Pychastic: Precise Brownian dynamics using Taylor-Itō integrators in Python

Radost Waszkiewicz⋆ , Maciej Bartczak , Kamil Kolasa and Maciej Lisicki

Institute of Theoretical Physics, Faculty of Physics, University of Warsaw
L. Pasteura 5, 02-093 Warsaw, Poland

⋆ radost.waszkiewicz@fuw.edu.pl

Abstract

In the last decade, Python-powered physics simulations ecosystem has been growing steadily, allowing greater interoperability, and becoming an important tool in numerical exploration of physical phenomena, particularly in soft matter systems. Driven by the need for fast and precise numerical integration in colloidal dynamics, here we formulate the problem of Brownian Dynamics (BD) in a mathematically consistent formalism of the Itō calculus, and develop a Python package to assist numerical computations. We show that, thanks to the automatic differentiation packages, the classical truncated Taylor-Itō integrators can be implemented without the burden of computing the derivatives of the coefficient functions beforehand. Furthermore, we show how to circumvent the difficulties of BD simulations such as calculations of the divergence of the mobility tensor in the diffusion equation and discontinuous trajectories encountered when working with dynamics on S^2 and $SO(3)$. The resulting Python package, Pychastic, is capable of performing BD simulations including hydrodynamic interactions at speeds comparable to dedicated implementations in lower-level programming languages, but with a much simpler end-user interface.

This publication is part of a bundle: Please cite both the article and the release you used.

Contents

1 Introduction

The dynamics of soft and colloidal matter systems is of importance for numerous technological and industrial processes, such as food products, pastes, creams, and gels [1]. Another important example are biological systems which involve aqueous suspensions of colloids, macromolecules, polymers, and cells. The diversity of the constituent elements, together with the tunability of their direct interactions, and the presence of hydrodynamic interactions (HI) mediated by the suspending fluid [2], gives rise to a multitude of complex dynamic phenomena, which can only be explored using appropriate numerical techniques [3, 4].

The choice of a suitable method derives from the characteristic time scales of the investigated dynamics [5]. In the context of macromolecules, when short-time effects are of interest, Molecular Dynamics (MD) simulations are a popular choice [6]. Short times may be comparable to the solvent relaxation time scale, which for a fluid of kinematic viscosity ν and speed of sound c scales as ν/c^2. For water, this time is of the order of 10^{-13} s. The idea behind MD simulations is to solve Newtonian equations of motion for atoms or molecules, which are a set of second-order ODEs with effective interaction potentials. In this case, HI can be resolved using either explicit solvent methods that resolve the molecular structure of the solvent or more approximate implicit solvent models. However, the typical time scales of colloidal motion are much longer; the velocity relaxation time for colloidal particles of size a and density comparable to that of the fluid is approximately a^2/ν, amounting to ca. 10^{-8} s for a 100 nm colloid. On coarser time scales, the colloidal velocity relaxes multiple times, and the Rayleigh description in terms of velocity becomes irrelevant [5, 7]. Instead, the description in terms of the position of a Brownian particle becomes possible on diffusive time scales, during which a particle diffuses over a distance of its own radius. This time scale is $a^2/D \sim \eta a^3/k_B T$, where η is the dynamic viscosity of the fluid, k_B is the Boltzmann constant, and T is the temperature. For the aforementioned particle, this time scale is ca. 10^{-3} s, and corresponds to the minimal time resolution of typical light scattering and microscopy experiments. Thus, the description of the dynamics on such coarse scales is made solely using the position of the particle. The clear separation of discussed time scales, combined with the numerical stiffness of Newton's equations, renders MD simulations not applicable to explore diffusive dynamics.

Instead, BD simulations present a convenient alternative, building on models suitable for

diffusive time scales, and involving accordingly long time steps for numerical computations. Since the velocity of particles varies very rapidly on such time scales, the position becomes a random variable whose properties are dictated by the fluctuation-dissipation theorem [7].

The development of a suitable theoretical description to explain the Brownian motion and its subsequent experimental verification at the beginning of the 20th century were a ground-breaking step that confirmed the atomic structure of matter [8]. Early works by pioneers such as Einstein [9], Smoluchowski [10], Fokker [11], and Planck [12], led to the description of diffusive processes through the underlying probability density functions (PDFs) for the positions of the particles and their temporal evolution, rather than representations of individual trajectories. The realisation that the path of a Brownian particle resembles a nowhere differentiable function paved the route for a different approach, now called stochastic calculus, with the classical works of Kolmogorov [13], Wiener, and finally Itō [14]. With these tools, the problem of Brownian motion can be recast in the form of stochastic differential equations (SDEs), which offer a practical route for numerical simulations.

In this contribution, we present our approach to the implementation of such Brownian dynamics. Our goal was to show how modern programming techniques available in Python: reflective and functional programming paradigms allow one to develop concise implementations of physically relevant problems. These constitute relatively recent developments in computer modelling, and we follow this trend to facilitate faster modelling of diffusive phenomena.

First, in Sec. 2 we review three popular ways of describing the Brownian motion using the Langevin equation, the Fokker-Planck equation, and Itō equation. We then provide an overview of available numerical integration packages for SDEs in Sec. 3.1. Next, we present our new package, Pychastic, in Sec. 3, along with three examples (in Sec. 4) of problems in which the packages greatly facilitate numerical computations. We conclude the work in Sec. 5.

2 Three vantage points: Langevin, Fokker-Planck, and Itō

Perhaps the most popular approach to modelling the trajectories of Brownian particles dates back to Langevin's work of 1908 [15]. The Langevin equation can be rationalised as an extension of Newtonian mechanics to include the effects of fluctuations by adding a stochastic force, F_n, acting on a particle. We write it below for a particle with a constant friction coefficient ζ. The average value of the force is zero, while its covariance at temperature T is given by the fluctuation-dissipation theorem [7], $\langle F_n(t)F_n(t')\rangle = 2k_B T\zeta\delta(t-t')$, with k_B being the Boltzmann constant. The equation of motion of the particle in the presence of a deterministic force F reads then

$$m\ddot{x} = -\zeta\dot{x} + F(x,t) + F_n(t). \tag{1}$$

For the case of a spherical particle of radius a in Stokes flow the friction coefficient is $6\pi\eta a$, η being the dynamic viscosity. We note that even though $x(t)$ is not differentiable once, and certainly not differentiable twice at any point even in the usual distributional sense (because the Lebesgue integral requires finite variation and realisations of the Wiener process have infinite variation), Eq. (1) can be given a proper interpretation by transforming it into an integral form. However, the above equation proved practical for numerical calculations, e.g. using an integration scheme analogous to the Euler-Maruyama method.

An alternative description via the Fokker-Planck equation circumvents the difficulties of interpreting the dynamics on a single trajectory in the Langevin equation by focussing on a probabilistic description. On the Brownian time scale, the position of a particle is now treated as a random variable, and the probability density function (PDF) $P(x,t)$ of finding the particle

at a location x at a time t evolves according to the partial differential equation

$$\frac{\partial P(x,t)}{\partial t} = \zeta^{-1}\frac{\partial}{\partial x}\left(F(x,t)P(x,t)\right) + D\frac{\partial^2 P(x,t)}{\partial x^2}, \tag{2}$$

with the diffusion coefficient defined by $D = k_B T/\zeta$. The problem is now well-posed mathematically, and the tools of mathematical analysis of PDEs can be employed to study the evolution of the underlying probability distribution, e.g. by examining its moments. However, questions involving individual trajectories become less straightforward: problems involving first passage time have to be dealt with with a careful treatment of the boundary conditions (e.g. substances vanishing in chemical reactions). Moreover, from the point of view of numerical simulations, this approach becomes impractical for large systems. Using finite-difference methods, for N particles in a 3D simulation box of size L and mesh size Δx, we typically require $(L/\Delta x)^{3N}$ points to track the PDF, which may become prohibitively large.

