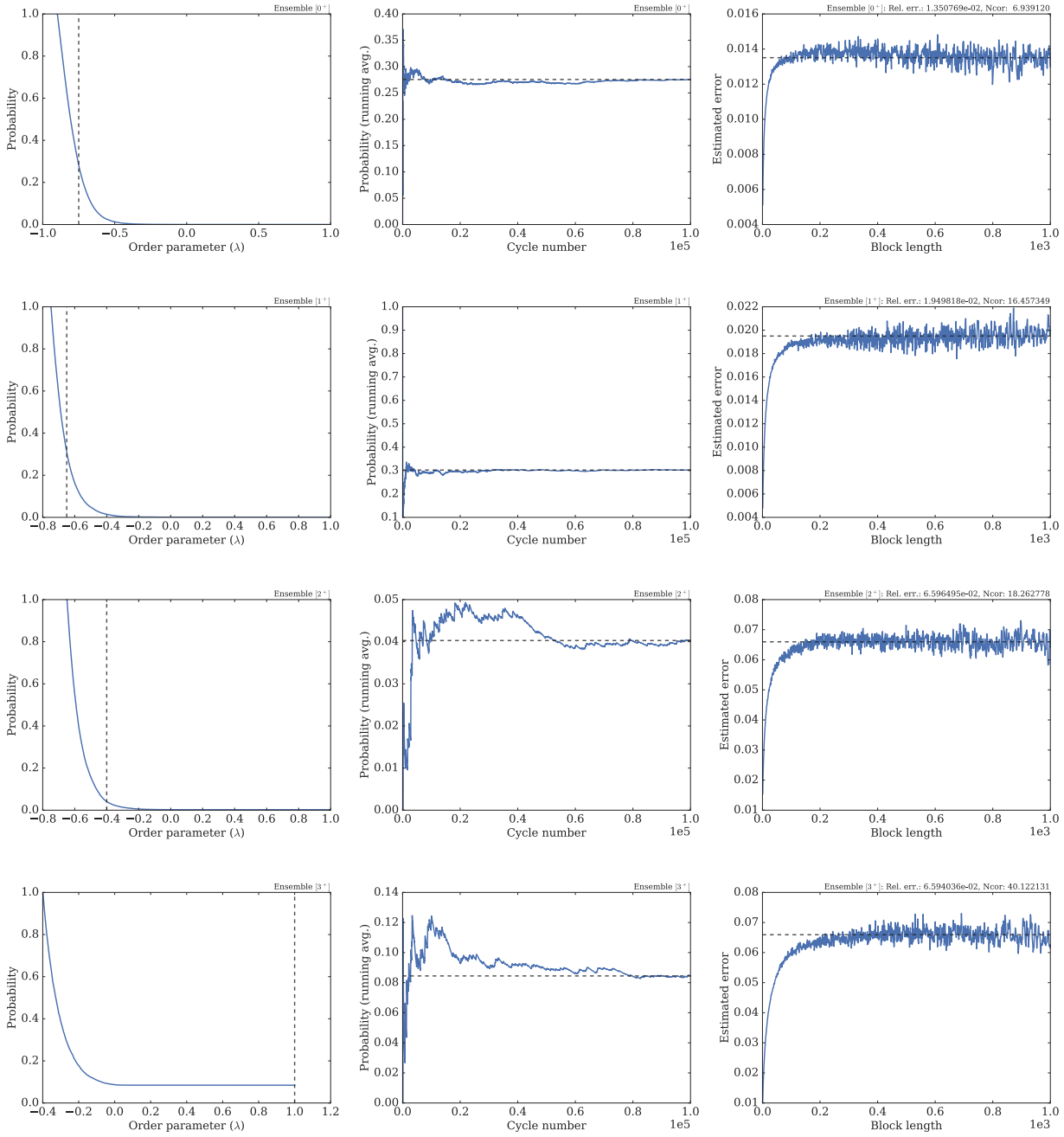
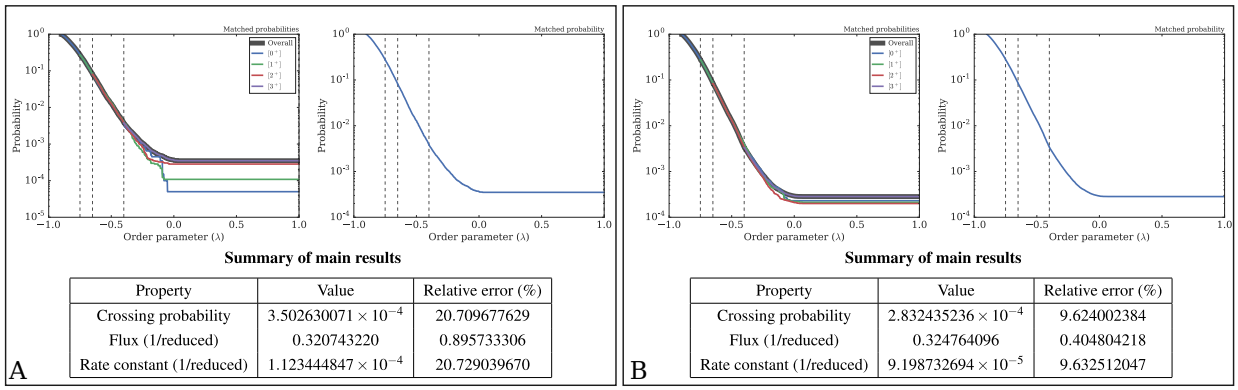


