Supporting Information for Field-Directed Self-Assembly of Mutually Polarizable Nanoparticles

Zachary M. Sherman, Dipanjan Ghosh, James W. Swan
zsherman@mit.edu, jswan@mit.edu

Documentation for the `MutualDipole` Plugin

This plugin creates a new class `MutualDipole`, which extends the `ForceCompute` class from HOOMD-Blue. When the class is created, it adds a new force to be computed in HOOMD-Blue: the electric/magnetic force on polarizable particles in an externally applied field. The particles are polarized by the external field and by the disturbance fields generated by other particle's dipole moments. Because the particle dipoles are all mutually coupled, the plugin solves a system of linear equations to find all of the particle dipoles simultaneously. Once the dipole moments are calculated, the force on each particle is then computed.

The particles are assumed to all be spheres with the same radius $a = 1$ in dimensionless units. The conductivity $\lambda_{p,i}$ of each particle i may be different and the fluid/solvent has a conductivity of $\lambda_f = 1$ in dimensionless units. Mixtures of nonpolarizable and polarizable particles may be used, and the plugin will ignore the nonpolarizable particles. System parameters, including the direction and strength of the applied field, can be changed over the course of a simulation with Python functions in the `MutualDipole` class. The computation time scales approximately linearly with the number of particles up to 10^6 particles (linear scaling may hold for even more particles but we did not test this) and logarithmically with the inverse of the error tolerance. Mutual polarization can be turned off so that the particles are only polarized by the external field and the dipole moments are constant over the course of the simulation.

```
class MutualDipole.compute.MutualDipole(group, conductivity, field, xi = 0.5, errortol = 1e-3,
    fileprefix = "", period = "0", constantdipoleflag = "0")
```

Parameters:

- `group` (`hoomd.group`) – The group of particles for which to compute the electric/magnetic dipoles and forces. This is referred to as the "active" group throughout the documentation and the comments in the code. `group` can be any subset of particles, but it cannot be empty. `group.all` is the HOOMD group for all particles.
- `conductivity` (`float list`) – A list of the conductivity $\lambda_{p,i}$ of each of the active particles in units of fluid conductivity λ_f. The length of the list is equal to the number of active particles, *i.e.* the number of particles in `group`. "Conductivity" is used generically to refer to either electric permittivity in the case of dielectric suspensions or magnetic permeability in the case of paramagnetic suspensions. Perfect conductors have a conductivity of `float("inf")` while perfect insulators have a conductivity of 0. A conductivity of 1 indicates that the particles and fluid have equal conductivities, and the particles do not polarize in an applied field. In this plugin, no particle's conductivity can be set to 1 because it causes terms to diverge. Instead, particles with conductivity of 1 should not be included in the active group because they do not polarize and would exert/feel no electric/magnetic forces. NOTE: The limit of particle conductivity approaching 1 corresponds to the constant dipole model, where the dipole moments of particles do not mutually polarize each other. If the constant dipole model is desired, `constantdipoleflag` must be set to 1. In this case, conductivity and field are redundant, and it is only the product $\beta_i \mathbf{E}_0$, where $\beta_i \equiv (\lambda_{p,i} - 1)/(\lambda_{p,i} + 2)$ and $\mathbf{E}_0$ is the external field in `field`, that is

important. The particle conductivity can be set arbitrarily as long as the product $\beta_i \mathbf{E}_0$ is the desired value.