The mathematical difficulties of the Langevin equation are absent in the proper treatment of SDEs in the Itō formalism. If now dX denotes the position increment of a particle in a time interval dt, we can write it as

$$dX = Fdt + \sqrt{2D}dW, \tag{3}$$

where Fdt denotes the systematic drift of the particle and $\sqrt{2D}dW$ denotes the stochastic (diffusive) component of motion (provided that D is constant and the metric tensor is constant as well). This equation is meant in the distributional sense with respect to the Itō integral, that is

$$X(T) = \int_0^T dX = \int_0^T F(X,t)dt + \int_0^T \sqrt{2D}dW. \tag{4}$$

Here, all integrals are taken in the Itō sense. Such a trajectory-focused formulation effectively deals with the ill-defined derivatives in the Langevin equation. Using Itō's lemma imposes formal rules of transformation of the coefficients, freeing us from a canonical coordinate description. Finally, the estimation of observable quantities such as expected values, correlation functions or equilibrium distributions of low-dimensional projections of evolving variables can be recovered using a Monte Carlo approach, which for M simulations converges as $\sqrt{M}$, regardless of dimensionality. The constant of proportionality of this convergence is controlled only by the variance of the variable of interest, which is often independent of the dimensionality of the equation. For example, in the problem of diffusion of a single particle in a semidilute suspension, the variance of the mean squared displacement of the tracer is weakly dependent on the number of particles in the simulation volume.

3 Pychastic: description of the package

3.1 Available numerical integration packages

Only a few SDE integration packages have been made available in recent years. Two notable examples are `DifferentialEquations.jl` for Julia and `ItoProcess` being a part of Mathematica. Now, we present `Pychastic` for Python, which takes advantage of the popularity of this language. Our package source code is available on Github, up-to-date documentation on ReadTheDocs and ready to install via `pip` via Python Package Index. As a preliminary comparison, we note that `DifferentialEquations.jl` has the largest variety of integrators (for example, many options for stiff equations), while Mathematica's `ItoProcess` and `Pychastic` contain essentially the same algorithms. However, the biggest drawback of `ItoProcess` is the lack of step post-processing. When working with SDEs defined on manifolds whose universal cover is not $\mathbb{R}^n$, such as a sphere S^2 or the space of rigid rotations $SO(3)$, any parameterisation

Table 1: Available SDE integration packages.

Package	DifferentialEquations.jl	ItoProcess	Pychastic
Language	Julia	Mathematica	Python
License	MIT	proprietary	MIT
Codebase	open	closed	open
scalar SDEs	yes	yes	yes
vector SDEs	yes	yes	yes
strong convergence	up to order 1.5	up to order 1.5	up to order 1.5
weak convergence	up to order 2.0	up to order 2.0	up to order 2.0
supports events	yes	no	yes

of $\mathbb{R}^n$ will contain singularities, thus an integrator which cannot handle discontinuous paths cannot reproduce, e.g. 3-dimensional rotational dynamics [16]. Table 1 contains a synthetic comparison of the three mentioned packages.

3.2 Implementation details

Pychastic contains implementations of three numerical integration schemes based on the Taylor-Itō expansion. They are the schemes of strong order 1/2, 1, and 3/2 which in the package are referred to as Euler, Milstein, and Wagner-Platen schemes, respectively.

The basis for these integration schemes is the Taylor-Itō expansion [18] which generalises the deterministic Taylor expansion to SDEs. We write it here for a one-dimensional problem driven by a Wiener process W

$$dX = a(X)dt + b(X)dW. \tag{5}$$

Following the notation of Kloeden & Platen [18], the Taylor-Itō expansion up to strong order 3/2 for a scalar function X has the form

$$
\begin{aligned}
X(T) = & X(0) + a\Delta + b\Delta W \\
& + \frac{1}{2}bb'\big((\Delta W)^2 - \Delta\big) \\
& + a'b\Delta Z + \frac{1}{2}\Big(aa' + \frac{1}{2}b^2 a''\Big)\Delta^2 \\
& + (ab' + \frac{1}{2}b^2 b'')(\Delta W\Delta - \Delta Z) + \frac{1}{2}b\big(bb'' + (b')^2\big)\Big(\frac{1}{3}(\Delta W)^2 - \Delta\Big)\Delta W,
\end{aligned}
\tag{6}
$$

where

$$\Delta = \int_0^T dt = T, \tag{7}$$

$$\Delta W = \int_0^T dW = W(T) - W(0), \tag{8}$$

$$\Delta Z = \int_0^T \int_0^s dW(s)\,ds. \tag{9}$$

The stochastic Euler scheme is based on the first line of this expansion, with three terms only. The Milstein scheme includes the next term, namely the second line of Eq. (6). Finally, the Wagner-Platen scheme includes all the terms mentioned above. We note from this expansion

that even the simplest schemes require correct computation of coefficients multiplying principal Wiener integrals (Δ, ΔW, ΔZ, and others in the multidimensional case) and sampling from correct distributions corresponding to these integrals. Importantly, an analogous expansion for vector quantities is considerably more complex, and other Wiener integrals arise, as detailed in [18]. For brevity, we do not write this expansion here explicitly, but we have implemented the vector Taylor-Itō expansion in Pychastic to enable simulations of both scalar and vector processes.

Although expressions for these coefficient functions can be written explicitly in principle, in Pychastic we take advantage of functional programming tools, which results in a greatly simplified implementation. First, by introducing the $\mathcal{L}^0$ and $\mathcal{L}^j$ operators (again using the Kloeden & Platen notation), we can express all coefficient functions by repeated application of the $\mathcal{L}$ operators to the a and b functions. Using the jax.grad functionality, this is implemented directly as

```python
def tensordot1(a, b):
    return jax.numpy.tensordot(a, b, axes=1)

def tensordot2(a, b):
    return jax.numpy.tensordot(a, b, axes=2)

# Taylor-Ito expansion operators
def L_t_operator(f,problem):
    @wraps(f)
    def wrapped(x):
        b_val = problem.b(x)
        val = tensordot1(jax.jacobian(f)(x), problem.a(x)) + 0.5 * tensordot2(
            jax.hessian(f)(x), tensordot1(b_val, b_val.T)
        )
        return val[:,jnp.newaxis,...] #indexing convention [spatial, time, ...
= noiseterms/time]

    return wrapped

def L_w_operator(f,problem):
    @wraps(f)
    def wrapped(x):
        val =  tensordot1(jax.jacobian(f)(x), problem.b(x))[:,jnp.newaxis,...]
        return jnp.swapaxes(val,1,-1)[:,...,0] # indexing convention [spatial,
noiseterms, ... = noiseterms/time]

    return wrapped
```

Second, different integration methods – Euler, Milstein, and Wagner-Platen – were implemented following the book by Kloeden & Platen [18]. These methods make use of samples of the principal Wiener integrals listed above. Unfortunately, the text contains typographic errors, which were found by examining the mean, variance and covariance properties of fundamental multiple Itō and Stratonovich integrals using a testing suite that is part of the Pychastic package. The errors we found were corrected appropriately and are listed in the Appendix A.

Finally, using the jax.lax.scan functionality of the package jax, we can generate many trajectories in parallel by programmatically vectorising the code representing a and b provided by the user.

4 Examples of usage

In the following, we show a few examples of usage that take advantage of various functionalities of Pychastic.

4.1 First passage problems, polar random walk

In nearly all physical applications of Brownian Dynamics, the simulated properties of the systems are observed via their moments rather than individual realisations of the underlying stochastic process. In fact, in many situations with random forcing, the paths of the process are only a model of reality and cannot be directly observed, as opposed to their statistical effect. For a smooth quantity g observed through its expected value $\mathbb{E}$ in thermodynamic equilibrium, it is the weak convergence rate that controls the error of $\mathbb{E}(g(X_t))$ at fixed t. With this in mind, it is tempting to dismiss methods with a high strong order of convergence as impractical.

