Scientific Report

Classical Molecular Dynamics & Friction Tensor Calculation

Alberto Prato - University of Padua

albertoprato@studenti.unipd.it

Contents

1 Introduction **1**

1.1 Model system 1

1.2 Zwanzig-Mori projection 1

1.3 Time evolution implementation 2

2 Programming strategy **4**

2.1 Architectural paradigm 4

2.2 Language choice: Fortran 4

3 Module description **5**

3.1 Force and potential energy calculation: `force_module` 6

3.1.1 Computational optimization 6

3.2 Initial energy minimization: `minimization_module` 7

3.3 Velocity initialization: `velocity_init_module` 7

3.3.1 Inertial frame correction and temperature rescaling 8

3.4 Equilibration phase and main molecular dynamics 8

3.4.1 Equilibration criteria 8

3.4.2 Main MD 9

3.5 Time correlation function: (`fft_correlation_module`) 9

3.5.1 The fast Fourier transform method 10

3.5.2 Implementation with FFTW library 10

3.6 Friction tensor computation: (`friction_module`) 11

4 Results and Discussion **12**

4.1 Simulation setup and parameters 12

4.2 Equilibration analysis 12

4.3 Friction tensor analysis 13

4.4 Conclusions and Future Perspectives 15

A Derivation of equation (7) **16**

B Periodic boundary conditions **19**

C Conjugate gradient method **20**

D The Box-Muller transform **21**

1 Introduction

Classical molecular dynamics (MD) is a computational framework that enables the sampling of phase space trajectories of N-body systems, connecting microscopic Hamiltonian to macroscopic ensemble averages. In the context of solvation dynamics, this approach allows the evaluation of dissipative properties through the analysis of force fluctuations. This report presents a theoretical derivation and a computational implementation designed to compute the time-dependent friction tensor (memory kernel) of a massive solute, from first principles.

1.1 Model system

We consider a system composed of four interacting massive moieties, treatable as spheres, embedded in a bath of N particles. The full deterministic classical phase space decomposes into a 12-dimensional subsystem of "slow" Cartesian coordinates $\mathbf{R}$, with conjugate momenta $\mathbf{P}$, and into a $3N$-dimensional bath described by coordinates $\mathbf{r}$ and conjugate momenta $\mathbf{p}$. Here is assumed that all slow coordinates share the same mass M and all bath coordinates share the same mass m.

The Hamiltonian of the system reads

$$H = \frac{1}{2}\,\mathbf{P}^{\mathrm{T}}\,\mathbf{M}^{-1}\,\mathbf{P} + V(\mathbf{R}) + \frac{1}{2}\,\mathbf{p}^{\mathrm{T}}\,\mathbf{m}^{-1}\,\mathbf{p} + u_{\mathrm{B}}(\mathbf{r}) + u_{\mathrm{int}}(\mathbf{R},\mathbf{r}) \tag{1}$$

$$= H_0(\mathbf{R},\mathbf{P}) + H_{\mathrm{B}}(\mathbf{r},\mathbf{p},\mathbf{R}) \tag{2}$$

where $V(\mathbf{R})$, is an external potential acting on the set of slow coordinates (e.g., the PES, which can depend parametrically on $\mathbf{r}$), $u_{\mathrm{B}}(\mathbf{r})$ is the interaction potential among the bath coordinates, while $u_{\mathrm{int}}(\mathbf{R},\mathbf{r})$ describes the interaction energy between the slow coordinates and the bath coordinates. In the last equality the Hamiltonian has been divided in two parts. H_0 describes the dynamics of the slow coordinates in vacuum. H_{B} is the Hamiltonian for the solvent interacting with the slow coordinates fixed at $\mathbf{R}$ (and zero momenta).

The Liouville equation for the probability density $\rho(\mathbf{Q},\mathbf{\Pi},t)$, with $\mathbf{Q} = (\mathbf{R},\mathbf{r})$ and $\mathbf{\Pi} = (\mathbf{P},\mathbf{p})$, is

$$\frac{\partial}{\partial t}\rho(\mathbf{Q},\mathbf{\Pi},t) = -i\hat{\mathcal{L}}\rho(\mathbf{Q},\mathbf{\Pi},t) \tag{3}$$

with

$$-i\hat{\mathcal{L}} = \{H,\cdot\} = \frac{\partial H}{\partial \mathbf{Q}}\cdot\frac{\partial}{\partial \mathbf{\Pi}} - \frac{\partial H}{\partial \mathbf{\Pi}}\cdot\frac{\partial}{\partial \mathbf{Q}} \tag{4}$$

1.2 Zwanzig-Mori projection

To recover the Fokker-Planck equation one proceeds with the Zwanzig-Mori projection [1] of the bath coordinates under the hypothesis that such coordinates are fast with respect

to $\mathbf{R}$ and $\mathbf{P}$. This is ensured by the conditions $m \ll M$. The projection operator reads

$$\hat{\mathcal{P}} = \rho_{\mathrm{B}}(\mathbf{r}, \mathbf{p}|\mathbf{R}) \int_V d\mathbf{r} d\mathbf{p} \tag{5}$$

where

$$\rho_{\mathrm{B}}(\mathbf{r}, \mathbf{p}|\mathbf{R}) = \frac{\exp[-\beta H_{\mathrm{B}}(\mathbf{r}, \mathbf{p}, \mathbf{R})]}{Z(\mathbf{R})} \tag{6}$$

with $\beta = (k_{\mathrm{B}}T)^{-1}$, ρ_{B} the Boltzmann equilibrium distribution conditional to the slow coordinates fixed on the value $\mathbf{R}$ and $Z(\mathbf{R})$ the partition function.

It is possible to recover an equation for the projected probability density $\tilde{\rho}(\mathbf{R}, \mathbf{P}, t)$ which reads

$$\begin{aligned} \frac{\partial}{\partial t}\tilde{\rho}(\mathbf{R}, \mathbf{P}, t) = &- \left(\mathbf{M}^{-1}\mathbf{P}\right) \cdot \frac{\partial}{\partial \mathbf{R}}\tilde{\rho}(\mathbf{R}, \mathbf{P}, t) - \tilde{\mathbf{F}}_{\mathrm{ext}}(\mathbf{R}) \cdot \frac{\partial}{\partial \mathbf{P}}\tilde{\rho}(\mathbf{R}, \mathbf{P}, t) \\ &+ \int_0^t d\tau \, \frac{\partial}{\partial \mathbf{P}} \cdot \left[\tilde{\boldsymbol{\xi}}(\mathbf{R}, \tau) \cdot \left(\frac{\partial}{\partial \mathbf{P}} + \beta \mathbf{M}^{-1}\mathbf{P}\right) \tilde{\rho}(\mathbf{R}, \mathbf{P}, t - \tau)\right] \end{aligned} \tag{7}$$

under the conditions: (i) the solvent is initially in thermal equilibrium with solute, (ii) $m \ll M$, (iii) $N \to \infty$, and (iv) the dynamics is followed along time intervals larger than the characteristic time scales of solvent molecules. The derivation of the equation is reported in appendix A.

