Ensembler: A Simple Package for Fast Prototyping and Teaching Molecular Simulations

Benjamin Ries, Stephanie M. Linker, David F. Hahn, Gerhard König, and Sereina Riniker*

Laboratory of Physical Chemistry, ETH Zürich, Vladimir-Prelog-Weg 2, 8093 Zürich, Switzerland

E-mail: sriniker@ethz.ch

Abstract

Ensembler is a Python package that enables method prototyping using 1D and 2D model systems, and allows to deepen the understanding of different molecular dynamics (MD) methods, starting from basic techniques to enhanced sampling and free-energy approaches. The ease of installing and using the package increases shareability, comparability, and reproducibility of scientific code developments. Here, we describe the implementation and usage of the package and provide an application example for free-energy calculation. The code of Ensembler is freely available on GitHub *https://github.com/rinikerlab/Ensembler*.

Introduction

New simulation methods are routinely tested on simple model systems. They provide valuable insights into the theory, conceptual advantages and limitations of the methods. While the results of new approaches are published, the implementation details may not always be

available or difficult to use with different computer infrastructures. As a result, sharing, reproducing, understanding, and comparing simulation methodologies is often cumbersome.[1] To address this issue, we have developed the Ensembler package, an easy-to-use, yet powerful platform in Python3[2] that enables fast prototyping of new methods and comparison against existing techniques using 1D or 2D systems. In contrast, the C/C++[3] code of traditional high-performance molecular dynamics (MD) packages is more efficient but also much more complex.

Ensembler is designed following the recommendations of Stodden *et al.*[4] for the enhanced reproducibility of computational methods, which includes making code publicly accessible, providing documentation, and using open licensing.[4] Furthermore, Ensembler uses state-of-the-art software engineering tools (i.e. git,[5] MolSSI cookie-cutter,[6] and binder[7]) to fulfill these recommendations and enable features like continuous integration and the transparent versioning of the code.

The methods currently available in Ensembler are:

- *Model systems*: Harmonic oscillators as well as dihedral-angle, double-well, and Lennard-Jones potential-energy functions[8]
- *Sampling algorithms*: Conjugated gradient[9] for energy minimization, Metropolis Monte Carlo (MC),[10] leap-frog integration[11] for MD, and Langevin integration[12] for stochastic dynamics (SD)
- *Enhanced sampling techniques*: Umbrella sampling,[13] simulated tempering/temperature replica-exchange simulations,[14] local elevation/metadynamics,[15,16]
- *Free-energy methods*: Free-energy perturbation (FEP),[17] Bennett's acceptance ratio (BAR),[18] thermodynamic integration (TI),[19] enveloping distribution sampling (EDS),[20–22] λ-EDS,[23] replica-exchange EDS (RE-EDS),[24] and conveyor-belt TI[25]

Simple model systems can also be used for teaching MD concepts to students, as they allow to intuitively understand fundamental concepts.[26] Ensembler is well suited for didactic

purposes because it is not only easy to use, but supports also a range of visualizations, i.e. interactive widgets, animations, and plots, which can be embedded in Jupyter notebooks.[27] Example Jupyter notebooks[27] are provided in the Ensembler GitHub repository.