A natural candidate for a counterexample are first passage times, where the answer requires a more subtle reasoning. Since the first passage problem is *path-dependent*, it would seem that strong convergence is important. However, the theorem regarding the first passage times established by Whitt [17] shows that the weak convergence of the process approximations $X_n \rightarrow X$ implies a weak convergence of the first passage times $T(X_n) \rightarrow T(X)$. In conclusion, for all popular physical applications, the weak order of convergence is the important one. In consequence, the Milstein scheme is never a good choice, since it is equivalent to the Euler scheme in terms of weak order of convergence, but is more computationally intensive and harder to implement. We illustrate the surprising result of [17] by numerical simulation in a familiar setting.

A simple analytically solvable case of the first passage problem is a two-dimensional diffusion process in the (X, Y) plane with constant drift velocity $\mathbf{v}$ and constant diffusion coefficient σ^2 in a system with an absorbing barrier at $Y = Y_b = 2$. The problem admits an exact solution, in which the probability density function for the hitting time $t_{hit} = \operatorname{argmin}_t (Y_t > Y_b)$ is given by

$$p_{t_{hit}}(\tau) = \frac{Y_b}{\sqrt{2\pi\sigma^2\tau^3}} \exp\left(-\frac{(Y_b - v_y\tau)^2}{2\sigma^2\tau}\right), \tag{10}$$

and so $\mathbb{E}[t_{hit}] = 1/2$.

Numerically, we calculate the sample of the first passage times and locations for $\mathbf{v} = (v_x, v_y) = (0, 4)$ with stopping condition $Y = Y_b = 2$ and initial condition $(X, Y) = (2, 0)$. The Itō equations for the two-dimensional diffusion driven by Wiener processes W and W' with drift velocities $[v_x, v_y]$ take the form of

$$dX = v_x dt + \sigma dW_t, \tag{11}$$
$$dY = v_y dt + \sigma dW_t', \tag{12}$$

where σ is a parameter that describes the strength of the noise compared to the drift. We first transform this problem into a different set of coordinates, for which we choose polar coordinates. While we retain the analytical solution, we acquire non-linear terms in the governing equations, which will help us study the convergence issues which arise when noise and drift terms are generally dependent on the position. Eq. (12) transformed into polar coordinates (r, ϕ) is expressed as

$$dr = \left(\frac{1}{2r} + v_y \sin\phi + v_x \cos\phi\right) dt + \sigma\cos\phi\, dW_t + \sigma\sin\phi\, dW_t', \tag{13}$$
$$d\phi = \frac{1}{r}\left(v_y \cos\phi + v_x \sin\phi\right) dt - \sigma\frac{\sin\phi}{r} dW_t + \sigma\frac{\cos\phi}{r} dW_t'. \tag{14}$$

The transformed equations are nonlinear, coupled, with nondiagonal, noncommutative noise terms. Vector equations in the package `Pychastic` are defined by providing both the drift vector `a` and the noise matrix `b` as callable to the `SDEProblem` constructor. Denoting the configuration by $q = (r, \phi)$, we write

```
1  import jaq.numpy as jnp
2
3  y_drift = 4.0
4  sigma = 1
5  y_barrier = 2
6
7  def drift(q):
8      return jnp.array(
9          [1 / (2 * q[0]) + y_drift * jnp.sin(q[1]), y_drift * jnp.cos(q[1]) / q[
10         0]]
11     )
12
13 def noise(q):
14     return sigma * jnp.array(
15         [[jnp.cos(q[1]), jnp.sin(q[1])], [-jnp.sin(q[1]) / q[0], jnp.cos(q[1])
16         / q[0]]]
17     )
18
19 problem = pychastic.sde_problem.SDEProblem(
20     a=drift,
21     b=noise,
22     x0=jnp.array([2.0, 0.0]),
23     tmax=2.0,
24 )
```

The `jax.numpy` package is a functional cousin of `numpy` [19], which is the fundamental package for scientific computing with Python [20]. The functional focus of `jax.numpy` enables automatic differentiation, and thus facilitates the use of the high-order method without the need for numerical differentiation, which can be imprecise and computationally costly. We obtain the trajectories using the `solve` method of `SDESolver`.

```
1  solver = pychastic.sde_solver.SDESolver(dt=2**(-5), scheme='milstein')
2  solution = solver.solve_many(problem,n_samples,seed=0)
```

Pychastic supports simulating many trajectories simultaneously with the `solve_many` method. Our strategy allows for concurrency in computation of random variables and equation coefficients thanks to the `jax` package we use as back-end (which in turn relies on vectorised mathematical operations provided by the `XLA` supporting architecture of most modern processors). The `jax` package is a fusion of three capabilities: `numpy`-like API for array-based computing, functional transformations (such as vectorisation, parallelisation and automatic differentiation), and modular back-end, allowing developers to test their work with just CPU and deploy the same code later on a GPU (or TPU) capable hardware. Furthermore, we avoid notoriously slow `Python` loops by using `jax.lax.scan` routines, taking advantage of just-in-time compilation and asynchronous dispatch, avoiding the problem of global interpreter lock, which cripples imperatively coded Python programmes. A more detailed guide on the advantages (and common pifalls) of `jax` package can be found in its documentation on [ReadTheDocs](#).

Importantly, even though the Milstein and Wagner-Platen schemes require the values of spatial derivatives of noise and drift terms, we did not have to provide them explicitly. They were calculated using an automatic differentiation procedure from the coefficient functions provided in the `SDEProblem` constructor.

We compare the results obtained with different solvers and time steps in Fig. 1. Sample
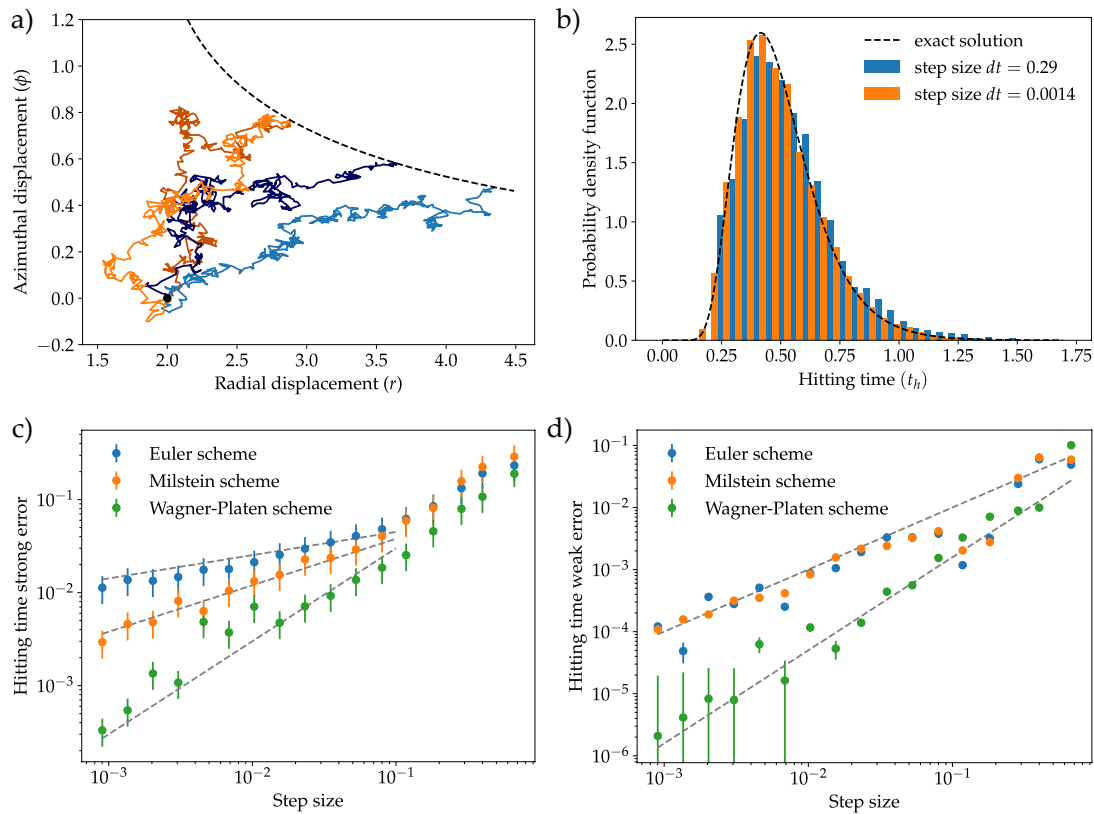
Figure 1: **a)** Four sample trajectories generated from Eq. (14) using the Euler algorithm together with the barrier $r \sin \phi = y = 2$. **b)** Distribution of hitting times for two different step sizes in the Euler algorithm together with the analytic solution of the problem, given by Eq. (10). Large step sizes typically overestimate the hitting time. **c)** Strong (path-wise) error of the hitting time for different step sizes and solvers together with power-law eye guides: $dt^{1/4}$, $dt^{1/2}$, dt. **d)** Error of the expected value of the first passage time together with eye-guides: $dt^{1/2}$, dt. Contrary to the strong error case, here Euler and Milstein algorithms coincide by Whitt's theorem.