The friction tensor is given by

$$\tilde{\boldsymbol{\xi}}(\mathbf{R}, t) = \int_V d\mathbf{r} d\mathbf{p} \, \delta\mathbf{F}_{\mathrm{int}}(\mathbf{r}, \mathbf{R}) \cdot \left[e^{-i\hat{\mathcal{L}}_{\mathrm{B}}t} \delta\mathbf{F}_{\mathrm{int}}(\mathbf{r}, \mathbf{R})\right]^{\mathrm{T}} \rho_{\mathrm{B}}(\mathbf{r}, \mathbf{p}|\mathbf{R}) \tag{8}$$

$$= \left\langle \delta\mathbf{F}(0) \cdot \delta\mathbf{F}(t)^{\mathrm{T}} \right\rangle_{\mathrm{B}} \tag{9}$$

This expression represents the time-autocorrelation function of the fluctuating part of the interaction force, averaged over the equilibrium canonical distribution of the bath, under the condition of steady solute on configuration $\mathbf{R}$. Equation (9) constitutes the theoretical core of this work. The temporal decay of this memory kernel characterizes the non-Markovian nature of the bath; a rapid decay implies a separation of time scales compatible with a Markovian approximation.

1.3 Time evolution implementation

As indicated by equation (9), computing the friction tensor requires sampling the stochastic force history exerted by the bath on a static solute. Consequently, the simulation strategy involves integrating the classical equations of motion for the solvent particles while keeping the solute degrees of freedom frozen.

The interaction terms $u_{\mathrm{B}}(\mathbf{r})$ and $u_{\mathrm{int}}(\mathbf{R}, \mathbf{r})$ in the Hamiltonian (1) are modeled via the Lennard-Jones (12-6) potential. The general form for the pairwise interaction energy is

$$u(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \tag{10}$$

where distinct parameter sets are adopted for solvent-solvent and solute-solvent interactions. The force exerted on particle i by particle j is derived analytically as the negative gradient of the potential

$$\mathbf{F}_{ij}(\mathbf{r}_{ij}) = \frac{24\epsilon}{r_{ij}^2}\left[2\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]\mathbf{r}_{ij} \tag{11}$$

Once the potentials are known, we solve the Newton's equations of motion, $\mathbf{F} = m\,\ddot{\mathbf{r}}$, applying a finite-difference approach. In particular, a successful simulation algorithm is the velocity Verlet algorithm [2], chosen for its simplicity, time-reversibility and numerical stability. The algorithm propagates the positions $\mathbf{r}$ and velocities $\mathbf{v}$ from time t to $t + \delta t$ via a three-step procedure:

$$\mathbf{v}\left(t + \tfrac{1}{2}\delta t\right) = \mathbf{v}(t) + \frac{\delta t}{2m}\mathbf{F}(t) \tag{12}$$

$$\mathbf{r}(t + \delta t) = \mathbf{r}(t) + \delta t\,\mathbf{v}\left(t + \tfrac{1}{2}\delta t\right) \tag{13}$$

$$\mathbf{v}(t + \delta t) = \mathbf{v}\left(t + \tfrac{1}{2}\delta t\right) + \frac{\delta t}{2m}\mathbf{F}(t + \delta t) \tag{14}$$

this corresponds to a "half-kick, drift, half-kick" scheme: velocities are advanced by a half-step using the current forces, positions are drifted using the updated half-step velocities, forces are re-evaluated at the new positions, and finally, velocities are fully advanced. This scheme translates directly into a Fortran implementation:

```
! Force evaluation at t
CALL force_calculation(...)
! Half-kick
vel_solv = vel_solv + 0.5_wp * dt * (force / mass_solv)
! Drift
pos_solv = pos_solv + dt * vel_solv
! Force evaluation at t + dt
CALL force_calculation(...)
! Half-kick
vel_solv = vel_solv + 0.5_wp * dt * (force / mass_solv)
```

in this snippet, `pos_solv`, `vel_solv`, and `force` are $(N, 3)$ arrays representing the phase space coordinates and forces of the solvent particles.

2 Programming strategy

2.1 Architectural paradigm

The software is designed using a procedural approach, which is a computer-programming paradigm that aligns with the structure of this many-body physical problem. Unlike object-oriented programming, which fragments the problem into discrete objects with associated methods, the procedural approach allows the system's data to be stored in contiguous memory arrays, enabling the evaluation of the forces and time-integration steps in efficient loops.

Furthermore, the code is organized into distinct `MODULE`s that encapsulate specific physical jobs. This ensures that the definition of the physics remains decoupled from the numerical integration algorithms.

2.2 Language choice: Fortran

The implementation utilizes Fortran 90, which was selected for its advantages in MD simulations. A primary benefit is that the language treats arrays as first-class mathematical entities. This allows for a direct and readable translation of physical equations into source code. For instance, the drift step of velocity Verlet algorithm is implemented as a single vector instruction, `vel = vel + 0.5 * dt * force / mass`. Additionally, Fortran offers explicit dynamic memory management (`ALLOCATE`/`DEALLOCATE`), which is essential for defining the system size at runtime.

The overall logical flow of the program is depicted in Fig. 1.

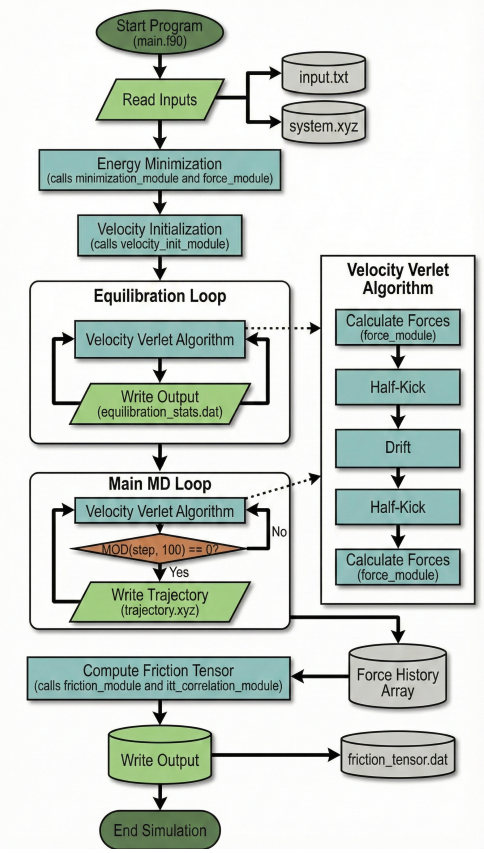

Figure 1: Flowchart of the MD simulation workflow.

