tidynamics: A tiny package to compute the dynamics of stochastic and molecular simulations

Pierre de Buyl[1]

1 Instituut voor Theoretische Fysica, KU Leuven, B-3001 Leuven, Belgium

DOI: 10.21105/joss.00877

Software

- Review
- Repository
- Archive

Submitted: 15 July 2018
Published: 27 August 2018

Summary

tidynamics provides an efficient implementation of the Fast Correlation Algorithm (FCA) to compute correlation functions of interest in molecular (Allen & Tildesley, 1987; G. R. Kneller, Keiner, Kneller, & Schiller, 1995) and stochastic (Gardiner, 2004) dynamics. Building on the FCA, tidynamics computes the autocorrelation (the correlation of a time series with itself), the correlation between two time-series, the mean-square displacement of a trajectory, and the cross-displacement (for off-diagonal realisations of Brownian motion). These correlation functions serve for the quantitative measure of relaxation and transport coefficients from numerical trajectories.

There is a lack of a self-contained implementation of the algorithm. The software nMoldyn (Hinsen, Pellegrini, Stachura, & Kneller, 2012) implements it within a larger library but it has a more complex interface and more dependencies. The goal of tidynamics is to serve as a reference implementation with a lighter interface.

tidynamics is designed as a library in which every routine operates directly on NumPy arrays and returns NumPy arrays. The interface is simple and enables convenient use in interactive sessions or in teaching material. We test the library against Python 2.7, 3.5, and 3.6 and several versions of NumPy to guard against a possible sensitivity to the API.

The Fast Correlation Algorithm relies on the Fourier transform to compute correlations. For this purpose, we use NumPy's (T. E. Oliphant, 2007) FFT module `np.fft`. The advantage of using Fourier transforms is a reduced computational cost in comparison to a direct loop over the data. We expect a scaling of the CPU time t_{CPU} of $t_{\mathrm{CPU}} \propto N \log N$ where N is the length of the time series. We show in Figure 3 the actual CPU time and compare it favourably to the expected scaling in the example *Scaling behaviour*.

NumPy (T. E. Oliphant, 2007) and SciPy (Jones, Oliphant, Peterson, & others, 2001) implement correlation routines as well. In the case of NumPy, the computation is based on a direct loop with a quadratic scaling of the CPU time $O(N^2)$. In SciPy, both the direct and a Fourier transform version are implemented.

The definition of `np.correlate` and `scipy.signal.correlate` differs from the definition traditionally used in dynamical systems in two ways. These routines do not correct for aperiodic signals, an issue that is addressed in the Fast Correlation Algorithm by zero-padding the signal, and they do not normalise the result by the actual number of samples. In addition to these differences, the relative complexity of building SciPy and its larger size motivated us to rely on NumPy only. NumPy and SciPy do not provide functions to compute the mean-square displacement of trajectories. The Fast Correlation Algorithm as applied to mean-square displacements was proposed by G. R. Kneller et al. (1995) and is less known than the plain correlation algorithm.

The benefits of using tidynamics originate in the implementation of the suitable definitions for the study of dynamical systems, good performance, and its ease of installation.

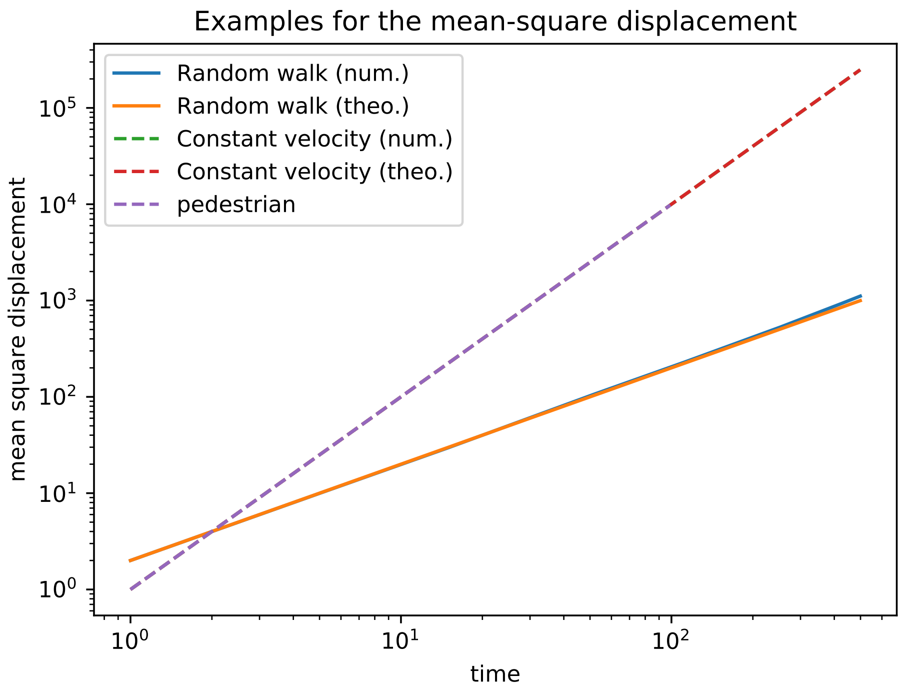

Figure 1: Mean-square displacement for a random walk and for a trajectory at constant velocity.

Examples

We show in Figures 1, 2, and 3 the examples provided with tidynamics. The examples in the repository include short codes for a random walk, for the Ornstein-Uhlenbeck process, and for the scaling performance analysis of the autocorrelation routine.