trajectories starting from the point $(r, \phi) = (2, 0)$ are shown in Fig. 1a. The barrier at $y = 2$ becomes a curve in polar parameterisation. In Fig. 1 we present the resulting distribution of hitting times measured from an ensemble of $N = 1000$ trajectories with two different time-step sizes and compared to the exact solutions. Choosing a too large step size can lead to an overestimation of the typical hitting time. Having implemented three stochastic integration algorithms, we compare the strong error of estimation of the hitting time in Fig. 1c. It is a measure of the numerical error per trajectory, defined as $\langle |\tilde{\tau} - \tau| \rangle$, where $\tilde{\tau}$ is the numerical estimate of hitting time and τ is actual hitting time that can be computed using the exact solution and a particular realisation of the Wiener process. The average is taken over an ensemble of realisations of the Wiener process. For different schemes, the strong error scales differently with the step size – from the linear dependence on $dt^{1/4}$ for the Euler scheme, through $dt^{1/2}$ for the Milstein algorithm, to dt for Wagner-Platen. We note that these exponents are different (smaller) than for strong convergence at a fixed time [18]. Similarly, in Fig. 1d we present the weak error of estimation of the average value of the hitting time. The weak error is defined as $|\langle \tilde{\tau} \rangle - \langle \tau \rangle|$, and reflects the difference between the ensemble average value of hitting time estimate and the true ensemble average hitting time (computed directly from the theoreti-

cal distribution). This quantity highlights the lack of difference between Euler and Milstein schemes, in agreement with Whitt's theorem [17] stating the dependence of the expected first passage time on the weak convergence rate of the scheme only. The power laws for convergence – $dt^{1/4}$ for Euler and Milstein algorithms and dt for the Wagner-Platen scheme – again have smaller exponents than for those for weak convergence at a fixed time.

4.2 Rotational Brownian motion, `step_post_processing` function

To highlight the ease-of-use features of the `Pychastic` package, we present the problem of rotational Brownian diffusion simulations. General BD simulations that involve rigid bodies require the application of finite rotations to diffusing objects. The resulting algorithms are widely applied to study problems in physics and biology. While resolving translational motion is straightforward, e.g. with the standard Ermak-McCammon (Euler) algorithm [21], the rotational part is more involved to simulate, because the domain of rotational motion is $SO(3)$ and this has to be taken into account when solving the equations of motion. One difficulty lies in the commonly used rotational coordinate systems, such as Euler angles [22], which contain strong singularities around the polar orientations. When curvilinear coordinates are used to describe the motion, the metric tensor gives rise to nontrivial additional terms in the equations of motion, the so-called metric or drift terms, which are frequently overlooked in rotational BD algorithms. For a summary, see Ref. [23].

To overcome these limitations, we reimplement the problem following the rotation-vector-based formulation of Evensen *et al.* [24]. We describe the angular position of the particle by the angle of rotation Φ around a unit vector $\boldsymbol{\delta}$ collinear with the axis of rotation. Generalised coordinates are encoded in the vector $\boldsymbol{q} = \Phi\boldsymbol{\delta}$. The rotational mobility matrix of the particle in the body-fixed frame is $\boldsymbol{\mu}_{\text{body}}$. We transform it into the laboratory frame by

$$\boldsymbol{\mu} = \boldsymbol{\Omega}^T \cdot \boldsymbol{\mu}_{\text{body}} \cdot \boldsymbol{\Omega}, \tag{15}$$

where $\boldsymbol{\Omega}$ is the relevant rotation matrix. We also define a velocity transformation matrix $\boldsymbol{\Xi}$ to further introduce the transformed mobility $\widehat{\boldsymbol{\mu}}$ by

$$\widehat{\boldsymbol{\mu}} = \boldsymbol{\Xi} \cdot \boldsymbol{\mu} \cdot \boldsymbol{\Xi}^T. \tag{16}$$

Now we can write the Itō SDE corresponding to the evolution of the generalised coordinates as

$$d\boldsymbol{q} = \widehat{\boldsymbol{\mu}} \cdot \left(\frac{\partial}{\partial \boldsymbol{q}} \log V\right) dt + k_B T \left(\frac{\partial}{\partial \boldsymbol{q}} \cdot \widehat{\boldsymbol{\mu}}\right) dt + \sqrt{2k_B T} \widehat{\boldsymbol{\mu}}^{1/2} \cdot d\boldsymbol{W}. \tag{17}$$

Here, V is the density of the volume element ($SO(3)$ Haar measure) with respect to the Lebesgue measure on $\mathbb{R}^3$. In the literature $\partial \log V/\partial \boldsymbol{q}$ is often called the metric force $\boldsymbol{F}^{(m)}$. Explicit expressions for $\boldsymbol{\Xi}$, $\boldsymbol{\Omega}$ and $\log V$ are given in Appendix B.