3 Module description

As illustrated in the workflow diagram (Fig. 1), the simulation is driven by the `main.f90` program. This unit manages the sequential execution of the computational stages, as it handles:

(i) the reading of simulation parameters (`input.txt`) and the initial configuration (`system.xyz`);

(ii) the geometric optimization of the initial solvent structure;

(iii) the initialization of solvent particles' velocities drawn from a Maxwell distribution at the target temperature;

(iv) an equilibration phase, during which the system relaxes from a lattice-like to a liquid state;

(v) the main molecular dynamics, where the phase space trajectory is simulated and the history of the forces exerted on the solute is recorded;

(vi) the final analysis, which processes the stored force history to compute the time-dependent friction tensor.

Following this structural hierarchy, the specific computational tasks are delegated to five specialized modules. These are described below in the order of their invocation and dependency.

3.1 Force and potential energy calculation: `force_module`

The `force_module` constitutes the physical core of the simulation, responsible for the evaluation of the total potential energy and the forces acting on all particles.

The module implements the Lennard-Jones 12-6 potential (10) to model both solvent-solvent and solute-solvent interactions. To simulate a bulk fluid environment, the routine implements periodic boundary conditions (PBC) utilizing the "minimum image convention". This ensures topological consistency, allowing particles to interact with the nearest periodic image of their neighbors. The implementation details of the PBC are provided in Appendix B.

3.1.1 Computational optimization

To reduce the computational cost, a spherical truncation is applied. Interactions between particles separated by a distance $r_{ij} > r_{\text{cutoff}}$ are neglected. For a typical cutoff radius of $r_{\text{cutoff}} = 2.5\,\sigma$, the magnitude of the Lennard-Jones potential at the boundary is just 1.6% of the well depth ϵ, rendering the error introduced by truncation negligible.

The numerical evaluation of the potential energy and forces requires a summation over particle pairs. To optimize efficiency, the algorithm exploits Newton's third law, which allows the summation to be restricted to unique pairs (i, j) where $i < j$. This reduces the number of force evaluations by a factor of two. The resulting nested loop structure is illustrated below:

```
! Loop over the particle i
DO i = 1, n_solv - 1
  ! Loop over the particle k (unique pairs only)
  DO k = i + 1, n_solv
      ...
      ! Evaluate forces and potential
      ...
  END DO
END DO
```

3.2 Initial energy minimization: `minimization_module`

The initialization of a MD simulation requires the construction of a starting configuration from packing algorithms. A frequent outcome of such procedures is the occurrence of "steric clashes". These high-energy overlaps result in enormous repulsive forces which, upon integration, would lead to numerical divergence. To resolve this, the system must undergo a structural relaxation process prior to the assignment of thermal velocities. Physically, this corresponds to a descent on the potential energy surface toward a local minimum, effectively cooling the system to 0 K.

The `minimization_module` implements this optimization using the Conjugate Gradient (CG) method, specifically the Polak-Ribière variant. Unlike the Steepest Descent method, which follows the local negative gradient and often suffers from oscillatory "zig-zag" convergence, the CG method constructs a search direction that is conjugate to the previous direction, ensuring a faster convergence rate. Further details on the CG method can be found in Appendix C.

3.3 Velocity initialization: `velocity_init_module`

The transition from a static, energy-minimized configuration (0 K) to a dynamic trajectory requires the assignment of initial velocities. The `velocity_init_module` performs this stochastic initialization by sampling atomic velocities from the Maxwell distribution at the target temperature T.

In the canonical ensemble, the probability density function for a Cartesian velocity component v_α (where $\alpha \in \{x, y, z\}$) of a particle with mass m is given by the Gaussian form

$$\rho(v_\alpha) = \sqrt{\frac{m}{2\pi k_B T}} \exp\left(-\frac{m v_\alpha^2}{2 k_B T}\right) \tag{15}$$

This distribution is characterized by a zero mean and a standard deviation of $\sigma = \sqrt{k_B T/m}$. To generate numerical values adhering to this distribution, the module employs the Box-Muller transform, an algorithm that maps uniformly distributed random numbers, $\xi \in (0, 1)$, onto the normal distribution. Some details of this transformation are provided in Appendix D. The implementation of the random assignment is shown below:

```
sigma_v = SQRT((kb * temp) / mass)    ! Standard deviation
CALL RANDOM_SEED()

DO i = 1, n_solv
  DO k = 1, 3
    CALL RANDOM_NUMBER(r1)    ! Generates uniform number in (0,1)
    CALL RANDOM_NUMBER(r2)    ! Generates uniform number in (0,1)

    ! Gaussian number with mean 0 and std.dev. = sigma_v
```

```
      vel_solv(i, k) = sigma_v * SQRT(-2.0_wp * LOG(r1)) &
                       * COS(2.0_wp * pi * r2)
    END DO
  END DO
```

3.3.1 Inertial frame correction and temperature rescaling

Due to the stochastic nature of the sampling, the initial random assignment may result in a non-zero net velocity. This would manifest as a translation of the entire system center of mass (COM) during the simulation. To ensure the dynamics evolve in a inertial reference frame, the module explicitly computes the COM velocity and subtracts it from every particle. Subsequently, to ensure the instantaneous kinetic temperature matches the target temperature, the velocities are rescaled based on the equipartition theorem ($E_{kin} = \frac{3}{2}Nk_BT$).

3.4 Equilibration phase and main molecular dynamics

Following the structural relaxation (`minimization_module`) and the assignment of velocities (`velocity_init_module`), the system must undergo an equilibration phase. During this period, the trajectory is integrated using the velocity Verlet scheme described in Section 1.3, but the phase space data is discarded. The objective is to allow the system to escape the initial minimum configuration and achieve the structural disorder characteristic of a fluid. At the end of this equilibration period, all memory of the initial configuration should be lost. This loss of memory is essential for the computation of the friction tensor, which is strictly valid only when measuring fluctuations around thermal equilibrium rather than non-equilibrium reorganization forces.

3.4.1 Equilibration criteria

The equilibration of the system can be monitored by analyzing the time-evolution of the potential energy and the mean squared displacement (MSD). The transitions to the liquid phase is considered complete only when the following criteria are met:

1. In the initial stages, the potential energy typically exhibits an increase from the deep minima of the crystal-like packing to the higher average energy of a liquid. The equilibration period should be extended at least until the potential energy ceases to drift and begins to oscillate around a constant mean value.

