Thermophoresis: A Molecular Dynamics Simulation

Chris Tiede

December 16, 2016

1 Introduction

It is fairly well established that on small scales, different physical phenomena begin to emerge as the ratio of the surface area to volume becomes comparatively large. One such example is that of phoretic motion–the set of processes by which a small particle can effectively "swim" as a result of a field or gradient in some local quantity around it so as to induce fluid motion around the particle surface. However, this gradient in some local quantity or field need not be externally applied, but it can also be generated by the particle itself–a process called auto-phoretic motion. Interestingly enough, there are a number of possible phoretic processes that can generate this kind motion. Some of the primary examples/ideas are particles that through some surface activity generate gradients in the concentration of some dissolved chemical (diffusiophoresis), in the electric potential (electrophoresis), or in temperature (thermophoresis) [1]. At NYU, there is a large experimental project exploring autophoretic swimmers driven by thermophoresis, and there are certain methods that can be used to simulate the motion of these particles [5], however, there is not currently a way to simulate this phenomenon from basic molecular interactions. Therefore, the goal of my project was to write a molecular dynamics code in python that could simulate the thermophoretic motion of one, and eventually many, particles in 3-dimensions with the eventual hope of simulating auto-thermophoretic motion.

Generally, molecular dynamics is a method of studying the time evolution of a large number of interacting particles in a system by solving Newton's equations of motion computationally. Molecular dynamics is a particularly useful tool in a number of fields because the microscopic state of a system is known at every time t, and this allows us the ability to readily examine any number of macroscopic time- and ensemble-averaged quantities arising from the underlying micro-structure of the system. Because of this fact, combined with the fact that thermophoretic flows are complicated and largely inhomogeneous [1], a full molecular model may offer relevant insights into understanding these processes.

2 Molecular Dynamics

2.1 Solving Equations of Motion

Obviously, a good molecular dynamics simulation needs a good method to integrate Newton's equations of motion. It is of primary importance, if one wants to produce results that actually resemble physical systems, that the choice of algorithm conserve energy. One of the most popular symplectic algorithms for molecular dynamics programs is an offshoot of the Verlet algorithm, the Velocity Verlet alogrithm. The Velocity Verlet is essentially the same as the Verlet algorith, except that it uses a calculation of the forces on a particle so as to give the updated position and velocity at the same time step. Specifically, the algorithm goes as follows:

$$\vec{x}(t+h) = \vec{x}(t) + \vec{v}(t)\ h + \frac{1}{2}\vec{a}(t)\ h^2 \tag{1}$$

$$\vec{v}(t+h) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+h)}{2} h\ . \tag{2}$$

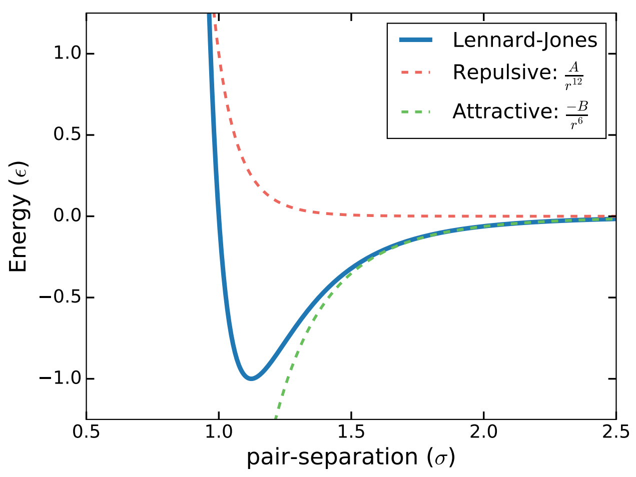
where h is our timestep [3]. As noted, the positions are updated with the old positions, old velocities, and old forces, while the velocities are updated from the old velocities via both the old forces and the forces calculated with the new positions. The algorithm is only second-order accurate, so for short timescales it will not produce trajectories as accurately as may be desired, but more importantly, it does conserve energy, and thus, will produce reliable trajectories over the longer timescales on which molecular dynamics simulations are typically run.