Implementation

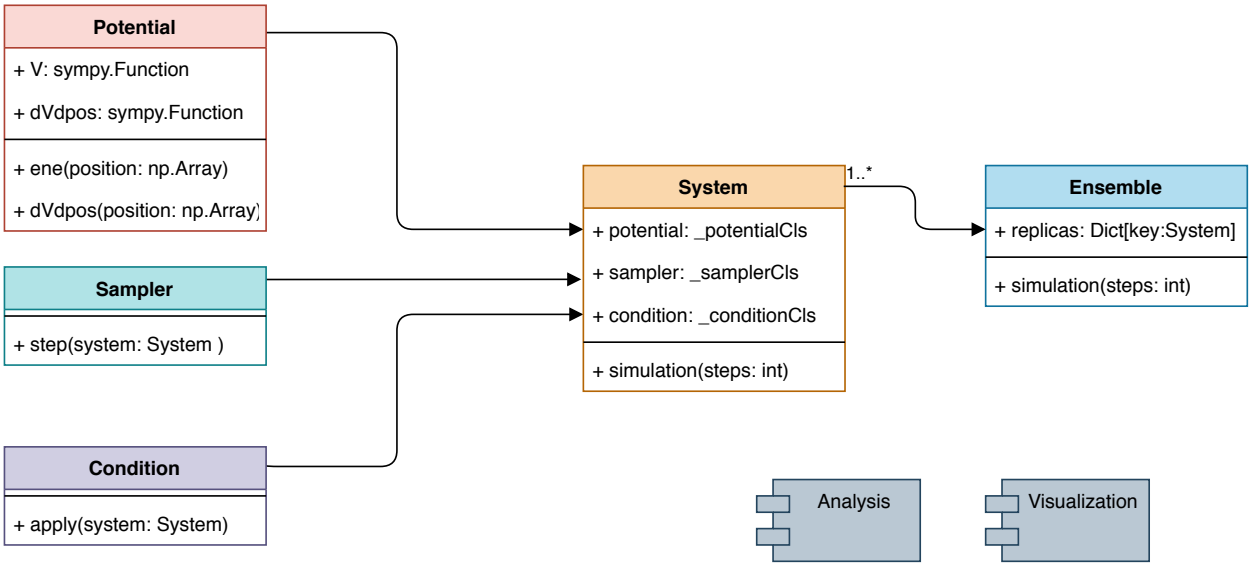


Figure 1: Unified modeling language (UML) class diagram of the five Ensembler base classes. *Potential class* (red) defines the potential-energy functions to be explored and generates the required derivatives automatically. The implementation of this class is based on the symbolic mathematical language of SymPy.[28] *Sampler classes* (cyan) are used for the sampling of potential-energy functions. *Condition classes* (purple) can have different functions, e.g. application of periodic boundary conditions,[29,30] thermostats, or restraints. *System classes* (orange) serve as the scaffold for the potential, sampler, and condition classes. In this structure, all components, parameters, and the results of a simulation are stored. *Ensemble class* (blue) is required for advanced simulation techniques, e.g. using multiple walkers that explore the energy landscapes of the same or different systems as in replica-exchange approaches.[14,31,32] *Analysis package* includes free-energy estimators such as the Zwanzig equation[17] or BAR.[18] The visualization functions in the *visualization package* enable an intuitive way of inspecting the simulation results.

Ensembler is implemented in Python3[2] and available on GitHub[33] (*rinikerlab/Ensembler*). The repository is based on the template of the MolSSI cookie-cutter[6] and comprises a code folder, an example folder for tutorials, example models contained in the provided Jupyter

notebooks,[27] an automatic pytest suite,[34] and the automatically generated sphinx[35] documentation. The code is continuously integrated via GitHub Actions,[36] providing information about code quality, test correctness, test coverage, and generation of an up-to-date documentation. Ensembler uses only open-source packages like the SciPy library[28,37–40] and Jupyter notebooks.[27] In the following, a user and a developer perspective are provided for the code structure.

User level: A simulation model in Ensembler consists of a potential class, a sampler class, and a system class wrapping the potential and the sampler (Figure 1), and provides control over the simulation approach. Additionally, multiple condition classes can be added that directly influence the simulation (e.g. periodic boundary condition[29,30] or thermostatting[41]). After the construction of the system, the simulation can be started directly with the *simulate* function. The resulting trajectory is in the form of a Pandas data frame.[39] The trajectory is thus easily compatible with other packages like NumPy[38] or scikit-learn[42] and can be stored in different formats, e.g. as .csv or .hf5 file. The system itself can be stored directly via the save function using serialization of the object with the Python package pickle. In most cases, only a few additional lines are needed to go from simple simulation technique to more advanced one, as shown below.