Since the evolution equation depends on the divergence of a product of two orientation-dependent matrices, writing Eq. (17) explicitly is quite cumbersome. Earlier works of [24] and [26] avoid this difficulty by approximating the gradients by sampling the transformed mobility matrix at nearby locations in the phase space. We can address this problematic term directly thanks to the automatic differentiation capabilities of the `jax` package.

```
1  def metric_force(q):
2      # Metric force, providing the Boltzmann distribution in equilibrium.
3      phi = jnp.sqrt(jnp.sum(q ** 2))
4      scale = jax.lax.cond( # Taylor expansion for the polar orientations.
5          phi < 0.01,
6          lambda t: -t / 6.0,
7          lambda t: jnp.sin(t) / (1.0 - jnp.cos(t)) - 2.0 / t,
```
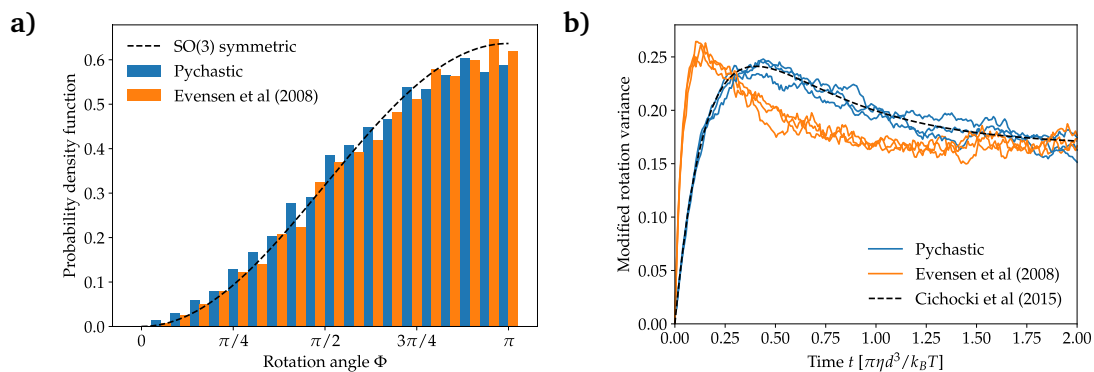
Figure 2: **a)** Distribution of rotation angles Φ after a long time in $N = 10^4$ simulations compared with the $SO(3)$ symmetric distribution $P^{eq}(\Phi)$ given by Eq. (18). **b)** Comparison of the time-dependent correlation structure arising from equation (17) for a spherical particle. Dashed lines show exact predictions from Ref. [25] and Eq. (19). Solid, colored lines show average values of 10^3 simulations (blue line corresponds to equation (17) and orange trajectories correspond to Ref. [24] containing a typo in their Eq. 12). Note that even though equilibrium distributions coincide, time dependent correlation structure is different.

```
8          phi,
9      )
10     return jax.lax.cond(
11         phi > 0.0, lambda: (q / phi) * scale, lambda: jnp.array([0.0, 0.0, 0.0
       ])
12     )
13
14 def t_mobility(q):
15     # Mobility matrix transformed to rotation vector coordinates.
16     Return transformation_matrix(q) @ mobility @ (transformation_matrix(q).T)
17
18 def drift(q):
19     # jax.jacobian has the differentiation index last (like mu_ij d_k) so
       divergence is contraction of the first and last axis.
20     return (
21     t_mobility(q) @ metric_force(q)
22     + jnp.einsum("iji->j", jax.jacobian(t_mobility)(q))
23     )
```

First, including the divergence of the mobility matrix in the new coordinate system is as simple as adding a correct contraction of `jax.jacobian` of the mobility tensor. To obtain this, we use the convenient `jnp.einsum` tool. This function, similarly to its analogue `np.einsum`, takes in a tensor, represented in the memory by an array of arrays of arrays etc. with the number of levels corresponding to the tensor rank, and performs contraction described symbolically by the index notation. In our case `"iji->j"` means 'contract the first index with the last index of a rank-3 tensor to produce a vector.'

Second, the domain of the equation is $SO(3)$, which is not covered by $\mathbb{R}^3$ (since $\pi_1(SO(3)) = \mathbb{Z}_2$ [16]). As a result, some trajectories, continuous in $SO(3)$, will necessarily be discontinuous in the $\mathbb{R}^3$ parameterisation. We adopt the method proposed by Ref. [24], where after each step of integration we project on the principal value of the rotation angle. In `pychastic`, we achieve this using the `step_post_processing` capability.

```
1  def canonicalize_coordinates(q):
2      phi = jnp.sqrt(jnp.sum(q ** 2))
3      max_phi = jnp.pi
4      canonical_phi = jnp.fmod(phi + max_phi, 2.0 * max_phi) - max_phi
5      return jax.lax.cond(
6          phi > max_phi,
7          lambda canonical_phi, phi, q: (canonical_phi / phi) * q,
8          lambda canonical_phi, phi, q: q,
9          canonical_phi,
10         phi,
11         q,
12     )
13 solver = pychastic.sde_solver.SDESolver(dt=0.01)
14 trajectories = solver.solve_many(
15     problem,
16     step_post_processing=canonicalize_coordinates,
17     n_trajectories=1000
18 )
```

The simulations were validated by checking the equilibrium properties and the time-dependent correlation structure. The equilibrium distribution $P^{eq}(\Phi)$ for the rotation angle Φ is given by

$$P^{eq}(\Phi) = \frac{1 - \cos\Phi}{\pi} \, . \tag{18}$$

To characterise the time evolution, we use the components of the modified rotation vector components $\Delta u_k(t) = -\frac{1}{2}\epsilon_{ijk}\Omega_{ij}(t)$, for which exact theoretical predictions were derived by Cichocki *et al.* [25] for an arbitrarily shaped molecule. For a spherical particle, the correlation functions are given by

$$\langle \Delta u_k(t) \Delta u_l(t) \rangle_0 = \left[\frac{1}{6} - \frac{5}{12} e^{-6D_r t} + \frac{1}{4} e^{-2D_r t} \right] \delta_{kl}^K \, , \tag{19}$$

where δ^K is the Kronecker delta. The diffusion coefficient for a sphere is $D_r = k_B T / \pi \eta d^3$, where d is the diameter of the particle.

In Fig. 2, we present a comparison between the results obtained with an algorithm based on Eq. (17) (blue lines), and a similar method based on equations (5-12) from Ref. [24] (orange). Fig. 2a shows the equilibrium distribution of the rotation angle Φ, together with the analytical prediction of Eq. (18). We note that although the equilibrium distribution predicted by both algorithms shows exact agreement with the theoretical predictions, the transformation matrix Ξ in Eq. (12) of Ref. [24] contains an error. We corrected this typographical error and present the proper formulation of the equations of motion matrices in the Appendix B. The discrepancy is visible in the time-dependent correlation function in Fig. 2b. Although at long times the system tends to the same equilibrium solution for both approaches, we see the agreement of our algorithm with the theoretical predictions of Eq. (19) (dashed line) and the deviation of the orange lines. This discrepancy highlights the need for better test cases that reliably test all properties of the simulated equation, and not solely the equilibrium distribution.

As mentioned above, the algorithm based on Ref. [24] is singularity-free, contrary to approaches based on Euler angles, which contain strong singularities around the polar orientations. However, it should be noted that the implementation of the code may still face some limitations, such as the computational inability to calculate $\sin\Phi/\Phi$ for $\Phi \to 0$. For these numerically restricted cases, we implemented the Taylor expansion of Ξ, Ω and $\log V$, based on Ref. [26]. However, we also report some typos in this publication. Therefore, in Appendix B we provide correct formulations of the Taylor-expanded terms.

4.3 Bead models with hydrodynamic interactions, `pygrpy` package integration

Hydrodynamically interacting beads (with and without springs) have been used successfully in modelling the properties of elastic macromolecules [21, 27–31]. Some questions about elastic macromolecules can be answered by computing the equilibrium ensemble of conformations (with methods such as Markov Chain Monte Carlo [32]). Simulations involving hydrodynamic interactions can, on the other hand, provide access to the dynamics and answer questions, e.g., about time scales of conformational change [33], mechanisms of protein association [34], pore translocation [35–37], near-wall hindered diffusion [38], and many other dynamical processes.

The starting point is often the Hamiltonian, which describes intramolecular interactions between the constitutive subunits of the molecule, modelled by a collection of beads. The potential energy landscape can then be used to compute the interaction forces. Thanks to the `jax autograd` capabilities, the forces arising from many-body mechanical interactions can be automatically calculated from the potential energy of the system.

The computation of hydrodynamic interactions, encoded in the mobility tensors of the respective macromolecules is a more involved task. Numerous methods are available, varying in scope, degree of precision, and complexity. For a review of popular methods applied to macromolecules, see Ref. [39]. For completeness, we supplemented `pychastic` with the package `pygrpy` for the calculation of grand mobility tensors in the Rotne-Prager-Yamakawa (RPY) approximation [40, 41], generalised to beads of different sizes [42]. The procedure is a Python port of the GRPY package [39]. The Python package `pygrpy` simplifies the implementation of similar bead-spring simulations, both stochastic and deterministic. Furthermore, `pygrpy` is compatible with the `jax` functional paradigm and allows automatic differentiation and vectorisation.