As noted earlier, our Velocity Verlet method depends integrally (see what I did there?? ;)) on the calculation of the force on each particle. Thus, the most fundamental aspect of our molecular dynamics code is how we calculate the forces on each particle. In order to do this, then, we define some interparticle interaction potential. A very common potential for simple interacting particles which I implemented in my simulations is the Lennard-Jones potential, plotted in Figure 1(a), and given by

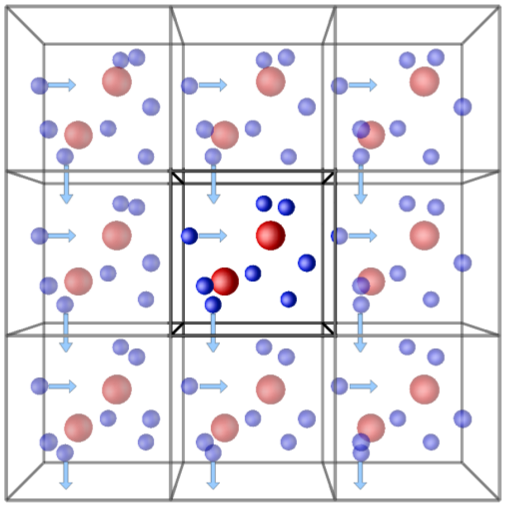
$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{3}$$

where V is the interaction potential between two particles separated by distance r with energy parameter ϵ and molecular diameter σ. The r^{-12} term acts as a hard-core repulsion (the pauli repulsion from overlapping electron orbitals) and the r^{-6} acts as a short range attraction calculated from van der Waals forces. The minimum in the potential occurs at $r = 2^{1/6}\sigma$. A final note on the Lennard-Jones potential is that it defines the basic energy and length scales of our problem–thus, all lengths are given in units of σ and all energies in units of ϵ.

Although molecular dynamics allows us to simulate systems made up of many, many particles, we are still unable to simulate a system anywhere near a mol of particles, as would be representative of an actual, real-life system. Therefore, it would not be appropriate to neglect effects due to the boundaries of the system. One method around this, however, is to use periodic boundaries in which a particle that leaves through the right boundary, say, will re-enter through the left boundary–Figure 1(b). In this way, then, we eliminate boundary effects and have created a system in some infinite bulk. Lastly, when calculating our forces, we do so according to the minimum-image convention. This is to say that when calculating the force between particle i and particle j, we use the minimum of the distance calculated entirely within our simulation box and the distance calculated through the relevant boundary(s).



(a)



(b)

Figure 1: **(a)** Plot of the standard Lennard-Jones potential as a function of interparticle separation. The attractive $-r^{-6}$ is calculated from van der Waals forces and the r^{12} term prevents the overlapping of particles. **(b)** Schematic of periodic boundaries understood as creating a matrix of identical systems surrounding our simulation cell such that any time a particle leaves the system, a copy of it enters on the opposite side.

2.2 Ensembles and Observables

We noted earlier that the Velocity Verlet algorithm conserves energy. Therefore, if we initialize our system to some setup, it will have some energy as defined by the sum of the potential energies due to the interaction potential (and any initial kinetic energy), and the system will evolve at constant energy. Thus, in its most primitive form a molecular dynamics simulation mimics a system in the microcanonical, or NVE, ensemble of statistical mechanics. However, the equipartition theorem tell us that a system with s degrees of freedom will have a temperature defined by the average kinetic energy of those particles according to the equation

$$sk_bT = \frac{1}{N}\sum_i mv_i^2 \ . \tag{4}$$

In our primitive NVE model, then, temperature is not fixed and simply varies according to the average kinetic energy of the particles in the system. However, this is not the case for most physical systems as most experiments are in contact with a heat bath and are often conducted at constant temperature. Thus, we need a way to control temperature in our simulations, and the most basic way of accomplishing this is via velocity rescaling. Velocity rescaling is simply calculating the current temperature of a system via the equipartition theorem, T_0, and calculating some scaling factor

$$\alpha = \left(\sqrt{\frac{T}{T_0}}\right)^{\frac{1}{\tau}} \tag{5}$$

where T is the desired temperature, and τ is some length scale over which one wishes/expects to achieve the desired temperature, and multiply all velocities by this factor every timestep, $v_i' = \alpha v_i$, in order to slowly drive or keep the system at this temperature.

