Advanced molecular dynamics modeling

Andreas V. Solbrå
University of Oslo
Department of Computational Physics

April 3, 2013

Abstract

The goal of this project is to gain experience in more advanced modeling in molecular dynamics. It will build directly on the previous project.

In the previous project we made a simple model for an Argon gas. In this project, we are interested in using and expanding on our model to look at some more interesting system. We want to create much larger systems than earlier, and the first thing we should do is to optimize and parallelize our code

1 Optimization

1.1 Parallelization using OpenMP

OpenMP is a wonderful "plug and play" parallelization API for situations where all your parallel threads share memory. Say you want have a loop that sums the first N terms in the harmonic series. In order to parallelize this, all you need to do is to go from

```
double sum = 0;

for(n=1;n<=N;n++){
  sum += 1./n;
  }
```

to

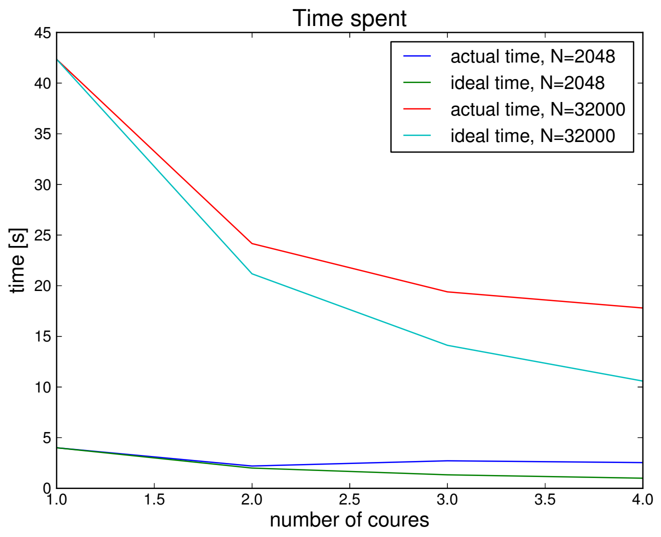


Figure 1: The figure shows the time spent for $N = 2048$ and $N = 32000$ particles as a function of cpu cores when using openMP, for a test simulation of 100 time steps.

```
double sum = 0;
#pragma omp parallel{
  double loc_sum = 0;
  #pragma omp for
  for(int n=1;n<=N;n++){
    loc_sum += 1./n;
  }
  #pragma omp critical{
    sum += loc_sum;
  }
}
```

The variable `loc_sum` is created privately for each thread as it is inside the parallel scope defined by the first pragma statement. The critical statement ensures that the different threads write to the global variable `sum` one at a time. If we know that one part of our code stands out as particularly time-consuming, it is often well worth while to try to parallelize it using openMP at first, as the implementation is so simple. In our case, the time-consuming part is the calculation of the forces, as this loops over many different particle pairs. We can test our code for different numbers of particles and processors. The result is shown i figure 1. We see a clear improvement when going from 1 to 2 processors, but after this there is almost no speedup. For smaller systems the computation time even increases! This is most likely due to overhead from adding the forces matrices together.

1.2 Optimizing the code for efficiency

We should note that we can also gain a lot from just writting our code in a more optimized way. In the previous project we used vectors and matrices from the armadillo c++ library.

This has the advantage of producing cleaner code, but it comes at the price of efficiency. The most important point is to stay away from armadillo functions such as `dot` and `norm`. These are terribly inefficient, and will often take up to 10 times as much time as a hardcoded version. Furthermore, armadillo vectors have overloaded +/- operators, that are quite slow compared to writing the additions out element by element. Using these changes gave an overall speedup factor of around 10, and was absolutely essential in order to make the code usable for large scale experiments.

2 Generating a nanoporous material

We will now look into ways of generating nanoporous matrial. As the c++ code can continue a simulation from an .xyz-file, we can thermalize our system in one simulation, do any manipulation in Python for simplicity and continue the simulation from our manipulated .xyz-file. Including the possibility of starting from a given .xyz-file in our program is very usefull, as it gives us an easy way to change parameters in the experiment such as thermostats and, as in this case, freeze some of the particles.

As a simple test, we can thermalize a system of liquid Argon in a $20 \times 20 \times 20$ cell with cell size $b = 5.72$ ÃĚ at $T = 0.851$, cut out a cylinder of 2 nm, and freeze all the particles outside this cylinder. The result of such an experiment is shown in figure 2

2.1 Model for nanopourous material

There are several different ways of generating a nanoporous material. One can for instance slowly expand the size of the size of our system while cooling it. Here we will look at a much simpler method. We simply make a a collection of spheres with radii in the range 2-3 nm, and freeze either the particles inside the spheres, or the particles outside. For our system, we will make 20 spheres and freeze the paricles outside the spheres. It is useful to find the porosity of such a system, that is, the relative amount of pore space in the volume. An easy way to do this is to assume that each particle takes up the same amount of space, and simply count the particles which are not frozen by this procedure. Since we allow the spheres to overlap, and be partially outside the system, it will be troublesome to find an analytic expression for this ratio, and we rather find it numerically through several experiments the results of these experiments are shown i figure 3, and gives an average porosity around 63% . We note that each sphere has an average volume of $4\pi/3\langle r^3\rangle \approx 68\text{nm}^2$ so that 20 non-overlapping spheres all inside the box of sides 20×5.72nm would yield an average porosity of 91%.