Figure 8: Example of analysis plots generated by PyRETIS for a case with four path ensembles, $[0^+]$–$[3^+]$ for the 1D potential simulation described in the main text. Each row corresponds to the results from a single path ensemble. From left to right: crossing probability as a function of the progress coordinate, the running average of the crossing probability at interface λ_{i+1} and the error in the crossing probability obtained by a block error versus average block size. It can be noted that each crossing probability starts at 1 (by the definition of the path ensemble crossing probability) and reach a value of 0, except for the last ensemble where some trajectories successfully reach the product state. In these crossing probabilities, the crossing of the next interface is also shown which defines the crossing probability for each ensemble. It is the running average of this crossing probability that is shown in the second column, and its error is shown in the third column.



Summary of main results

Property	Value	Relative error (%)
Crossing probability	$3.502630071 \times 10^{-4}$	20.709677629
Flux (1/reduced)	0.320743220	0.895733306
Rate constant (1/reduced)	$1.123444847 \times 10^{-4}$	20.729039670

Summary of main results

Property	Value	Relative error (%)
Crossing probability	$2.832435236 \times 10^{-4}$	9.624002384
Flux (1/reduced)	0.324764096	0.404804218
Rate constant (1/reduced)	$9.198732694 \times 10^{-5}$	9.632512047

Figure 9: Example of overall analysis output as generated by PyRETIS for the 1D potential case described in the main text. The analysis was performed after 20 000 steps (panel A) and after 100 000 steps (panel B). The overall crossing probability is obtained as a function of the progress coordinate by matching and aligning the partial crossing probabilities (the probabilities shown in the first column in Fig. 8). In the leftmost figures in the panels, all the individual probabilities are shown, while only the overall crossing probability is shown in the rightmost figures. The main results from the analysis (overall crossing probability, flux and rate with their relative errors) are reported in tables as shown at the bottom of the panels.

A. File "distorder.py" defining a new order parameter:

```python
"""Defining a custom order parameter."""
import numpy as np
from pyretis.orderparameter import OrderParameter


class DistOrder(OrderParameter):
    def __init__(self, index1, index2):
        """Example of a distance order parameter.

        The distance is calculated between two
        particles selected by providing two
        indices.

        Parameters
        ----------
        index1 : integer
            Index in the particle list for
            particle 1.
        index2 : integer
            Index in the particle list for
            particle 2.
        """
        # Give the parameter a name which may be
        # used for output to screen etc.:
        name = 'DistOrder'
        super(DistOrder, self).__init__(name)
        self.index1 = index1
        self.index2 = index2

    def calculate(self, system):
        """Calculate the order parameter."""
        particles = system.particles
        delta = (particles.pos[self.index1] -
                 particles.pos[self.index2])
        return np.sqrt(np.dot(delta, delta))
```