Developer level: The code of Ensembler is built on five interface-like base classes that allow extensive use of the inheritance concept and polymorphism[3] throughout the package. These fundamental classes are *potential*, *sampler*, *condition*, *system*, and *ensemble* (Figure 1), which can be grouped into three layers. *Potential*, *sampler*, and *condition classes* form the primary layer, providing different techniques to be used as components in a simulation. *Potential classes* provide the potential-energy functions in a symbolic form using SymPy,[28] enabling automatic on-the-fly derivation and simplification of the potential-energy function. *Sampler classes* are used to explore the potential-energy function (e.g. conjugate gradient,[9] Metropolis MC,[10] or leap-frog[11] integration). A new method can easily be implemented by inheriting from the *sampler class* and overwriting a single function called *step*. Finally,

condition classes provide additional functionalities such as thermostatting[41] and periodic boundary conditions[29,30]). New techniques can be implemented by inheriting the base *condition class* and overwriting the function *apply*. In the second layer, the first-layer components are wrapped into one *system class* that executes the simulation(s) and manages the input and output. An optional higher-order layer is available in form of the *ensemble class*, which allows the user to perform simulations with replica exchange.[14,24,31,32] If additional parameters are needed in a newly designed class, the constructor of the new child class can be adapted but must call the parent constructor.

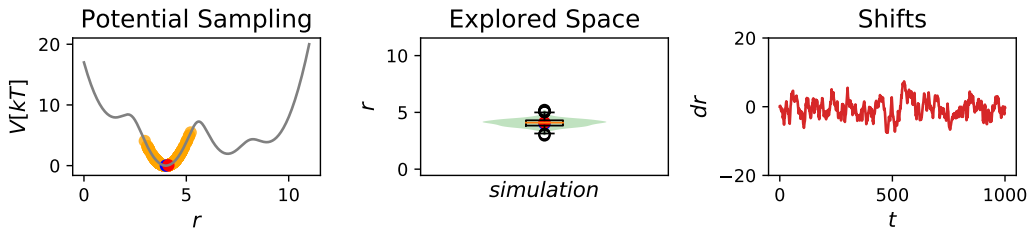
Applications and Examples

The code for the examples shown below can be found in the GitHub repository *https://github.com/rinikerlab/Ensembler/examples*.

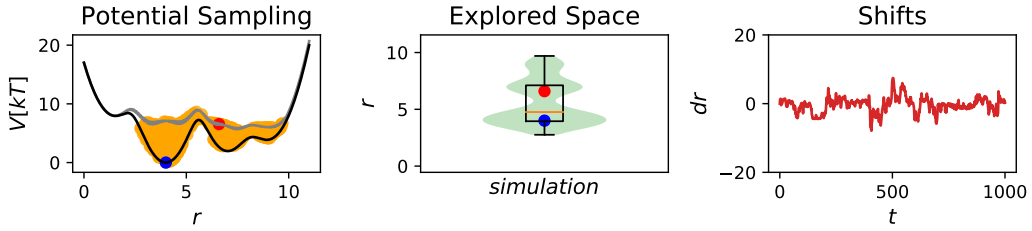
Code Example

In typical applications of Ensembler, the user selects a potential-energy function from the available ones, e.g. a potential-energy function with four wells, and initializes it with chosen parameters. To sample this system with stochastic dynamics (SD),[12] the sampling method is instantiated and passed to the *system class*. The simulation is performed by calling the function *simulate* with the desired number of steps passed as parameter. Subsequently, the results can be analysed using the built-in visualization functions that are compatible with the *simulation class* of Ensembler. As can be seen in Figure 2a, the energy barriers between the different minima were not crossed during the chosen simulation length. To overcome the sampling issue, enhanced sampling techniques can be employed,[26] e.g. local elevation[15]/metadynamics[16] (Figure 2b). Thereby, a time-dependent biasing potential is generated, i.e. a Gaussian biasing potential is added to positions that were already visited such that they become energetically less favorable. This decreases the likelihood of visiting

(a) Standard Langevin Simulation



(b) Langevin Simulation with Local Elevation/Metadynamics



(c) Example Source Code

```
#Local elevation/metadynamics simulation:
##Imports
from ensembler.potentials.OneD import fourWellPotential, metadynamicsPotential
from ensembler.samplers.stochastic import langevinIntegrator
from ensembler.system import system
from ensembler.visualisation.plotSimulations import simulation_analysis_plot

##Simulation Setup
origpot = fourWellPotential(Vmax=4, a=1.5, b=4.0, c=7.0, d=9.0,  ah=2., bh=0., ch=0.5, dh=1.)
### Difference between Langevin and local elevation simulation
V = metadynamicsPotential(origpot, amplitude=0.35, sigma=0.5, n_trigger=15)
sampler = langevinIntegrator(dt=0.1, gamma=10, old_position=3)

system2=system(potential=V, sampler=sampler, start_position=4, temperature=1)

##Simulate
cur_state = system2.simulate(steps=1000,withdraw_traj=True, init_system=True)

##Visualize
positions = np.linspace(start=0, stop=10, num=1000) #phase space to be visualized
fig, out_path = simulation_analysis_plot(system=system2, title="Local Elevation/Metadynamics Simulation",
                                         limits_coordinate_space=positions)
```