2. The physical state of the solvent is also verified by calculating the MSD of the solvent particles relative to their initial positions

$$\mathrm{MSD}(t) = \frac{1}{N}\sum_{i=1}^{N}\left\langle \left|\mathbf{r}_i(t) - \mathbf{r}_i(0)\right|^2 \right\rangle \tag{16}$$

A solid-like state is characterized by an MSD that oscillates around a constant mean value, whereas a liquid state is identified by a linear growth of the MSD with time.

To further demonstrate the readability and clarity of the Fortran code, the implementation of the MSD calculation is shown below:

```
msd = 0.0_wp
DO i = 1, n_solv
  dist_sq = 0.0_wp
  DO k = 1, 3
    dist_sq = dist_sq + (pos_solv(i, k) - pos_solv0(i, k))**2
  END DO
  msd = msd + dist_sq
END DO
msd = msd / DBLE(n_solv)
```

During the equilibration phase, the time evolution of these quantities is saved to `equilibration_stats.dat` to verify that the relaxation time is sufficient. The plots of these parameters are presented in Section 4.2.

3.4.2 Main MD

Once equilibration is certified, the main MD loop commences. In this phase, the system is propagated for a defined number of steps using the same integrator. The crucial difference is that the forces exerted by the bath on the solute particles are explicitly stored at every time step into the `force_history` array. This dataset serves as the input for the analysis performed by the subsequent modules.

Furthermore, as illustrated in Fig. 1, the system trajectory is sampled and saved to the file `trajectory.xyz` during this phase. To minimize disk usage, the coordinates are written every 100 steps, implemented via the conditional logic `IF (MOD(step, 100) == 0)`.

3.5 Time correlation function: (`fft_correlation_module`)

The central objective of the post-processing phase is the evaluation of the memory kernel $\tilde{\boldsymbol{\xi}}(t)$, defined as the time-autocorrelation function of the fluctuating forces exerted on the solute. This function quantifies the persistence of memory in the system as it measures how long a force fluctuation at time t_0 continues to influence the system at a later time $t_0 + t$.

For a discrete time series of force vectors sampled at intervals δt, we label successive timesteps with τ, such that $t = \tau\,\delta t$. Formally, the discrete autocorrelation function

between two force components α and β for a lag time τ is defined as

$$C_{\alpha\beta}(\tau) = \langle \delta F_\alpha(0)\, \delta F_\beta(\tau) \rangle = \frac{1}{\tau_{\max}} \sum_{\tau_0=1}^{\tau_{\max}} \delta F_\alpha(\tau_0)\, \delta F_\beta(\tau_0 + \tau) \tag{17}$$

where the bracket notation indicates an average over $\tau_{\max}$ time origins.

3.5.1 The fast Fourier transform method

A direct evaluation of equation (17) requires nested summations for every lag time τ, resulting in an algorithmic complexity of $\mathcal{O}(N^2)$. For long MD trajectories (typically $10^5 - 10^6$ steps) this becomes a computational bottleneck. To overcome this, the `fft_correlation_module` invokes the convolution/correlation theorem, which relates the correlation of two signals in the time domain to pointwise multiplication in the frequency domain.

It follows from this theorem that the autocorrelation of a real signal simplifies to the inverse transform of the spectral density

$$C_{\alpha\beta}(\tau) = \mathcal{F}^{-1}\left[\delta \hat{F}^*_\alpha(\omega)\, \delta \hat{F}_\beta(\omega)\right] \tag{18}$$

where $\delta \hat{F}(\omega) = \mathcal{F}[\delta F(t)]$ is the discrete Fourier transform of the fluctuating force component. By utilizing the fast Fourier transform (FFT) algorithm the complexity is reduced to $\mathcal{O}(N \log_2 N)$.

3.5.2 Implementation with FFTW library

The `fft_correlation_module` utilizes the FFTW3 (*Fastest Fourier Transform in the West*) library to perform the forward and backward transforms. The core Fortran implementation, capable of computing both auto- ($\alpha = \beta$) and cross-correlations ($\alpha \neq \beta$), is shown below:

```
! Calculate fluctuations from input arrays x and y
mean_x = SUM(x) / DBLE(N)
mean_y = SUM(y) / DBLE(N)

! Copy real values into complex vector
DO i = 1, N
  signal_x(i) = CMPLX(x(i)- mean_x, 0.0_wp, KIND=C_DOUBLE_COMPLEX)
  signal_y(i) = CMPLX(y(i)- mean_y, 0.0_wp, KIND=C_DOUBLE_COMPLEX)
END DO

! Forward FFT (from time to frequency)
CALL fftw_execute_dft(plan_fwd_x, signal_x, fft_x)
CALL fftw_execute_dft(plan_fwd_y, signal_y, fft_y)

! Compute power spectrum ( |F(w)|^2 )
```

```
DO i = 1, N_pad    !N_pad = 2 * N (required for linear correlation)
  power_spectrum(i) = CONJG(fft_x(i)) * fft_y(i)
END DO

! Backward FFT (from frequency to time)
CALL fftw_execute_dft(plan_bwd, power_spectrum, result_complex)
```

3.6 Friction tensor computation: (`friction_module`)

The final stage of the program is managed by the `friction_module`. This unit allows the calculation of the friction tensor elements $\xi_{ij}^{\alpha\beta}(t)$ by invoking the FFT routine.

In the current version available in the repository, the implementation is explicitly restricted to the computation of the diagonal elements, $\xi_{ii}^{\alpha\alpha}(t)$. This choice was made to minimize memory consumption and focuses the analysis on the self-friction coefficients experienced by the four solute particles.

However, the code allows for a straightforward extension to the full tensor, which would capture both spatial cross-correlations ($\alpha \neq \beta$) and coupling between distinct particles ($i \neq j$). The generalized algorithm required to compute the complete tensor consists of a quadruply nested loop structure, as demonstrated below:

```
DO I = 1, n_solute
  DO a = 1, 3
    ! Extract force history for solute particle I
    force_traj_A = force_history(I, a, :)

    DO J = 1, n_solute
      DO b = 1, 3
        ! Extract force history for solute particle J
        force_traj_B = force_history(J, b,  :)

        ! Compute Correlation via FFT
        CALL compute_correlation_fft(force_traj_A, force_traj_B, &
                                     tensor_element)
      END DO
    END DO
  END DO
END DO
```

in this snippet, `force_history` is a three-dimensional array containing the complete time series of the forces exerted on the solute. The three indices correspond to the solute particle $(1, 2, 3, 4)$, the Cartesian dimension (x, y, z), and the time step, respectively.

4 Results and Discussion

To validate the computational implementation and demonstrate the capabilities of the code, a simulation was performed on a representative sand-box.

4.1 Simulation setup and parameters

The illustrative test case consists of a solution of 4 massive solute particles (modeled as carbon atoms) embedded in a bath of 11.452 solvent particles (modeled as water). The initial configuration was generated using the `Packmol` software [5], which distributed the molecules within a cubic simulation box of side length $L = 20.0$ Å.