The RPY approximation is by far the most popular method of accounting for hydrodynamic interaction in numerical models of soft matter systems [43]. The mobility tensors calculated in this way preserve positive definiteness and are divergence-free, significantly simplifying the BD algorithm [21]. In essence, RPY is a far-field approximation that includes all terms that decay slower than the inverse third power of the interparticle distances, but is less accurate at smaller distances. When the particles come close together, it is necessary to include higher-order terms of the multipole expansion [44] and lubrication corrections [45]. However, as shown by Żuk *et al.* [42], the RPY approximation can be generalised to overlapping particles, and thus it can also be used to model complex-shaped particles as conglomerates of rigidly glued overlapping spheres, in particular to calculate hydrodynamic properties of biological macromolecules, as in the GRPY method [39].

To highlight the ease of use and interoperability of `pychastic` and `pygrpy`, we implement a benchmark problem proposed by Cichocki et al. [46]. It concerns the diffusion coefficient of an elastic "macromolecule" composed of 4 beads of radii $r_i \in \{3, 1, 1, 1\}$ joined into a string, as shown in Fig. 3a. Here, the length scale is the radius of a small bead a. The neighbouring beads interact directly via harmonic potentials, with the equilibrium distance $d_i = 4$ and the spring constant $k = 5.5 \, k_B T / a$, and indirectly through hydrodynamic interactions.

We determine the diffusion coefficient D by tracking the mean square displacement $\langle (\boldsymbol{x}(t) - \boldsymbol{x}(0))^2 \rangle$ (MSD) of a point $\boldsymbol{x}$ of the molecule. If the observation time is long enough, MSD $\approx 6Dt$. However, for shorter times, the coefficient of proportionality in the MSD(t) curve is different and depends on the choice of the reference point on the molecule. Although biophysical experiments typically measure the long-time diffusion coefficient, numerical simulations have easier access to short-time diffusivity. To minimise the difference, one should choose a particular point, called the centre of diffusion, which can be constructed as a weighted average of the positions of beads with weights determined from the hydrodynamic mobilities of an ensemble of equilibrium configurations of the elastic molecule [46].
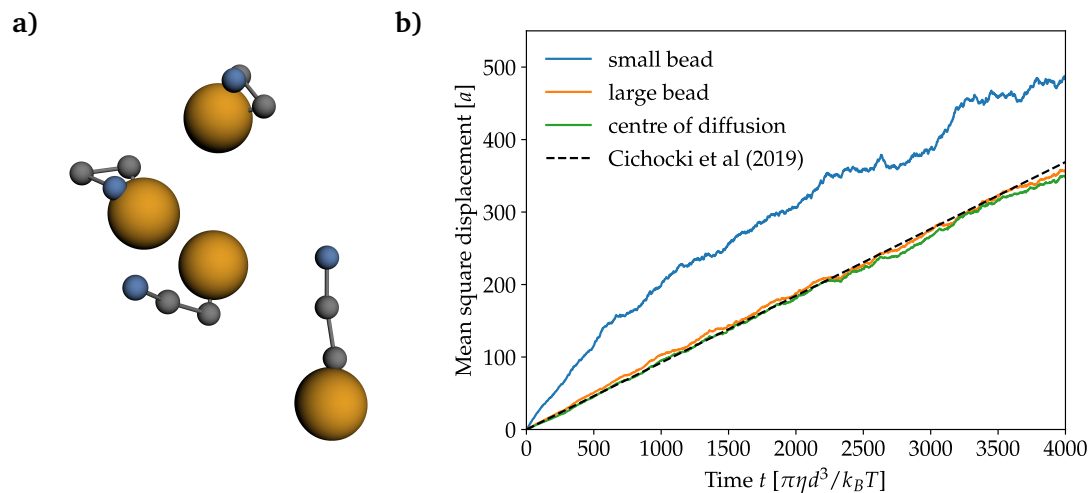
a)

b)

Figure 3: **a)** Representative configurations of four co-diffusing beads connected with harmonic springs at time (bottom to top) $t/\tau_d = 2, 2000, 4000, 6000$. **b)** Mean square displacement of three different tracked locations: the small bead at the end of the chain (blue), the large bead at the other end (orange) and the weighted average of all four beads with weights corresponding to the effective centre of diffusion, defined according to Ref. [46]. The dashed line corresponds to numerical results reported in [46] based on an extremely long, single-trajectory simulation.

We use this system to test our algorithm. In Fig. 3b, we show the temporal evolution of the MSD when different reference points on the elastic molecule are chosen: the small terminal bead, the large bead, and the centre of diffusion. The time is scaled by $\tau_d = \pi\eta d^3/k_B T$. We computed hydrodynamic interactions using the package pygrpy, which allows computations of mobility tensors for macromolecules composed of unequally sized and possibly overlapping spherical beads in the Rotne-Prager-Yamakawa approximation. The dashed line is the numerical result of [46] where the diffusion coefficient was estimated using a single very long BD trajectory. Indeed, we see that tracking the centre of diffusion provides a good estimate of the long-time behaviour of the MSD. When one chooses to track one of the smaller beads or one of the beads further away from the middle of the molecule, precision decreases primarily because rotational diffusion plays a bigger role in the motion of such tracers. The strength of our algorithm is that we use many trajectories instead of a single long one, thus avoiding the need for a more complex method of calculating the diffusion coefficient, involving recursive subdivisions of the simulation interval described by Frenkel and Smit [47].

```
1  radii = jnp.array([3.0,1.0,1.0,1.0]) # sizes of spheres used
2  n_beads = len(radii)
3  equilibrium_dist = 4.0
4  spring_constant = 5.5
5
6  def pot_ene(x): # potential energy
7      locations = jnp.reshape(x,(n_beads,3))
8      distance_ab = jnp.sqrt(jnp.sum((locations[0] - locations[1])**2))
9      distance_bc = jnp.sqrt(jnp.sum((locations[1] - locations[2])**2))
10     distance_cd = jnp.sqrt(jnp.sum((locations[2] - locations[3])**2))
11     ene = 0.5*spring_constant*jnp.sum(
12     ([distance_ab,distance_bc,distance_ca] - equilibrium_dist)**2
13     )
14     return ene
```

```
15
16 def drift(x):
17     locations = jnp.reshape(x,(n_beads,3))
18     mu = pygrpy.jax_grpy_tensors.muTT(locations,radii)
19     force = -jax.grad(pot_ene)(x)
20     return jnp.matmul(mu,force)
21
22 def noise(x):
23     locations = jnp.reshape(x,(n_beads,3))
24     mu = pygrpy.jax_grpy_tensors.muTT(locations,radii)
25     return jnp.sqrt(2)*jnp.linalg.cholesky(mu)
26
27 problem = pychastic.sde_problem.SDEProblem(
28     drift,
29     noise,
30     x0 = jnp.reshape(jnp.array([
31     [-2.,0.,0.],
32     [2.,0.,0.],
33     [6.,0.,0.],
34     [10.,0.,0.]
35     ]),(3*n_beads,)),
36     tmax = 10000.0)
```

5 Conclusion

We developed a novel Python package `pychastic` dedicated to efficient numerical solutions of SDEs. The package implements the classical truncated Taylor-Itō integrators up to strong order $\mathcal{O}(dt^{3/2})$ providing a precise treatment of both weak (e.g. equilibrium distributions, diffusion coefficients) and strong problems. We included a set of simple test cases that provide exact reference points for testing future stochastic integration algorithms. The analysis of three-dimensional rotational Brownian motion benchmarks is particularly important because it encompasses many of the difficult aspects of SDE approaches to Brownian dynamics: divergence terms in the evolution equation, handling discontinuous trajectories (unavoidable in the case of $SO(3)$), and spurious agreements when testing only the equilibrium distribution. We hope that `pychastic` will ease future studies of Brownian dynamics problems, especially problems that involve hydrodynamic interactions. The project is open source, and we hope to encourage collaboration and its further development.