Acknowledgements

PdB is a postdoctoral fellow of the Research Foundation Flanders - FWO. PdB would like to acknowledge useful discussions with Konrad Hinsen and feedback about the library from Max Linke.

References

Allen, M. P., & Tildesley, D. J. (1987). *Computer simulation of liquids*. Oxford: Clarendon Press.

Gardiner, C. W. (2004). *Handbook of stochastic methods for physics, chemistry, and the natural sciences*. Springer series in synergetics (3rd ed.). Springer.

Hinsen, K., Pellegrini, E., Stachura, S., & Kneller, G. R. (2012). NMoldyn 3: Using task farming for a parallel spectroscopy-oriented analysis of molecular dynamics simulations. *J. Comput. Chem.*, *33*(25), 2043–2048. doi:10.1002/jcc.23035

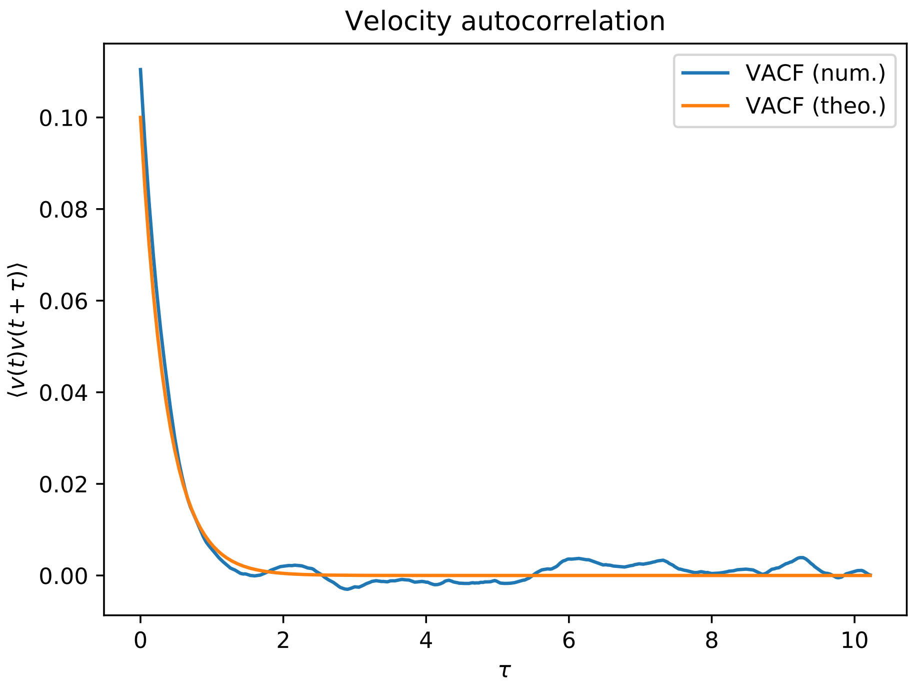

Figure 2: Velocity autocorrelation function for a Ornstein-Uhlenbeck process (Gardiner, 2004).

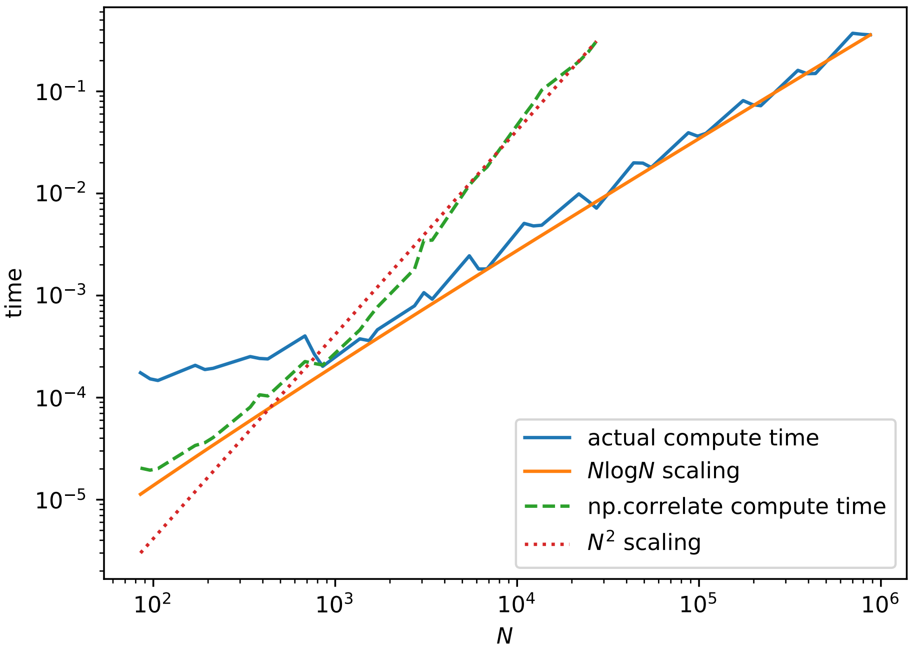

Figure 3: Scaling of the CPU time with respect to the length of the data.

Jones, E., Oliphant, T., Peterson, P., & others. (2001). SciPy: Open source scientific tools for Python. Retrieved from http://www.scipy.org/

Kneller, G. R., Keiner, V., Kneller, M., & Schiller, M. (1995). NMOLDYN: A program package for a neutron scattering oriented analysis of molecular dynamics simulations. *Comp. Phys. Commun.*, *91*(1), 191–214. doi:10.1016/0010-4655(95)00048-K

Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, *9*, 10–20. doi:10.1109/MCSE.2007.58