It is worth mentioning, however, that because this method simply scales velocities in order to achieve a desired average kinetic energy, it does not actually produce a Boltzmann distribution of velocities, and thus, does not accurately reflect the canonical ensemble. Though beyond the scope of this project, there do exist a number of methods, usually involving some kind of modified Lagrangian, that can accurately produce the canonical ensemble.

Another quantity of interest in molecular dynamics simulations is often the diffusion of particles in the system. One method for measuring the diffusion coefficients of different types of particles in our system is to track the Mean-Squared Displacement of the particles. The mean squared displacement is the average of the square of the displacement of a set of particles from their original position, and is give by

$$\mathrm{MSD} = \langle \Delta x^2 \rangle = \frac{1}{N}\sum_i (\vec{x_n}(t) - \vec{x_n}(t_0))^2 \ . \tag{6}$$

Einstein tells us, then, that for particles interacting thermally, their diffusion coefficient is given by a linear relationship between the MSD and time,

$$\langle \Delta x^2 \rangle = 2dD\Delta t \tag{7}$$

where d is the dimensionality of the system and D is the diffusion coefficient. In my simulations, then, I added a bigger particle of about twenty times the mass of the surrounding particles, and expected it to behave in a Brownian fashion, so I tracked its mean squared displacement.

3 Computational Methods

3.1 Initialization

Before one can run a molecular dynamics simulation, one needs to define the initial positions and velocities of every particle in the system. In order to do this, I wrote two functions to initialize the positions of the particle in my system. The first of these inserted particles into the box randomly, while checking that none of the particles overlap; and in the second, I placed each of the particles in a cubic lattice with a the maximally allowed characteristic length given the box size and number of particles in the system. For my simulations, I always used a cubic simulation cell of side length either $L = 6or8(\sigma)$. For particle number, I used up to 500 particles, and usually chose my particle number such that the volume fraction of my system ($\phi = \pi N/6V$) was between $\phi = 0.4$ and $\phi = 0.5$. Specific box size, particle number, and volume fractions will be indicated when relevant in Results.

With regards to velocities, I initialized the system by randomly assigning velocities from the Boltzmann Distribution, and then I subtracted any center of mass velocity so as to avoid accidentally adding any drift to my system. An example of an initial velocity distribution for $T = 1.0(\epsilon/k_B)$ is shown in Figure 3.

3.2 Force Calculation

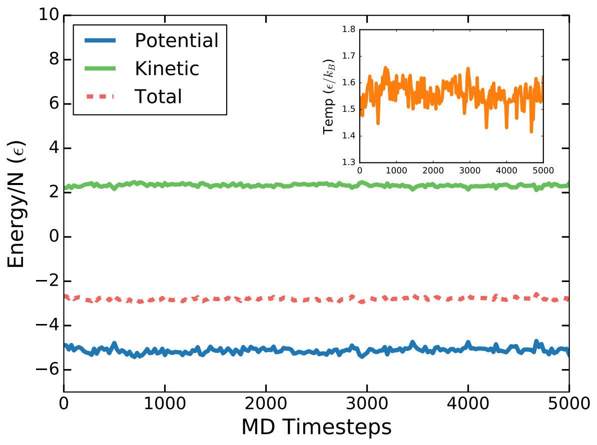
One of the core elements of a molecular dynamics code, as mentioned earlier, is calculating the forces on each particle due to every other particle in the system. In my code's final form for this project, I accomplished this by keeping an $N \times 3$ 2D-array which stored the Cartesian coordinates for every particle in the simulation. In order to calculate the force, then, I stored the distance between each particle in each Cartesian direction in a 3D-array, and then used this array to calculate the magnitude squared of every distance, the magnitude of every distance, and the unit "vector" associated with the matrix of particle separations. These three matrices are then used to calculate the force on each particle due to ever other particle. It is worth mentioning that this was computationally much more efficient than looping through every particle and calculating the force on it due to every other particle, but the building and manipulation of these matrices still makes up the majority of my computational time. In the future, then, because I am using a short-range potential, I would hope to keep a neighbor list for each particle and then use only these neighbor lists to calculate the forces in order to further speed up my code and allow for the inclusion of even more particles.