Figure 2: Langevin simulation of a four-well potential energy-function: Results when sampling (1000 steps) with the standard SD integrator (a) or with local elevation[15]/metadynamics[16] (b). The left panel shows the potential-energy surface (black), the sampled range (orange), as well as the start point (blue) and end point (red). The middle panel shows the sampled space as a violin/box plot with the start point (blue) and end point (red). The right panel shows the shift $\Delta r_t = r_{t+1}$ - r_t as a function of simulation time t. (c) Source code to perform the simulations. First, the four-well *potential class* and the Langevin *sampler class* are initiated. Next, they are wrapped by a *system class*, which executes the simulation. Note that only one line has to be added to use the enhanced sampling technique (marked in bold). Visualizations are generated with built-in functions.

known positions again. The enhanced sampling technique can be applied by adding a single line of code compared to the previous simulation (Figure 2c).

Application Example: Free-Energy Calculation

Free-energy calculation is an important field in computational chemistry, e.g. to estimate protein-ligand binding or polymer formation.[22,43–45] The calculation of alchemical free-energy differences, ΔF, with Ensembler is exemplified in the tutorial Jupyter notebook on GitHub with a mutation of the equilibrium position of a one-dimensional harmonic oscillator. This mutation corresponds to a change of a covalent bond type at the terminus of a linear molecule. This system is very simple such that the result can be calculated analytically as reference (Table 1). In the following, the sampling of the two end states of the model system and the results with different methods are shortly discussed.

A simple approach is to simulate one end state and estimate ΔF with the Zwanzig equation.[17] The quality of the result depends on the phase-space overlap between the two end states.[46] Alternatively, one can simulate both end states separately and use BAR,[18] yielding more converged results.[46] If the phase-space overlap between the two end states is not sufficient, more advanced sampling methods are necessary to obtain converged results. One possibility is to generate intermediate states as a linear combination of the two end states A and B with the coupling parameter λ, i.e. $H(\lambda) = (1-\lambda)H_A + \lambda H_B$, such that $H(\lambda = 0) = H_A$ and $H(\lambda = 1) = H_B$. The intermediate states are positioned at discrete λ-points between 0 and 1.[47,48] ΔF can be estimated using FEP[17] or BAR[18] as the path over all intermediates, or with TI[19] as the integral along λ.

Another elegant free-energy method is EDS,[20,21] where a reference-state Hamiltonian H_r is sampled, which guarantees phase-space overlap of the reference state with all end states,

$$H_R = -\frac{1}{\beta s}\ln(e^{(-\beta s(H_A - E_A^R)} + e^{(-\beta s(H_B - E_B^R)}), \tag{1}$$

where $1/\beta = k_B T$, k_B being the Boltzmann constant and T the absolute temperature. H_R can be optimized for sampling using two kinds of parameters: The smoothing parameter s lowers the energy barriers between the end states, whereas the energy offsets E^R ensure equal weighting of all end states. In our example, both end states are sampled sufficiently during the EDS simulation with $s = 0.3$ and the energy offsets $E^R = [0, 0]$. Subsequently, the Zwanzig equation[17] is used to obtain ΔF between the end states.[20,21] Recently, a hybrid form of EDS and λ-coupling was introduced, termed λ-EDS.[23] At $\lambda = 0$ or 1, H_R is equal to the Hamiltonians of the respective end states, while conventional EDS is recovered with λ=0.5 (except for an offset).[23] λ-EDS allows for a λ-weighting of the exponential terms in the EDS equation.