- `field` (`float list`) – A list of the x, y, and z components of the externally applied electric/magnetic field $\mathbf{E}_0$ in units of $\sqrt{E/a^3\lambda_f}$, where E is the energy scale of the simulation and a is the particle radius. While the energy scale in the simulation can be chosen arbitrarily, the length scale of the simulation MUST be the particle radius, *i.e.* the plugin assumes a particle radius of 1 in dimensionless units. This is important for the numerical precision of the plugin.
- `xi` (`float`) – Ewald splitting parameter ξ that controls the convergence rates of the real space and wave space Ewald sums. If ξ is increased, the wave space sum converges slower while the real space sum converges faster and more time is spent computing the wave space sum. If ξ is decreased, the wave space sum converges faster while the real space sum converges slower and more time is spent computing the real space sum. ξ can be optimized to apportion the computation time between the two sums so that the total computation time is minimized. The default value of 1/2 distributes the computation time more or less equally between the two sums and is typically a good choice. Depending on the system size, concentration, or structure, the optimal ξ can change. The optimal ξ is nearly always in the range $0.1 < \xi < 1$.
- `errortol` (`float`) – Desired error tolerance for the numerical method. This is the maximum error in each element of the matrix/vector multiplication of the potential tensor and dipole vector. This is also the error tolerance used for the iterative solution to the linear system of equations for the particle dipoles. This tolerance is *not* the error in the particle dipoles and forces. The error in the particle dipoles and forces is affected by the condition number of the matrix in the linear system of equations, and it is possible for the error to accumulate for large numbers of particles. The numerical method is spectrally accurate, so the computation time is logarithmic in the error tolerance (i.e. an order of magnitude decrease in the error tolerance corresponds to a linear increase in the computation time).
- `fileprefix` (`string`) – Prefix for the names of output files containing the particle positions, dipole moments, and electric/magnetic forces. The frame number and file extension are automatically appended to the specified `fileprefix` each time an output file is written. `fileprefix = "output"` would generate file names like output.0000000000.txt, output.0000000100.txt, output.0000000200.txt, and so on.
- `period` (`int`) – Number of time steps between writing output files. A file containing the particle positions, dipole moments, and electric/magnetic forces will be output on the current time step and then every `period` time steps. The positions are in units of a, dipoles in units of $\sqrt{Ea^3\lambda_f}$, and forces in units of E/a, where a is the particle radius, E is the energy scale, and λ_f is the fluid conductivity. If `period = 0`, output files are not written.
- `constantdipoleflag` (`int`) – Indicates whether or not to use the constant dipole model instead of the mutual dipole model. If `constantdipoleflag = 0`, the mutual dipole model is used and particles can mutually polarize one another. This is the default setting. If `constantdipoleflag = 1`, the mutual dipole model is turned off and the constant dipole model is used instead. In the constant dipole model, the dipole moment of each particle is $\mathbf{S}_i = 4\pi a^3 \lambda_f \beta_i \mathbf{E}_0$, where $\beta_i = (\lambda_{p,i} - 1)/(\lambda_{p,i} + 2)$, $\lambda_{p,i}$ is the particle conductivity in `conductivity`, and $\mathbf{E}_0$ is the external field in `field`. The units of the dipole are $\sqrt{a^3\lambda_f E}$, where a is the particle radius, λ_f is the fluid conductivity, and E is the energy scale of the simulation. Only the product $\beta_i \mathbf{E}_0$ is important. The particle conductivity and field can be set arbitrarily as long as the product $\beta_i \mathbf{E}_0$ is the desired value. Particles do not mutually polarize one another and the value of each particle's dipole moment remains constant throughout the simulation. Because the plugin does not need to solve a linear system of equations for the particle dipoles, the computation time is decreased for the constant dipole model compared to the mutual dipole model, usually by a factor of 2 or 3. The difference in computation time depends on the number of matrix/vector products needed to iteratively solve for the dipoles in the mutual dipole model. The constant dipole model is significantly less accurate than the mutual dipole model and does not capture key qualitative features of real suspensions, so this model should be used cautiously.

Functions:

`MutualDipole.update_field(field)`

Change the externally applied field. Only updates the field and particle dipoles and does not recompute any other parameters. Note that for `constantdipoleflag = 1` (constant dipole model), this changes the particle dipole moments. This function is faster than `update_parameters`, so use this if only the field/dipole is changing.

Parameters:

- `field` (`float list`) – New externally applied field $\mathbf{E}_0$ in units of $\sqrt{E/(a^3\lambda_f)}$, where E is the energy scale of the simulation, a is the particle radius, and λ_f is the fluid conductivity.

`MutualDipole.update_parameters(conductivity, field, fileprefix = "", period = 0, constantdipoleflag = 0)`

Change/update simulation parameters. This recalculates all precomputed quantities, including tables for the real space and wave space sums. This function *must* be called any time the size or shape of the simulation box changes throughout the course of the simulation. This function is slower than `update_field`, so do not use `update_parameters` if only the field/dipole is changing. The first two arguments must be specified, even if they are not changing. The last three arguments revert back to their default values if they are not specified.

Parameters:

- `conductivity` (`float list`) – A list of the new conductivity $\lambda_{p,i}$ of each of the active particles in units of fluid conductivity λ_f. This must be specified even if the conductivity does not change.
- `field` (`float list`) – New externally applied field $\mathbf{E}_0$ in units of $\sqrt{E/a^3\lambda_f}$, where E is the energy scale of the simulation, a is the particle radius, and λ_f is the fluid conductivity. This must be specified even if the field does not change.
- `fileprefix` (`string`) – New prefix for output file names. This reverts back to the default value (`""`) if it is not specified.
- `period` (`int`) – New number of time steps between writing output files. This reverts back to the default value (0) if it is not specified.
- `constantdipoleflag` (`int`) – New indicator for the mutual dipole versus constant dipole models. This reverts back to the default value (0) if it is not specified.