Lastly, it is worth mentioning here, that in order to keep these forces from 'exploding' due to inadvertent particle overlap, it is important to choose a timestep that is substantially small as compared to some characteristic time for our system. In Lennard-Jones units, the characteristic time for a simulation is given by $\sqrt{\sigma^2 m/\epsilon}$. In my simulations, then, I used a timestep $h = 0.1$ times this characteristic time.

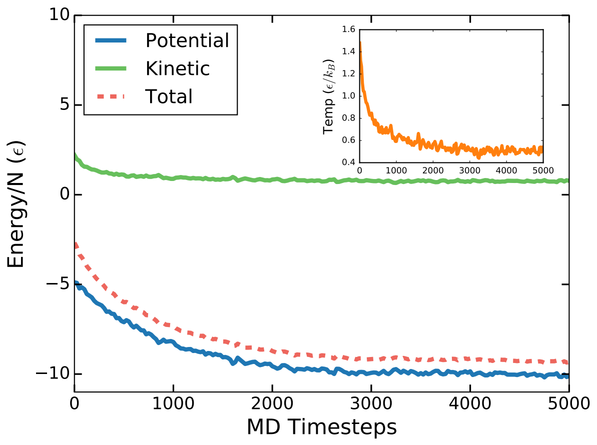
4 Results

The first check to make sure that a molecular dynamics code is running properly is to make sure that it conserves energy. For this I ran simulation in the NVE ensemble with box length $L = 8\sigma$ with 200 particles ($\phi = 0.20$) with initial temperature $T = 1.5$ in reduced LJ units. Plots of kinetic energy, potential energy, and total energy are given in Figure 2(a). As we can see, there are noticeable fluctuations in the potential and kinetic energies–as would be expected–but the total energy stays constant. One may notice, however, that there are small fluctuations in the total energy, but this is likely due to the relative lack of precision in the Verlet method over small time periods. What is most important is that the method does not demonstrate any discernible drift in the energy over longer timescales. We notice though, as expected since we are operating in the NVE ensemble, that the temperature is not conserved (the inset) as the kinetic energy fluctuates in the constant give and take with the potentials between the particles. In Figure 2(b), though, in order to test our thermostatting, I plot the energies and the temperature in a run with the same initial configurations as in the NVE ensemble, except in this run, the temperature of the system is scaled to reduced temperature $T = 0.5$ and the time scale τ is set to $\tau = 200$ steps. We notice here, that the energy is not conserved, and this is precisely because by rescaling the velocities we are "injecting"–or in this case, rather, stealing–energy from the system as we drive it towards our desired temperature. Looking at the temperature, we see that it does take some time for the temperature to reach its desired value, but that after that, it remains relatively constant (it could be force to be exactly constant by decreasing our length scale parameter τ in the velocity rescaling, but too much velocity rescaling would completely mask any physical meaning in our system).

Once I verified that my molecular dynamics code was working as desired, the first step towards trying to getting a colloid to swim in a temperature gradient is actually implementing the larger particle itself. In order to do this, I also defined two particle types and assigned the second type of particle a mass of twenty times those of the smaller type. When running these simulations, then, the larger particle ought to exhibit Brownian behavior, so I examined its Mean-Squared Displacement as a function of run time. In these simulations, then, what we should expect to see–for both the larger colloid and the bath of surrounding particles–is that at short times they are behaving ballistically. That is to say that they are inertial particles that are not interacting frequently enough so as to exchange thermal energy, and thus simply move according to the kinematic equations with a mean-squared displacement depending quadratically on time. However, after some short time, the particles ought to start interacting more and more frequently so as to exchange thermal energy, and in this regime–as they move towards thermal equilibrium–the particles' MSD ought to



(a)



(b)