All free-energy calculations discussed above were performed with Ensembler for a total of 10'000 Monte Carlo (MC)[10] steps, and each simulation was repeated five times. The simulation results listed in Table 1 show that larger errors are obtained without intermediate states due to insufficient phase-space overlap. Using ten λintermediate states together with TI gave the best result, however, this approach is also the computationally most expensive one (i.e. ten separate simulations). s EDS and λ-EDS, on the other hand, yielded also good results, while requiring only one simulation (given a set of suitable reference-state parameters). We refer to the Jupyter notebook in the Ensembler GitHub repository for the source code, more detailed information on these methods as well as additional methods like conveyor-belt TI[25] and RE-EDS,[24,49] which combine enhanced sampling and free-energy methods.

Conclusion

In this work, we introduced Ensembler as a tool to support teaching of MD simulations and free-energy techniques, and to enable rapid prototyping of new methods using 1D or 2D model systems. The package provides a large set of implemented methods for comparison.

Table 1: Estimated ΔF for the model system. Sampling was performed with Monte Carlo (MC)[10] for 10'000 steps in each simulation. Each calculation was replicated five times and the averaged result is shown together with the standard deviation. The duration of the computations (without visualizations) was estimated directly in the Jupyter notebook and is given relative to the FEP simulation (absolute duration = 2.0 seconds). The performance was tested on a Lenovo Thinkpad T420s with an Intel i5-2520 (2.5 GHz) CPU and 8 GB RAM. The RAM usage for the full Jupyter notebook execution was in total 578 MB.

Method	Average ΔF $[k_BT]$	Deviation from analytical result $[k_BT]$	Speed (rel. to FEP) Simulation	Analysis
analytical	1.275	-	-	-
FEP[17]	6.579 ± 1.009	5.305	1.0	0.1
BAR[18]	2.437 ± 0.500	2.437	3.0	2.1
FEP 10-λ-points	1.406 ± 0.431	0.131	14.0	0.7
TI[19] 10-λ-points	1.242 ± 0.015	0.033	14.0	0.04
EDS[20–22]	0.958 ± 0.110	0.317	2.4	0.2
λ-EDS[23] $\lambda = 0.5$	0.987 ± 0.111	0.287	3.1	0.2

The open-source basis, the lean structure, and the simplicity of Python3 form a convenient and efficient framework. The code examples and application example for free-energy calculation highlight the ease of using Ensembler. With this, Ensembler can contribute to improving the shareability, comparability, and reproducibility for method development in our field.

Acknowledgement

The authors gratefully acknowledge financial support by the Swiss National Science Foundation (Grant Number 200021-178762), the Scholarship Fund of the Swiss Chemical Industry, and by ETH Zurich (ETH-34 17-2). S.M.L was also supported by the PhD scholarship of the German National Academic Foundation.

References

(1) Peng, R. D. Reproducible Research in Computational Science. *Science* **2011**, *334*, 1226–1228.

(2) Van Rossum, G.; Drake, F. L. Python 3 Reference Manual. 2009.

(3) Stroustrup, B. *The C++ Programming Language*, 4th ed.; Addison-Wesley, 2013.

(4) Stodden, V.; McNutt, M.; Bailey, D. H.; Deelman, E.; Gil, Y.; Hanson, B.; Heroux, M. A.; Ioannidis, J. P.; Taufer, M. Enhancing Reproducibility for Computational Methods. *Science* **2016**, *354*, 1240–1241.

(5) Chacon, S.; Straub, B. *Pro git*; Springer Nature, 2014.

(6) Naden, L. N.; Smith, D. G. A. Cookiecutter for Computational Molecular Sciences (CMS) Python Packages. 2018.

(7) Jupyter, P.; Bussonnier, M.; Forde, J.; Freeman, J.; Granger, B.; Head, T.; Kelley, K.; Nalvarte, G.; Osheroff, A.; Pacer, M.; Panda, Y.; Perez, F.; Ragan-kelley, B.; Willing, C. Binder 2.0 – Reproducible, Interactive, Sharable Environments for Science at Scale. *Proc. of the 17th python in Science Conf.* **2018**, 113–120.

(8) Jones, J. E. On the Determination of Molecular Fields. I. From the Variation of the Viscosity of a Gas with Temperature. *Proc. Royal Soc. London* **1924**, *106*, 441–462.

(9) Hestenes, M. R.; Stiefel, E. Methods of Conjugate Gradients for Solving Linear Systems. *J. Res. Natl. Bur. Stand.* **1952**, *49*, 409–436.

(10) Hastings, W. K. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika* **1970**, *57*, 97–109.