To emulate a realistic chemical environment, the Lennard-Jones interaction parameters were adapted from the standard OPLS-AA (for solute) and TIP3P (for solvent) force fields as reported by Jorgensen *et al.* [4]. The simulation employs a consistent system of units: distances in Ångströms, time in picoseconds, mass in Daltons, and temperature in Kelvin. The specific simulation parameters provided in the `input.txt` file are listed below:

```
50000 0.001                ! nk, dt (ps)
18.015 63.597 3.1507       ! Solvent: m (Dalton), epsilon_OO, sigma_OO
43.472 3.338               ! Solute: epsilon_CO, sigma_CO (angstrom)
298.15 0.831               ! Temperature (K), Boltzmann const
20.0                       ! L_box used in packmol
```

The system dynamics were propagated for a total duration of 50 ps, discretized into 50.000 time steps of $\delta t = 0.001$ ps.

4.2 Equilibration analysis

The first phase of the analysis concerns the equilibration of the system. As shown in Figure 2, the transition from the static, energy-minimized initial condition to a dynamic liquid state is monitored via the total potential energy and the MSD.

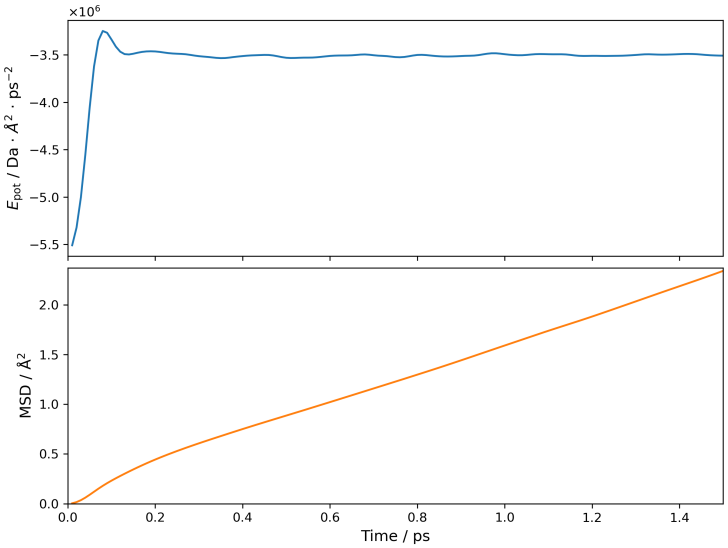

Figure 2: Time evolution of system properties during the initial equilibration phase.

The upper panel of Fig. 2 illustrates the rapid thermalization of the system. The potential energy rises sharply from the deep minimum of the optimized packing and stabilizes around a stationary mean value within the first few fractions of a picosecond. This indicates that the system has successfully escaped the initial energy minimum. Simultaneously, the bottom panel shows that the MSD exhibits an immediate linear time-dependence, characteristic of the diffusive regime of a liquid, confirming that no lattice-like memory of the initial Packmol configuration remains.

4.3 Friction tensor analysis

The molecular dynamics trajectory was analyzed to compute the time-dependent friction tensor. As detailed in Section 3.6, the current implementation focuses on the diagonal elements of the tensor, $\xi_{ii}^{\alpha\alpha}(t)$.

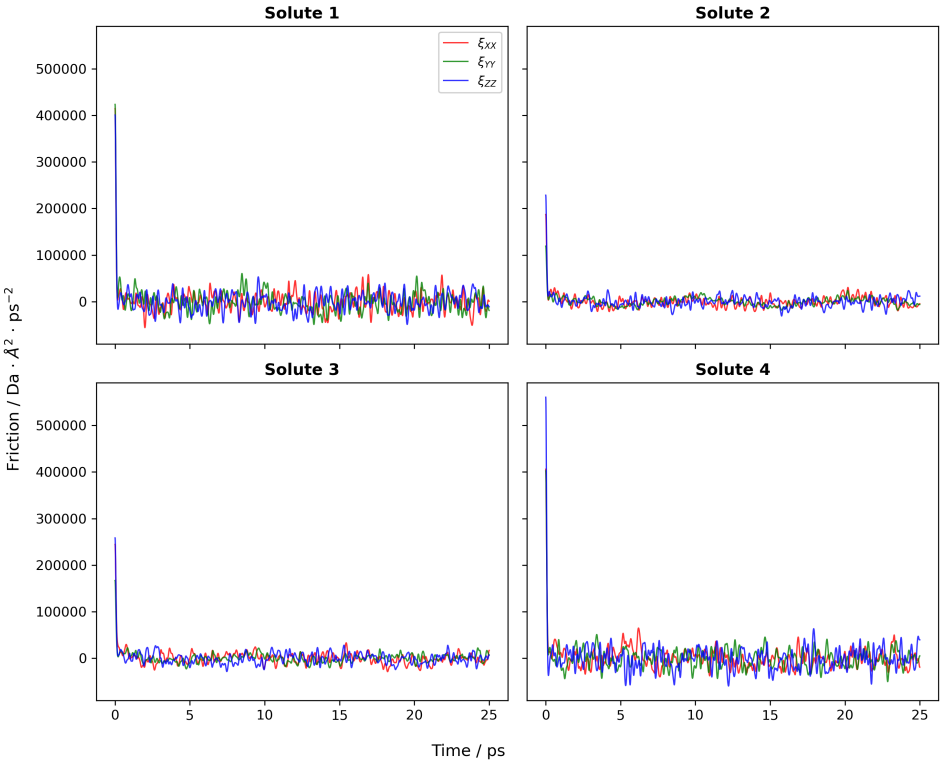

Figure 3: Time-dependent memory kernel $\tilde{\xi}(t)$ (diagonal components).