Acknowledgements

The authors thank Piotr Szymczak for his insightful feedback.

A   Appendix: Typos in integration schemes in Ref. [18]

In the book: *Numerical solution of stochastic differential equations* by Peter E. Kloeden and Eckhard Platen in equation 5.8.11 we have found the following typos:

1. In the definition of $D^p_{j1,j2,j3}$: the fourth summand should read $\zeta_{j1,r}\eta_{j3,l+r}$ instead of $j1$.

2. In the definition of b_j: there is a $1/\pi$ factor missing before sum.

3. In the definition of $C^p_{j1,j2}$: second term should be $+1/r\,\eta_{j1,r}\eta_{j2,l}$ instead of $-l/r$.

B   Appendix: Rotational Brownian motion

In equation (17) we recall the equations of rotational Brownian motion, formulated in Ref. [24]. Angular orientation is described by the vector $\boldsymbol{a} = (a_1, a_2, a_3) = \Phi\boldsymbol{\delta}$, where Φ is the angle of rotation around a unit vector $\boldsymbol{\delta}$, which corresponds to the axis of rotation. Because the expressions in Ref. [24] contain typos, here we present the proper expressions for coordinate transformation and rotation matrices $\boldsymbol{\Xi}$ and $\boldsymbol{\Omega}$:

$$\boldsymbol{\Xi} = \left(\frac{1}{\Phi^2} - \frac{\sin\Phi}{2\Phi(1-\cos\Phi)}\right)\begin{pmatrix} a_1a_1 & a_1a_2 & a_1a_3 \\ a_2a_1 & a_2a_2 & a_2a_3 \\ a_3a_1 & a_3a_2 & a_3a_3 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} \frac{\Phi\sin\Phi}{1-\cos\Phi} & -a_3 & a_2 \\ a_3 & \frac{\Phi\sin\Phi}{1-\cos\Phi} & -a_1 \\ -a_2 & a_1 & \frac{\Phi\sin\Phi}{1-\cos\Phi} \end{pmatrix}, \quad (20)$$

$$\boldsymbol{\Omega} = \frac{1}{\Phi^2}\begin{pmatrix} \Phi^2\cos\Phi & -a_3\Phi\sin\Phi & a_2\Phi\sin\Phi \\ a_3\Phi\sin\Phi & \Phi^2\cos\Phi & -a_1\Phi\sin\Phi \\ -a_2\Phi\sin\Phi & a_1\Phi\sin\Phi & \Phi^2\cos\Phi \end{pmatrix} + \frac{1-\cos\Phi}{\Phi^2}\begin{pmatrix} a_1a_1 & a_1a_2 & a_1a_3 \\ a_2a_1 & a_2a_2 & a_2a_3 \\ a_3a_1 & a_3a_2 & a_3a_3 \end{pmatrix}. \quad (21)$$

The matrix $\boldsymbol{\Xi}$ transforms the velocities from the Cartesian coordinate system to the one described by $\boldsymbol{a}$. The matrix $\boldsymbol{\Omega}$ is a simple rotation matrix. These two matrices combined allow transformation of the body-fixed mobility matrix $\boldsymbol{\mu}_{\text{body}}$ to the lab-fixed, $\boldsymbol{a}$-described mobility matrix, given as $\widehat{\boldsymbol{\mu}} = \boldsymbol{\Xi} \cdot \boldsymbol{\Omega}^T \cdot \boldsymbol{\mu}_{\text{body}} \cdot \boldsymbol{\Omega} \cdot \boldsymbol{\Xi}^T$.

The term $\left(\frac{\partial}{\partial \boldsymbol{q}}\log V\right)$ from Eq. (17) can be associated with the metric force $\boldsymbol{F}^{(m)}$, which guarantees the Boltzmann distribution in equilibrium and is given as

$$\left(\frac{\partial}{\partial \boldsymbol{q}}\log V\right) = \boldsymbol{F}^{(m)} = -k_B T\left(\frac{\sin(\Phi)}{1-\cos(\Phi)} - \frac{2}{\Phi}\right)\boldsymbol{\delta}. \quad (22)$$

Due to the numerical limitations, it is sometimes inevitable to perform the Taylor expansion of the above quantities, as presented in Ref. [26]. However, we also found typos in this case. Therefore, we provide the correctly expanded matrices $\boldsymbol{\Xi}$ and $\boldsymbol{\Omega}$ for $\Phi \to 0$,

$$\boldsymbol{\Xi} = \frac{1}{12}\begin{pmatrix} a_1a_1 & a_1a_2 & a_1a_3 \\ a_2a_1 & a_2a_2 & a_2a_3 \\ a_3a_1 & a_3a_2 & a_3a_3 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} 2 & -a_3 & a_2 \\ a_3 & 2 & -a_1 \\ -a_2 & a_1 & 2 \end{pmatrix} + O(\Phi^2), \quad (23)$$

$$\boldsymbol{\Omega} = \begin{pmatrix} 1 & -a_3 & a_2 \\ a_3 & 1 & -a_1 \\ -a_2 & a_1 & 1 \end{pmatrix} + \frac{1}{2}\begin{pmatrix} a_1a_1 & a_1a_2 & a_1a_3 \\ a_2a_1 & a_2a_2 & a_2a_3 \\ a_3a_1 & a_3a_2 & a_3a_3 \end{pmatrix} + O(\Phi^2). \quad (24)$$

The Taylor-expanded metric force around $\Phi \to 0$ becomes

$$\boldsymbol{F}^{(m)} = k_B T\frac{\Phi}{6}\boldsymbol{\delta} + O(\Phi^3). \quad (25)$$

References

[1] P.-G. De Gennes, *Soft matter*, Science **256**, 495 (1992), doi:10.1126/science.256.5056.495.

[2] S. Kim and S. J. Karrila, *Microhydrodynamics: Principles and selected applications*, Butterworth-Heinemann, Oxford, UK, ISBN 9780750691734 (1991), doi:10.1016/C2013-0-04644-0.

[3] M. Praprotnik, L. D. Site and K. Kremer, *Multiscale simulation of soft matter: From scale bridging to adaptive resolution*, Annu. Rev. Phys. Chem. **59**, 545 (2008), doi:10.1146/annurev.physchem.59.032607.093707 .

[4] U. D. Schiller, T. Krüger and O. Henrich, *Mesoscopic modelling and simulation of soft matter*, Soft Matter **14**, 9 (2018), doi:10.1039/C7SM01711A.

[5] J. K. G. Dhont, *An introduction to dynamics of colloids*, Elsevier, Amsterdam, Netherlands, ISBN 9780080535074 (1996).

[6] M. Karplus and G. A. Petsko, *Molecular dynamics simulations in biology*, Nature **347**, 631 (1990), doi:10.1038/347631a0.

[7] N. van Kampen, *Stochastic processes in physics and chemistry*, Elsevier, Amsterdam, Netherlands, ISBN 9780080475363 (2007), doi:10.1016/B978-0-444-52965-7.X5000-4.

[8] R. M. Mazo, *Brownian motion*, Oxford University Press, Oxford, UK, ISBN 9780199556441 (2008), doi:10.1093/acprof:oso/9780199556441.001.0001.

[9] A. Einstein, *Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen*, Ann. Phys. **322**, 549 (1905), doi:10.1002/andp.19053220806.

[10] M. von Smoluchowski, *Zur kinetischen theorie der Brownschen molekularbewegung und der suspensionen*, Ann. Phys. **326**, 756 (1906), doi:10.1002/andp.19063261405.

[11] A. D. Fokker, *Die mittlere energie rotierender elektrischer dipole im strahlungsfeld*, Ann. Phys. **348**, 810 (1914), doi:10.1002/andp.19143480507.

[12] M. Planck, *Sitzungsberichte der Königlich Preussischen akademie der wissenschaften zu Berlin*, Deutsche Akademie der Wissenschaften zu Berlin, Berlin, Germany (1917).