Figure 2: **(a)** Plots of the energy and temperature (inset) in the NVE ensemble as functions of MD time for system with 200 particles with intial $T = 1.5$. **(b)** Plots of the energy and temperature in the NVT ensemble via velocity rescaling as functions of MD time for system with 200 particles, initial $T = 1.5$, and desired $T = 0.5$.

display a linear dependence on time–as discussed earlier. In order to test, this, I simulated 200 particles in a box with side length $L = 6$ ($\phi = 0.5$) over 5000 timesteps. In Figure 3 I have plotted the average MSD of particles in the surrounding bulk as well as that of the colloidal particle averaged over 5 runs, and one can see that in both cases, the particles initially exhibit ballistic (or quasi-ballistic in the case of the colloid) behavior, but that after some equilibration time they each transition into the linear, thermal regime. We also notice that the particles thermalize rather quickly and that it take the colloid about 3 or 4 times as long to equilibrate with the rest of the system, as we would expect.

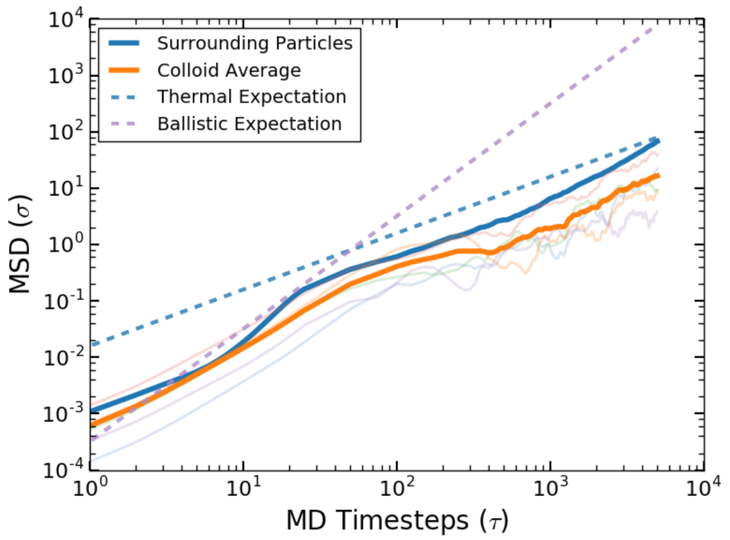


Figure 3: The Mean Squared Displacement of both the colloidal particle and surrounding smaller particles. We see that both type of particle initially behave ballistically, but that they each eventually transitino into the thermal regime with a linear dependence of MSD on time. The MSD of the colloidal particle was averaged over 5 separate and independent runs with each individual run plotted faintly behind the average.

Thus, having shown that the colloidal particle when added to the molecular dynamics simulation behaves in a physically relevant way–namely that it obeys Einsteins diffusion law–the next step towards getting the colloid to swim in a temperature gradient is putting a temperature gradient across the MD simulation. In order to make sure that the temperature gradient is functioning properly, we would expect to see a redistribution of the density of particles in the system according to the temperature gradient; namely the colder regions ought to become more dense than the hotter regions, because particles in the hotter regions have a higher average kinetic energy and speed, and are therefore more likely to leave their region, while particles in the colder region are moving comparatively slower, and are thus less likely to leave their region. One important note when testing the temperature gradient, though, is that because I am using periodic boundary conditions, it is important that the temperature gradient also be periodic and symmetric with respect to the boundary. However, in my initial test case, I simply split my system in half along the z-direction, where half of the system was hotter than the other with a sharp temperature discontinuity in the middle, and therefore, the periodicity across multiple iterations of the system is preserved as the same discontinuity in the center also exists at each of the boundaries. Specifically, then, I simulated a fairly dilute system of 200 particles with no colloidal particle in a box with side length $L = 8$ ($\phi = 0.2$) where the whole system was equilibrated over 2500 steps to a temperature of $T = 1.0$, and then for the next 5500 steps I heated the the system with z-coordinate $z > 4.0$ up to a temperature of $T = 1.5$, while keeping the region $z < 4.0$ at $T = 1$. I chose to use this relatively dilute system because each particle has diameter $\sigma = 1.0$, and thus, because of the LJ potential's hard-core repulsion, too dense of a system would not allow for adequate density rearrangements. In these simulations with the basic temperature split implemented by velocity rescaling, I was able to achieve a noticeable redistribution of density in accordance with expectation as presented in Figure 4.