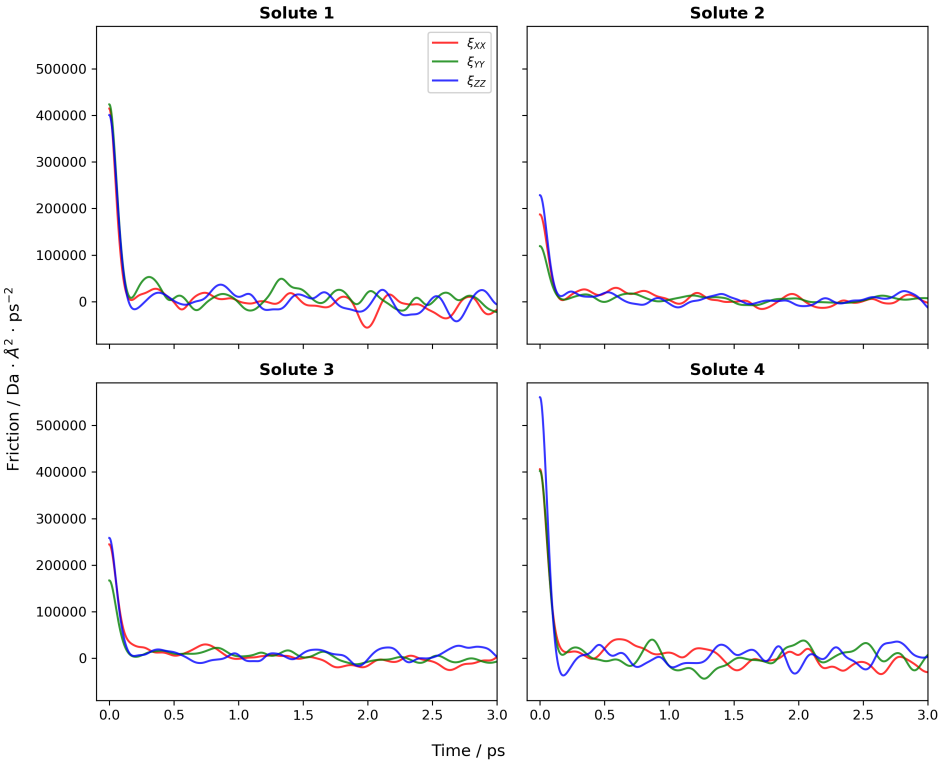

Figure 4: Short-time behavior of the memory kernel $\tilde{\xi}(t)$ (0 - 3 ps).

Fig. 3 displays the time evolution of the friction kernel for the four solute particles.

To resolve the relevant timescales of the solvent-solute coupling, the analysis focuses on the short-time behavior shown in Fig. 4. The memory kernel is characterized by a large initial value at $t = 0$. This is followed by a rapid decay occurring within a characteristic timescale of $0.1 - 0.2$ ps. The rapid loss of correlation suggests that for phenomena occurring on timescales $t \gg 0.5$ ps, memory effects become negligible, and the dynamics can be effectively described within the Markovian approximation.

4.4 Conclusions and Future Perspectives

The computational framework presented herein establishes a valid alternative to hydrodynamic models for the evaluation of the friction tensor. While standard hydrodynamic approaches rely on the hydrodynamic radius, which is a phenomenological parameter that can become ill-defined or physically misleading, the present method computes dissipation directly from microscopic forces.

Despite the successful validation of the current implementation, future development will extend the program's capabilities in two key aspects:

1. Systematically varying the solute geometry to map the friction tensor $\tilde{\boldsymbol{\xi}}(\mathbf{R}, t)$ across the conformational landscape.

2. Implementing the transformation to project the Cartesian friction tensor onto internal coordinates.

Appendix A: Derivation of equation (7)

It is possible to divide the Liouvillean into

$$i\hat{\mathcal{L}} = i\hat{\mathcal{L}}_0 + i\hat{\mathcal{L}}_{\mathrm{B}} \tag{19}$$

with

$$i\hat{\mathcal{L}}_0 = \mathbf{P}^{\mathrm{T}}\,\mathbf{M}^{-1}\cdot\frac{\partial}{\partial\mathbf{R}} + \mathbf{F}_0(\mathbf{r},\mathbf{R})\cdot\frac{\partial}{\partial\mathbf{P}} \tag{20}$$

and

$$i\hat{\mathcal{L}}_{\mathrm{B}} = \mathbf{p}^{\mathrm{T}}\,\mathbf{m}^{-1}\cdot\frac{\partial}{\partial\mathbf{r}} + \mathbf{F}_{\mathrm{B}}(\mathbf{r},\mathbf{R})\cdot\frac{\partial}{\partial\mathbf{p}} \tag{21}$$

with the forces

$$\mathbf{F}_0(\mathbf{R},\mathbf{r}) = -\frac{\partial}{\partial\mathbf{R}}(V(\mathbf{R}) + u_{\mathrm{int}}(\mathbf{R},\mathbf{r})) = \mathbf{F}_{\mathrm{ext}}(\mathbf{R}) + \mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}(\mathbf{R},\mathbf{r}) \tag{22}$$

and

$$\mathbf{F}_{\mathrm{B}}(\mathbf{r},\mathbf{R}) = -\frac{\partial}{\partial\mathbf{r}}(u_{\mathrm{B}}(\mathbf{r}) + u_{\mathrm{int}}(\mathbf{r},\mathbf{R})) = \mathbf{F}_{\mathrm{BB}}(\mathbf{r}) + \mathbf{F}_{\mathrm{int}}^{(\mathbf{r})}(\mathbf{r},\mathbf{R}) \tag{23}$$

where $\mathbf{F}_{\mathrm{ext}}(\mathbf{R})$ is the force from the external potential (PES), $\mathbf{F}_{\mathrm{BB}}(\mathbf{r})$ is the internal force among bath particles, $\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}(\mathbf{R},\mathbf{r})$ is the force exerted by the bath on the system and $\mathbf{F}_{\mathrm{int}}^{(\mathbf{r})}(\mathbf{r},\mathbf{R})$ is the force exerted by the system on the bath.

From now on the dependence on the coordinates $\mathbf{Q},\mathbf{\Pi}$ has been dropped to keep the expressions more compact. We call $f(t)$ and $g(t)$ the two projections

$$f(t) = \hat{\mathcal{P}}\rho(t) = \rho_{\mathrm{B}}\,\tilde{\rho}(t) \tag{24}$$

and

$$g(t) = (1-\hat{\mathcal{P}})\rho(t) \tag{25}$$

By applying $\hat{\mathcal{P}}$ and $\hat{\mathcal{Q}} = 1-\hat{\mathcal{P}}$ to equation (3) and solving formally for $g(t)$ (assuming the bath is initially thermally equilibrated, i.e., the $g(0)$ term is equals to zero), we obtain

$$\frac{\partial f(t)}{\partial t} = -i\hat{\mathcal{P}}\hat{\mathcal{L}}f(t) + \int_0^t d\tau\,\hat{\mathcal{P}}i\hat{\mathcal{L}}\,e^{-i\hat{\mathcal{Q}}\hat{\mathcal{L}}\tau}\,i\hat{\mathcal{Q}}\hat{\mathcal{L}}f(t-\tau). \tag{26}$$

which is the Nakajima-Zwanzig generalized master equation.