(11) van Gunsteren, W. F.; Berendsen, H. J. A Leap-Frog Algorithm for Stochastic Dynamics. *Mol. Sim.* **1988**, *1*, 173–185.

(12) Brünger, A.; Brooks, C. L.; Karplus, M. Stochastic Boundary Conditions for Molecular Dynamics Simulations of ST2 Water. *Chem. Phys. Lett.* **1984**, *105*, 495–500.

(13) Torrie, G. M.; Valleau, J. P. Nonphysical Sampling Distributions in Monte Carlo Free-Energy Estimation: Umbrella Sampling. *J. Comput. Phys.* **1977**, *23*, 187–199.

(14) Sugita, Y.; Okamoto, Y. Replica-Exchange Molecular Dynamics Method for Protein Folding. *Chem. Phys. Lett.* **1999**, *314*, 141–151.

(15) Huber, T.; Torda, A. E.; van Gunsteren, W. F. Local Elevation: A Method for Improving the Searching Properties of Molecular Dynamics Simulation. *J. Comput. Aided Mol. Des.* **1994**, *8*, 695–708.

(16) Laio, A.; Parrinello, M. Escaping Free-Energy Minima. *Proceed. Natl. Acad. Sci. U.S.A.* **2002**, *20*, 12562–12566.

(17) Zwanzig, R. W. High-Temperature Equation of State by a Pertubation Method. I. Nonpolar Gases. *J. Chem. Phys.* **1954**, *22*, 1420–1426.

(18) Bennett, C. H. Efficient Estimation of Free Energy Differences From Monte Carlo Data. *J. Comput. Phys.* **1976**, *22*, 245–268.

(19) Kirkwood, J. G. Statistical Mechanics of Fluid Mixtures. *J. Chem. Phys.* **1935**, *3*, 300–313.

(20) Christ, C. D.; van Gunsteren, W. F. Enveloping Distribution Sampling: A Method to Calculate Free Energy Differences From a Single Simulation. *J. Chem. Phys.* **2007**, *126*, 184110.

(21) Christ, C. D.; van Gunsteren, W. F. Multiple Free Energies From a Single Simulation: Extending Enveloping Distribution Sampling to Nonoverlapping Phase-space Distributions. *J. Chem. Phys.* **2008**, *128*, 174112.

(22) Christ, C. D.; Mark, A. E.; van Gunsteren, W. F. Basic Ingredients of Free Energy Calculations: A Review. *J. Comput. Chem.* **2009**, *31*, 1569–1582.

(23) König, G.; Glaser, N.; Schroeder, B.; Hünenberger, P. H.; Riniker, S. An Alternative to Conventional λ-Intermediate States in Alchemical Free Energy Calculations: λ-Enveloping Distribution Sampling. *J. Chem. Inf. Model.* **2020**, *60*, 5407–5423.

(24) Sidler, D.; Schwaninger, A.; Riniker, S. Replica Exchange Enveloping Distribution Sampling (RE-EDS): A Robust Method to Estimate Multiple Free-Energy Differences From a Single Simulation. *J. Chem. Phys.* **2016**, *145*, 154114.

(25) Hahn, D. F.; Hünenberger, P. H. Alchemical Free-Energy Calculations by Multiple-Replica -Dynamics: The Conveyor Belt Thermodynamic Integration Scheme. *J. Chem. Theory Comput.* **2019**, *15*, 2392–2419.

(26) Pohorille, A.; Jarzynski, C.; Chipot, C. Good Practices in Free-Energy Calculations. *J. Phys. Chem. B* **2010**, *114*, 10235–10253.

(27) Kluyver, T.; Ragan-kelley, B.; Pérez, F.; Granger, B.; Bussonnier, M.; Frederic, J.; Kelley, K.; Hamrick, J.; Grout, J.; Corlay, S.; Ivanov, P.; Avila, D.; Abdalla, S.; Willing, C.; Development Team, J. Jupyter Notebooks - A Publishing Format for Reproducible Computational Workflows. *ELPUB* **2016**, 87–90.