B. Input settings for using the new order parameter:

```
Orderparameter
---------------
class = DistOrder
index1 = 1
index2 = 2
module = distorder.py
```

Figure 10: Example showing how a new order parameter is created. Panel A show the Python code needed to create a new order parameter for PyRETIS: The new order parameter needs to subclass the generic order parameter class defined in PyRETIS. Further, a method needs to be defined which handle the actual calculation of the order parameter taking the "system" object from PyRETIS as its input. We refer to the on-line manual[40] for a more detailed explanation, but note that we are here free to make use of any Python library (e.g. Numpy[30] as shown here), external extensions (e.g. Python extensions written in C) or even external programs. Panel B show the PyRETIS input needed to make use of the new order parameter. The "module" keyword specifies the location of the file containing the new order parameter.

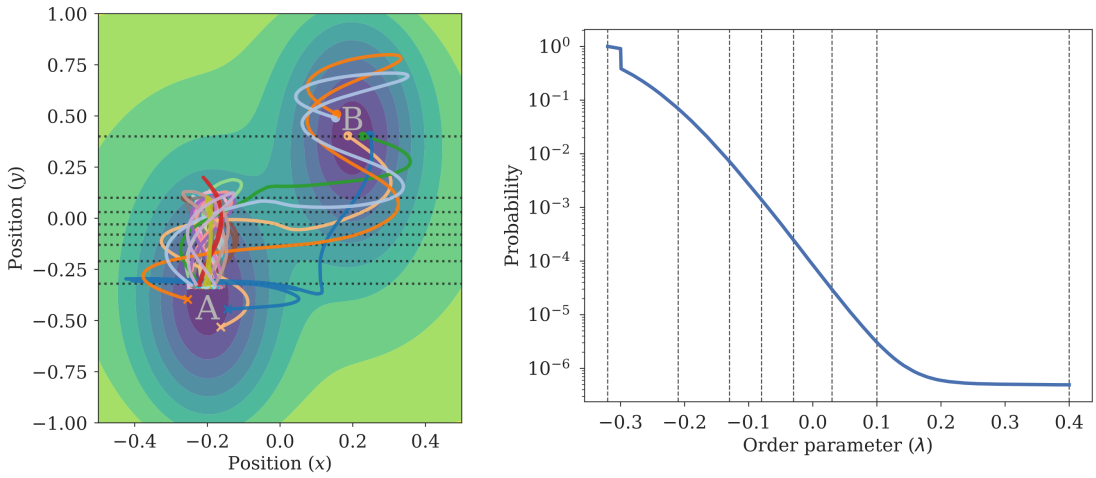


Figure 11: (Left) The 2D potential energy as a function of the two Cartesian coordinates x and y. In this potential several order parameters can be considered, for instance $\lambda = x$ or $\lambda = y$. Here, we have used $\lambda = y$ together with an extra energy criterion to identify the stable states. The energy criterion is only used close to the first and last interfaces and requires that the total energy is less than -9 (reduced units) for the system to be considered as being in state A or in state B. Interfaces were positioned as shown in the figure (dashed horizontal lines) and the solid lines are different trajectories as obtained in the simulations for the last path ensemble. Crosses indicate starting points and circles indicate ending points for the trajectories. (Right) The crossing probability obtained from PyRETIS simulations. The jump in the crossing probability close to the first interface is due including the energy term in the order parameter (the similar jump close to the last interface is not visible on the logarithmic scale). The vertical dashed lines indicate the interface positions. Simulation settings and input files are available on-line at: `http://pyretis.org/examples/examples-2d-hysteresis.html`

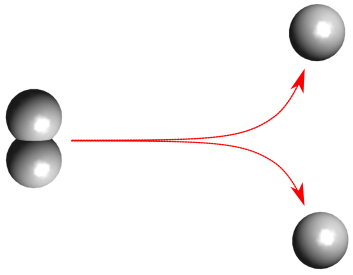


Figure 12: The H_2 dissociation studied. In this example, the progress coordinate is defined as the distance between the two atoms. Simulation settings and input files are available on-line at: `http://pyretis.org/examples/examples-cp2k-hydrogen.html`