The first term on the right-hand side of equation (26) represents the average evolution (reversible dynamics). Using equations (19) and (24) and noting that $i\hat{\mathcal{L}}_{\mathrm{B}}\rho_{\mathrm{B}} = 0$, the first term reads

$$-i\hat{\mathcal{P}}\hat{\mathcal{L}}f(t) = -\rho_{\mathrm{B}}\left[\mathbf{M}^{-1}\mathbf{P}\cdot\frac{\partial}{\partial\mathbf{R}} + \tilde{\mathbf{F}}_{\mathrm{ext}}\cdot\frac{\partial}{\partial\mathbf{P}}\right]\tilde{\rho}(t) \tag{27}$$

where $\tilde{\mathbf{F}}_{\mathrm{ext}}$ is the total force acting on the slow coordinates, averaged over the equilibrium bath distribution

$$\tilde{\mathbf{F}}_{\mathrm{ext}} = \int_V d\mathbf{r}d\mathbf{p}\,\mathbf{F}_0\,\rho_{\mathrm{B}} \tag{28}$$

The integral term of equation (26) represents the dissipative dynamics. First, we want to evaluate the term $i\hat{\mathcal{Q}}\hat{\mathcal{L}}(\rho_{\mathrm{B}}\tilde{\rho})$ of the integrand. Using the product rule (Leibniz)

$$i\hat{\mathcal{L}}(\rho_{\mathrm{B}}\tilde{\rho}) = (i\hat{\mathcal{L}}\rho_{\mathrm{B}})\tilde{\rho} + \rho_{\mathrm{B}}(i\hat{\mathcal{L}}\tilde{\rho}) \tag{29}$$

and since ρ_{B} does not depend on $\mathbf{P}$ but depends on $\mathbf{R}$ via the interaction potential and the partition function

$$i\hat{\mathcal{L}}\rho_{\mathrm{B}} = \mathbf{M}^{-1}\mathbf{P}\cdot\frac{\partial}{\partial\mathbf{R}}\rho_{\mathrm{B}} \tag{30}$$

where

$$\frac{\partial\rho_{\mathrm{B}}}{\partial\mathbf{R}} = \rho_{\mathrm{B}}\frac{\partial}{\partial\mathbf{R}}\left(-\beta u_{\mathrm{int}} - \ln Z\right) \tag{31}$$

$$= \rho_{\mathrm{B}}\beta\left(\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})} - \langle\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\rangle_{\mathrm{B}}\right) = \beta\rho_{\mathrm{B}}\,\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})} \tag{32}$$

where in the last equality we identified the force $\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}$ and the mean force $\langle\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\rangle_{\mathrm{B}} = \beta^{-1}\frac{\partial}{\partial\mathbf{R}}\ln Z$ and we defined the fluctuating force $\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})} = \mathbf{F}_{\mathrm{int}}^{(\mathbf{R})} - \langle\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\rangle_{\mathrm{B}}$. Thus the first addend of equation (29) reads

$$(i\hat{\mathcal{L}}\rho_{\mathrm{B}})\tilde{\rho} = \beta\left(\mathbf{M}^{-1}\mathbf{P}\cdot\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\right)\rho_{\mathrm{B}}\tilde{\rho}. \tag{33}$$

The second addend of equation (29) is simply evaluated applying $i\hat{\mathcal{L}}_0$ to $\tilde{\rho}$

$$\rho_{\mathrm{B}}(i\hat{\mathcal{L}}\tilde{\rho}) = \rho_{\mathrm{B}}\left(\mathbf{M}^{-1}\mathbf{P}\cdot\frac{\partial\tilde{\rho}}{\partial\mathbf{R}} + (\mathbf{F}_{\mathrm{ext}} + \mathbf{F}_{\mathrm{int}}^{(\mathbf{R})})\cdot\frac{\partial\tilde{\rho}}{\partial\mathbf{P}}\right) \tag{34}$$

We now apply $\hat{\mathcal{Q}}$ which removes any term that is averaged over the bath, so that the surviving terms are only those containing the interaction force fluctuations

$$i\hat{\mathcal{Q}}\hat{\mathcal{L}}f(t) = \rho_{\mathrm{B}}\,\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\cdot\left(\frac{\partial}{\partial\mathbf{P}} + \beta\mathbf{M}^{-1}\mathbf{P}\right)\tilde{\rho}(t) \tag{35}$$

where we see that $i\hat{\mathcal{Q}}\hat{\mathcal{L}}$ acts as a "fluctuation generator".

We substitute equation (35) into the integral term of equation (26). We adopt the approximation of a fast bath, replacing the projected propagator $e^{-i\hat{\mathcal{Q}}\hat{\mathcal{L}}\tau}$ with the bath propagator $e^{-i\hat{\mathcal{L}}_{\mathrm{B}}\tau}$. This implies that the system variables are effectively frozen on the timescale of bath relaxation. The integral term reads

$$\mathcal{I} = \int_0^t d\tau\,\hat{\mathcal{P}}i\hat{\mathcal{L}}\,e^{-i\hat{\mathcal{L}}_{\mathrm{B}}\tau}\left[\rho_{\mathrm{B}}\,\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\cdot\left(\frac{\partial}{\partial\mathbf{P}} + \beta\mathbf{M}^{-1}\mathbf{P}\right)\tilde{\rho}(t-\tau)\right] \tag{36}$$

The operator $\hat{\mathcal{P}}i\hat{\mathcal{L}}$ involves an integration over the bath variables. Again, the only non-vanishing contribution comes from the $\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}\cdot\frac{\partial}{\partial\mathbf{P}}$ term within $i\hat{\mathcal{L}}_0$. The integral term becomes

$$\begin{aligned}\mathcal{I} &= \int_0^t d\tau\frac{\partial}{\partial\mathbf{P}}\cdot\left[\int_V d\mathbf{r}d\mathbf{p}\,\rho_{\mathrm{B}}\,\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}(0)\cdot\left(e^{-i\hat{\mathcal{L}}_{\mathrm{B}}\tau}\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}(0)\right)^{\mathrm{T}}\right]\cdot\left(\frac{\partial}{\partial\mathbf{P}} + \beta\mathbf{M}^{-1}\mathbf{P}\right)\tilde{\rho}(t-\tau)\\ &= \int_0^t d\tau\frac{\partial}{\partial\mathbf{P}}\cdot\left[\int_V d\mathbf{r}d\mathbf{p}\,\rho_{\mathrm{B}}\,\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}(0)\cdot\delta\mathbf{F}_{\mathrm{int}}^{(\mathbf{R})}(-\tau)^{\mathrm{T}}\right]\cdot\left(\frac{\partial}{\partial\mathbf{P}} + \beta\mathbf{M}^{-1}\mathbf{P}\right)\tilde{\rho}(t-\tau)\end{aligned} \tag{37}$$

The term in the square brackets is the time-correlation function of the fluctuating forces, identified as the friction tensor

$$\tilde{\boldsymbol{\xi}}(\tau) = \left\langle \delta\mathbf{F}_{\text{int}}^{(\mathbf{R})}(0) \cdot \delta\mathbf{F}_{\text{int}}^{(\mathbf{R})}(\tau)^{\text{T}} \right\rangle_{\text{B}} \tag{38}$$