(28) Meurer, A.; Smith, C. P.; Paprocki, M.; Čertík, O.; Kirpichev, S. B.; Rocklin, M.; Kumar, A.; Ivanov, S.; Moore, J. K.; Singh, S.; Rathnayake, T.; Vig, S.; Granger, B. E.; Muller, R. P.; Bonazzi, F.; Gupta, H.; Vats, S.; Johansson, F.; Pedregosa, F.; Curry, M. J.; Terrel, A. R.; Roučka, Š.; Saboo, A.; Fernando, I.; Kulal, S.; Cimrman, R.; Scopatz, A. SymPy: Symbolic Computing in Python. *PeerJ Comput. Sci.* **2017**, *3*, e103.

(29) Cheatham, T. E.; Miller, J. L.; Fox, T.; Darden, T. A.; Kollman, P. A. Molecular Dynamics Simulations on Solvated Biomolecular Systems: The Particle Mesh Ewald Method Leads to Stable Trajectories of DNA, RNA, and Proteins. *J. Am. Chem. Soc.* **1995**, *117*, 4193–4194.

(30) Leach, A. R. *Molecular Modelling – Principles and Applications*; Pearson Education Limited, 2001.

(31) Sugita, Y.; Kitao, A.; Okamoto, Y. Multidimensional Replica-Exchange Method for Free-Energy Calculations. *J. Chem. Phys.* **2000**, *113*, 6042–6051.

(32) Yamauchi, M.; Okumura, H. Development of Isothermal-Isobaric Replica-Permutation Method for Molecular Dynamics and Monte Carlo Simulations and Its Application to Reveal Temperature and Pressure Dependence of Folded, Misfolded, and Unfolded States of Chignolin. *J. Chem. Phys.* **2017**, *147*, 184107.

(33) Github, GitHub. 2020; `https://github.com/`.

(34) Krekel, H.; Oliveira, B.; Pfannschmidt, R.; Bruynooghe, F.; Laugher, B.; Bruhin, F.; Al, E. Pytest: Helps You Write Better Programs. 2004.

(35) Brandl, G. Sphinx Documentation Tool. 2008.

(36) Github, Github Actions. 2007; `https://docs.github.com/en/actions/reference`.

(37) Virtanen, P. et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **2020**, *17*, 261–272.

(38) Van Der Walt, S.; Colbert, S. C.; Varoquaux, G. The NumPy Array: A Structure for Efficient Numerical Computation. *Comput. Sci. Eng.* **2011**, *13*, 22–30.

(39) McKinney, W. Data Structures for Statistical Computing in Python. *Proc. of the 9th Python in Science Conf.* **2010**, *445*, 51–56.

(40) Hunter, J. D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng.* **2007**, *9*, 99–104.

(41) Andersen, H. C. Molecular Dynamics Simulations at Constant Pressure and/or Temperature. *J. Chem. Phys.* **1980**, *72*, 2384–2393.

(42) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; Duchesnay, E. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.

(43) Hansen, N.; van Gunsteren, W. F. Practical Aspects of Free-Energy Calculations: A Review. *J. Chem. Theory Comput.* **2014**, *10*, 2632–2647.

(44) Cournia, Z.; Allen, B. K.; Beuming, T.; Pearlman, D. A.; Radak, B. K.; Sherman, W. Rigorous Free Energy Simulations in Virtual Screening. *J. Chem. Inf. Model.* **2020**,

(45) Armacost, K. A.; Riniker, S.; Cournia, Z. Novel Directions in Free Energy Methods and Applications. *J. Chem. Inf. Model.* **2020**, *60*, 1–5.

(46) König, G.; Brooks, B. R.; Thiel, W.; York, D. M. On the Convergence of Multi-scale Free Energy Simulations. *Mol. Sim.* **2018**, *44*, 1062–1081.

(47) Valleau, J. P.; Card, D. N. Monte Carlo Estimation of the Free Energy by Multistage Sampling. *J. Chem. Phys.* **1972**, *57*, 5457–5462.

(48) Straatsma, T. P.; McCammon, J. A. Multiconfiguration Thermodynamic Integration. *J. Chem. Phys.* **1991**, *95*, 1175–1188.

(49) Sidler, D.; Cristòfol-Clough, M.; Riniker, S. Efficient Round-Trip Time Optimization for Replica-Exchange Enveloping Distribution Sampling (RE-EDS). *J. Chem. Theory Comput.* **2017**, *13*, 3020–3030.

Graphical TOC Entry