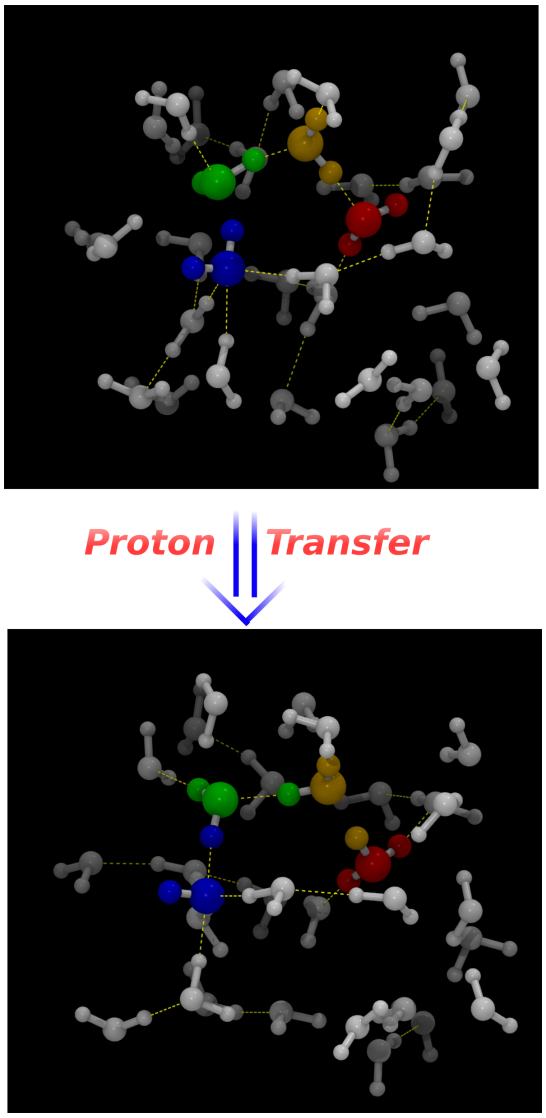


Figure 13: Proton transfer in a liquid water cluster. In blue, green, yellow and red the water molecules participating to the proton transfer are highlighted. The yellow lines show the respective hydrogen bonds. In the top panel, the water molecules are shown before the proton transfer. The lower panel shows the system after the proton transfer with highlighted water molecules being part of the proton transfer reaction chain. This example illustrates how reaction mechanisms can be studied from the generated trajectories in RETIS simulations.

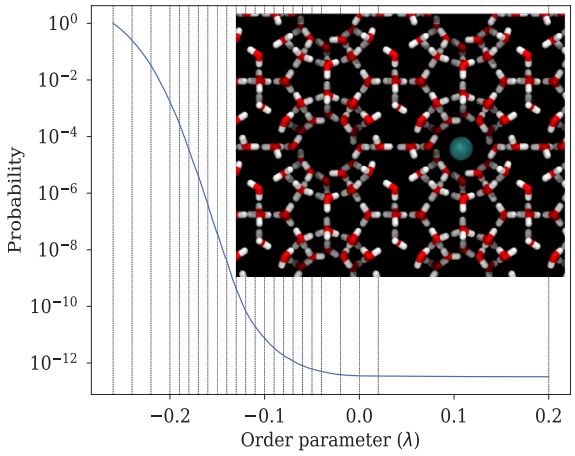


Figure 14: The sI hydrate structure studied in the methane diffusion example. The methane molecule (color cyan) can diffuse between cages, through 5- or 6-membered rings, via a hopping mechanism. In this study water is modeled using TIP4P and methane as a single united-atom particle. This example requires the positioning of several interfaces as shown in the figure by the dashed lines. The full PyRETIS settings and inputs are available at `http://pyretis.org/examples/examples-gromacs-hydrate.html`