Example Script:

```python
from hoomd_script import *
from hoomd_plugins import MutualDipole

# Other HOOMD-Blue code setting up
# .
# .
# .
# a simulation with N particles

mutdip = MutualDipole.compute.MutualDipole(group=group.all(), conductivity=[float("inf")]*N,
    field=[0.0, 0.0, 1.0], xi=0.5, errortol=1e-3, fileprefix="output", period=100,
    constantdipoleflag=0)

run(1000)

mutdip.update_field(field=[0.0, 0.0, 0.5])

run(1000)
```

Test Calculation

The effective conductivity tensor $\boldsymbol{\lambda}_{\text{eff}}$ for a dispersion of spherical nanoparticles of radius a is

$$\boldsymbol{\lambda}_{\text{eff}} = \lambda_f \mathbf{I} + \frac{3\phi}{4\pi a^3}\mathbf{C}, \tag{1}$$

where λ_f is the fluid conductivity and $\mathbf{C}$ is the capacitance tensor that relates the average particle dipole $\mathbf{S} \equiv \sum_i \mathbf{S}_i/N$, where $\mathbf{S}_i$ is the dipole moment of particle i, to the external field $\mathbf{E}_0$ according to the linear relation $\mathbf{S} = \mathbf{C} \cdot \mathbf{E}_0$. For isotropic configurations, $\boldsymbol{\lambda}_{\text{eff}}$ and $\mathbf{C}$ can be represented with scalars, $\boldsymbol{\lambda}_{\text{eff}} = \lambda_{\text{eff}}\mathbf{I}$ and $\mathbf{C} = C\mathbf{I}$, where

$$\lambda_{\text{eff}} = \lambda_f + \frac{3\phi C}{4\pi a^3}. \tag{2}$$

For simple, isotropic lattices (simple cubic, body-centered-cubic, face-centered-cubic, hexagonally-closed-packed) of mutually polarizable dipoles, the capacitance is known analytically and the effective conductivity is

$$\lambda_{\text{eff}} = \lambda_f + \frac{3\lambda_f \beta\phi}{1-\beta\phi}, \tag{3}$$

where $\beta \equiv (\lambda_p/\lambda_f - 1)/(\lambda_p/\lambda_f + 2)$ and λ_p is the particle conductivity.

A script `run.py` is provided that runs a HOOMD simulation to compute the particle dipole moments $\mathbf{S}_i$ for a simple cubic lattice of $N = 125000$ particles. In the simulation, dimensionless quantities are used, $\widetilde{\mathbf{E}}_0 \equiv \mathbf{E}_0\sqrt{E/a^3\lambda_f}$, $\widetilde{\mathbf{S}}_i \equiv \mathbf{S}_i\sqrt{Ea^3\lambda_f}$, and $\widetilde{\mathbf{C}} \equiv \mathbf{C}a^3\lambda_f$, where E is the energy scale (often the thermal energy k_BT). For a unit dimensionless field, $\widetilde{E}_0 \equiv |\widetilde{\mathbf{E}}_0| = 1$, the dimensionless capacitance is equal to the strength of the average dimensionless dipole, $\widetilde{C} = \widetilde{S}$, where $\widetilde{S} \equiv |\widetilde{\mathbf{S}}|$. This script can be used to compute the effective conductivity of a simple cubic lattice at different volume fractions ϕ and particle conductivities λ_p. First, use `run.py` to calculate the particle dipoles for a unit field $\widetilde{E}_0 = 1$. The particle configuration, dipole moments, and forces are output as `.txt` files. Then, average the particle dipole moments to compute the capacitance, $\widetilde{C} = \widetilde{S}$. Finally, substitute $\widetilde{C}$ into equation (2) to obtain λ_{eff}. Figure 1 shows sample results for various particle conductivites and volume fractions.

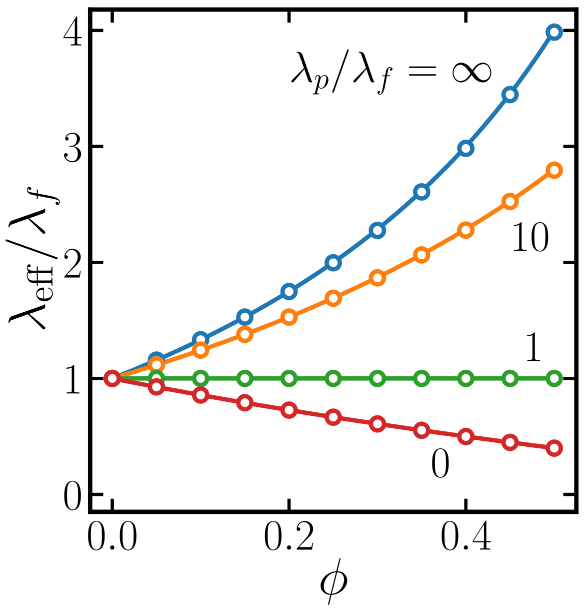

Figure 1: Effective conductivity λ_{eff}, made dimensionless by the fluid conductivity λ_f, of a simple cubic lattice of particles as a function of volume fraction ϕ for various particle conductivities λ_p. The analytical results are shown with lines while the numerical results using the `MutualDipole` plugin are shown with points.