[13] A. Kolmogoroff, *Über die analytischen methoden in der wahrscheinlichkeitsrechnung*, Math. Ann. **104**, 415 (1931), doi:10.1007/BF01457949.

[14] K. Itô, *On stochastic differential equations*, American Mathematical Society, Providence, Rhode Island, USA, ISBN 9780821812044 (1951).

[15] P. Langevin, *Sur la théorie du mouvement brownien*, C. R. Acad. Sci. **146**, 530 (1908).

[16] B. C. Hall, *Lie groups, Lie algebras, and representations: An elementary introduction*, Springer, Cham, Switzerland, ISBN 9783319134673 (2015), doi:10.1007/978-3-319-13467-3.

[17] W. Whitt, *Weak convergence of first passage time processes*, J. Appl. Prob. **8**, 417 (1971), doi:10.2307/3211913.

[18] P. Kloeden and E. Platen, *Numerical solution of stochastic differential equations*, Springer, Berlin, Heidelberg, ISBN 9783662126165 (1992), doi:10.1007/978-3-662-12616-5.

[19] C. R. Harris et al., *NumPy*, https://numpy.org/.

[20] C. R. Harris et al., *Array programming with NumPy*, Nature **585**, 357 (2020), doi:10.1038/s41586-020-2649-2.

[21] D. L. Ermak and J. A. McCammon, *Brownian dynamics with hydrodynamic interactions*, J. Chem. Phys. **69**, 1352 (1978), doi:10.1063/1.436761.

[22] H. Goldstein, *Classical mechanics*, Pearson Education, London, UK, ISBN 9788177582833 (2002).

[23] I. M. Ilie, W. J. Briels and W. K. den Otter, *An elementary singularity-free rotational Brownian dynamics algorithm for anisotropic particles*, J. Chem. Phys. **142**, 114103 (2015), doi:10.1063/1.4914322.

[24] T. R. Evensen, S. N. Naess and A. Elgsaeter, *Brownian dynamics simulations of rotational diffusion using the Cartesian components of the rotation vector as generalized coordinates*, Macromol. Theory Simul. **17**, 403 (2008), doi:10.1002/mats.200800031.

[25] B. Cichocki, M. L. Ekiel-Jeżewska and E. Wajnryb, *Brownian motion of a particle with arbitrary shape*, J. Chem. Phys. **142**, 214902 (2015), doi:10.1063/1.4921729.

[26] I. M. Ilie, W. K. den Otter and W. J. Briels, *Rotational Brownian dynamics simulations of clathrin cage formation*, J. Chem. Phys. **141**, 065101 (2014), doi:10.1063/1.4891306.

[27] R. M. Jendrejack, J. J. de Pablo and M. D. Graham, *Stochastic simulations of DNA in flow: Dynamics and the effects of hydrodynamic interactions*, J. Chem. Phys. **116**, 7752 (2002), doi:10.1063/1.1466831.

[28] R. G. Larson, T. T. Perkins, D. E. Smith and S. Chu, *Hydrodynamics of a DNA molecule in a flow field*, Phys. Rev. E **55**, 1794 (1997), doi:10.1103/PhysRevE.55.1794.

[29] A. Cressman, Y. Togashi, A. S. Mikhailov and R. Kapral, *Mesoscale modeling of molecular machines: Cyclic dynamics and hydrodynamical fluctuations*, Phys. Rev. E **77**, 050901 (2008), doi:10.1103/PhysRevE.77.050901.

[30] C.-C. Hsieh, S. Jain and R. G. Larson, *Brownian dynamics simulations with stiff finitely extensible nonlinear elastic-Fraenkel springs as approximations to rods in bead-rod models*, J. Chem. Phys. **124**, 044911 (2006), doi:10.1063/1.2161210.

[31] J. Huang and T. Schlick, *Macroscopic modeling and simulations of supercoiled DNA with bound proteins*, J. Chem. Phys. **117**, 8573 (2002), doi:10.1063/1.1511506.

[32] B. A. Berg, *Markov chain Monte Carlo simulations and their statistical analysis*, World Scientific, Singapore, ISBN 9789812389350 (2004), doi:10.1142/5602.

[33] S. He and H. A. Scheraga, *Macromolecular conformational dynamics in torsional angle space*, J. Chem. Phys. **108**, 271 (1998), doi:10.1063/1.475378.

[34] S. H. Northrup, J. A. Luton, J. O. Boles and J. C. L. Reynolds, *Brownian dynamics simulation of protein association*, J. Comput.-Aided Mol. Des. **1**, 291 (1988), doi:10.1007/BF01677278.

[35] T. Yamanouchi and A. Satoh, *Improvement of trapping performance of magnetic particles by magnetic multi-poles via Brownian dynamics simulations of magnetic rod-like particles in a Hagen-Poiseuille flow*, Mol. Phys. **120**, e2067503 (2022), doi:10.1080/00268976.2022.2067503.

[36] R. Waszkiewicz and M. Lisicki, *Hydrodynamic effects in the capture of rod-like molecules by a nanopore*, J. Phys.: Condens. Matter **33**, 104005 (2020), doi:10.1088/1361-648x/abd11b.

[37] L. Qiao and G. W. Slater, *Capture and translocation of a rod-like molecule by a nanopore: Orientation, charge distribution and hydrodynamics*, Phys. Chem. Chem. Phys. **24**, 6444 (2022), doi:10.1039/D2CP00313A.

[38] S. A. Rogers, M. Lisicki, B. Cichocki, J. K. G. Dhont and P. R. Lang, *Rotational diffusion of spherical colloids close to a wall*, Phys. Rev. Lett. **109**, 098305 (2012), doi:10.1103/PhysRevLett.109.098305.

[39] P. J. Zuk, B. Cichocki and P. Szymczak, *GRPY: An accurate bead method for calculation of hydrodynamic properties of rigid biomacromolecules*, Biophys. J. **115**, 782 (2018), doi:10.1016/j.bpj.2018.07.015.

[40] J. Rotne and S. Prager, *Variational treatment of hydrodynamic interaction in polymers*, J. Chem. Phys. **50**, 4831 (1969), doi:10.1063/1.1670977.

[41] H. Yamakawa, *Transport properties of polymer chains in dilute solution: Hydrodynamic interaction*, J. Chem. Phys. **53**, 436 (1970), doi:10.1063/1.1673799.

[42] P. J. Zuk, E. Wajnryb, K. A. Mizerski and P. Szymczak, *Rotne-Prager-Yamakawa approximation for different-sized particles in application to macromolecular bead models*, J. Fluid Mech. **741**, R5 (2014), doi:10.1017/jfm.2013.668.

[43] G. Nägele, *Brownian dynamics simulations*, Forschungszentrum Jülich, Jülich, Germany, ISBN 9783893364305 (2006).

[44] B. Cichocki, R. B. Jones, R. Kutteh and E. Wajnryb, *Friction and mobility for colloidal spheres in Stokes flow near a boundary: The multipole method and applications*, J. Chem. Phys. **112**, 2548 (2000), doi:10.1063/1.480894.

[45] M. L. Ekiel-Jeżewska and E. Wajnryb, *Precise multipole method for calculating hydrodynamic interactions between spherical particles in the stokes flow*, Transworld Research Network, Trivandrum, India, ISBN 9788178954004 (2009).

[46] B. Cichocki, M. Rubin, A. Niedzwiecka and P. Szymczak, *Diffusion coefficients of elastic macromolecules*, J. Fluid Mech. **878**, R3 (2019), doi:10.1017/jfm.2019.652.

[47] D. Frenkel and B. Smit, *Understanding molecular simulation: From algorithms to applications*, Academic Press, Cambridge, UK, ISBN 9780122673511 (2002), doi:10.1016/B978-0-12-267351-1.X5000-7.