Combining the drift term from equation (27) and the integral term derived above we arrive at the equation for the evolution of the projected probability density

$$\begin{aligned} \frac{\partial}{\partial t}\tilde{\rho}(t) = &- \left(\mathbf{M}^{-1}\mathbf{P}\right) \cdot \frac{\partial}{\partial \mathbf{R}}\tilde{\rho}(t) - \tilde{\mathbf{F}}_{\text{ext}} \cdot \frac{\partial}{\partial \mathbf{P}}\tilde{\rho}(t) \\ &+ \int_0^t d\tau \, \frac{\partial}{\partial \mathbf{P}} \cdot \left[\tilde{\boldsymbol{\xi}}(\tau) \cdot \left(\frac{\partial}{\partial \mathbf{P}} + \beta \mathbf{M}^{-1}\mathbf{P} \right) \tilde{\rho}(t-\tau) \right] \end{aligned} \tag{39}$$

Appendix B: Periodic boundary conditions

To approximate the properties of a macroscopic bulk fluid the simulation employs periodic boundary conditions. The central cubic simulation cell of side length L is conceptually replicated throughout space to form an infinite lattice. Consequently, particles are free to cross the boundaries of the primary cell; topological continuity is preserved by the re-entry of the corresponding periodic image through the opposite face, ensuring the conservation of number density.

For systems governed by short-range potentials, the calculation of pairwise interactions is defined by the minimum image convention. Under this protocol, a particle i interacts solely with the nearest periodic image of any other particle j. A prerequisite for the validity of the minimum image convention is that $r_{\text{cutoff}} \leq L/2$, which guarantees that a particle does not interact with its own periodic image.

In the Fortran implementation, the periodic boundaries and minimum image distances are handled using the nearest integer function `ANINT`. To maintain coordinates within the central box (assuming the origin is at the center), particle positions are re-mapped whenever they drift across a boundary:

```
pos_solv(i, 1) = pos_solv(i, 1) - &
                 box_L * ANINT(pos_solv(i, 1) / box_L)
```

where the first index of `pos_solv(:,:)` select the i-th solvent particle and the second index the dimension. Here `box_L` is a variable containing the cubic box length L.

Most critically, the distance vector $\mathbf{r}_{ij}$ must represent the shortest path between particles through the periodic lattice. This is calculated by applying the same wrapping logic to the distance vector:

```
diff(dim) = pos(j, dim) - pos(i, dim)
diff(dim) = diff(dim) - box_L * ANINT(diff(dim) / box_L)
```

This operation ensures that `diff(dim)` correctly represents the component of the distance vector between particle i and the closest image of particle j.

Appendix C: Conjugate gradient method

The method assumes that near a local minimum, the potential energy surface $f(\mathbf{x})$ can be approximated by a quadratic expansion

$$f(\mathbf{x}) \approx c - \mathbf{b} \cdot \mathbf{x} + \frac{1}{2}\mathbf{x} \cdot \mathbf{A} \cdot \mathbf{x} \tag{40}$$

where $\mathbf{A}$ is the Hessian matrix of second derivatives. The algorithm iteratively updates the system configuration by performing a line minimization along a search direction $\mathbf{h}_i$. The search direction, which is defined as a linear combination of the current gradient $\mathbf{g}_{i+1}$ and the previous search direction $\mathbf{h}_i$

$$\mathbf{h}_{i+1} = \mathbf{g}_{i+1} + \gamma_i \mathbf{h}_i \tag{41}$$

This implementation utilizes the Polak-Ribière variant, in which the scalar γ_i is computed as

$$\gamma_i = \frac{(\mathbf{g}_{i+1} - \mathbf{g}_i) \cdot \mathbf{g}_{i+1}}{\mathbf{g}_i \cdot \mathbf{g}_i} \tag{42}$$

where $\mathbf{g}_i$ and $\mathbf{g}_{i+1}$ are the gradient vectors at the current and next iteration, respectively.

The algorithm iterates until the maximum force component acting on any particle falls below a convergence tolerance (e.g., 10^{-4} reduced units).

Appendix D: The Box-Muller transform

In molecular dynamics, the initialization of particle velocities at a finite temperature T requires sampling from the Maxwell distribution. This is mathematically equivalent to generating random numbers from a normal distribution. Since standard pseudorandom number generators typically provide values uniformly distributed on the interval $(0, 1)$, a transformation method is required to map these uniform numbers onto the target distribution.

A random number ζ' chosen from a distribution with mean μ and variance σ is related to a number ζ generated from the normal distribution with zero mean and unit variance by

$$\zeta' = \mu + \sigma\,\zeta. \tag{43}$$

The problem is reduced to sampling ζ. One way (of many) to do this is the Box-Muller transform, which involves two steps and the generation of two uniform random numbers [3]:

(a) generate independent uniform random numbers ξ_1 and ξ_2 on $(0, 1)$;

(b) calculate $\zeta_1 = \sqrt{-2\ln\xi_1}\cos(2\pi\xi_2)$ and $\zeta_2 = \sqrt{-2\ln\xi_1}\sin(2\pi\xi_2)$

To obtain velocity components v_α with the correct physical variance $\sigma_v^2 = k_B T/m$, the standard variate must be scaled by the thermal velocity σ_v.

Although the transform yields two independent normal numbers, for reasons of algorithmic simplicity, the implementation presented in the `velocity_init_module` utilizes only ζ_1 (the cosine term) to determine each velocity component. The final expression implemented is therefore:

$$v_\alpha = \mu + \sigma_v\,\zeta_1 = 0 + \sqrt{\frac{k_B T}{m}}\left(\sqrt{-2\ln\xi_1}\cos(2\pi\xi_2)\right) \tag{44}$$

This derivation corresponds exactly to the implementation presented in the `velocity_init_module`.

References

[1] Zwanzig, R. Phys. Rev. 1961, 124, 983–992.

[2] Swope, W. C.; Andersen, H. C.; Berens, P. H.; Wilson, K. R. J. Chem. Phys. 1982, 76, 637–649.

[3] Box, G. E. P.; Muller, M. E. Ann. Math. Stat. 1958, 29, 610–611.

[4] Jorgensen, W. L.; Maxwell, D. S.; Tirado-Rives, J. J. Am. Chem. Soc. 1996, 118, 11225–11236;
Jorgensen, W. L.; Chandrasekhar, J.; Madura, J. D.; Impey, R. W.; Klein, M. L. J. Chem. Phys. 1983, 79, 926–935.

[5] Martinez, L.; Andrade, R.; Birgin, E. G.; Martinez, J. M. J. Comput. Chem. 2009, 30, 2157–2164.