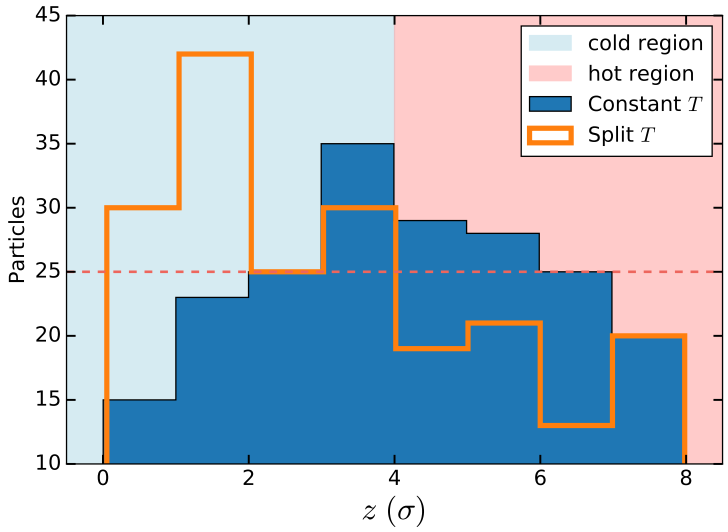


Figure 4: The number of particles in each of 8 regions along the z-direction. The colder and hotter regions are indicated, and we can see that, as expected, the colder region has become more dense and has more particles than the hotter region. The red line indicates a perfetly diffuse system, and if we averaged the case at constant temperature over many iterations, this is what we would expect to see.

In order to actually add a colloid to a temperature gradient and to see it swim, though, I will need a much gradual temperature gradient with a ladder of temperature levels that still maintains symmetry and periodicity across our system and the lattice of systems respectively. I did in fact implement a gradient with 8 separate regions in the z-direction where the middle two were the hottest ($T = 2.0$), and then it stepped down symmetrically in both directions down to the edges with $T = 1.0$, and I added the colloid to this system. However, in my initial runs under these conditions, while it appears that the colloid may perhaps be biased to move in the z-direction along the gradient, I have not been able to generate anything conclusive enough to show that this is not just my bias looking for the result I desire.

5 Conclusions and Future Work

Overall, in this project, I built a molecular dynamics code from scratch that operated according to the Lennard-Jones potential and the Velocity Verlet algorithm for integrating Newton's equations of motion. Furthermore, I was able to successfully add a colloidal particle to the system and verify that it behaved as expected via calculations of the Mean Squared Displacement, and I implemented temperature controls via the velocity rescaling method in order to place a temperature gradient across the whole system and observe a redistribution of particle density as a result

Extensions of this work will continue trying to place the colloidal particle in a more continuous temperature gradient in order to observe it "swimming" towards the cold region, and after this the task will become how to place the temperature gradient across the particle itself so that the particle is itself generating the local temperature gradient that ought to drive its motion. With regards to the molecular dynamics code itself, now that I have the basic implementation, I hope to build it out and make it more robust by including other functionalities. One functionality that I particularly hope to include is a neighbor list so as to dramatically increase the run speed of my program.

References

[1] R. Golestanian, T.B. Liverpool, and A. Ajdari, "Designing phoretic micro- and nano-swimmers," New J. Phys. 9 (2007) 126.

[2] Han, Minsub, "Thermophoresis in liquids: a molecular dynamics simulation study," Journal of Colloid and Interface Science, 284 (2005) 339.

[3] D. Frenkel and B. Smit, Understanding Molecular Simulation: From Algorithms to Applications, Academic Press Ince, San Diego, CA, 1996.

[4] D.C. Rapaport, Art of Molecular Dynamics Simulation, Cambridge Univ. Press, Cambridge, UK, 1995.

[5] D. Grier, "Trochoidal Trajectories of Self-Propelled Janus Particles in a Divergent Laser".