We can now visualize our system. Figure 4 shows the result of such and experiment, where the system has been thermalized at $T = 1.05$ after the particles outside the spheres.

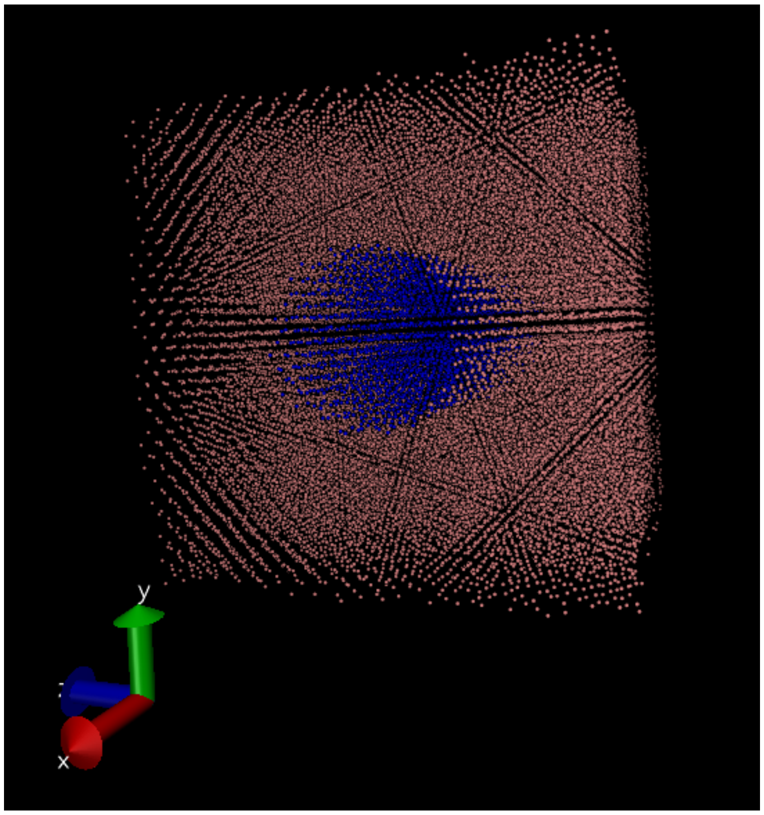


Figure 2: The figure shows a system with a cylinder of size 2 nm

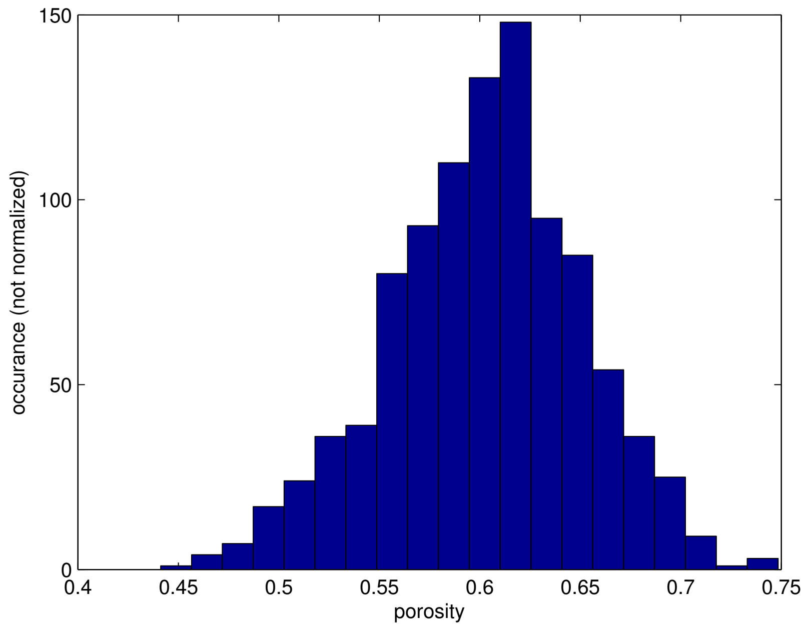


Figure 3: The figure shows the porosity distribution of a system of 20 spheres.

2.2 Low-density fluid in a nanoporous material

We now want to model the flow a low density through the fluid we have created. This is a task well suited for our Python framework. We simply loop over all the unfrozen atoms and set a 50% chance of deleting them. This can be found in the python function `removeSomeUnFrozenAtoms(self,eps)`. We can messure the pressure of the system, as well as the pressure distribution in space. One way to do the latter is to calculate the pressure for each cell, and then plot these somehow. In figure 5, they have been plotted in ovito coloring the cell dots by amount of pressure

3 Diffusion in a nanopourous material

Figure 7 shows the average square displacement of the low-density fluid.

4 Flow through a nanoporous material

We can induce a flow in the material by introducing an external force, $\mathbf{F} = F_x\mathbf{i}$ acting on each atom - similar to gravity. In the case of flow in a gravitational field, Darcy's law is

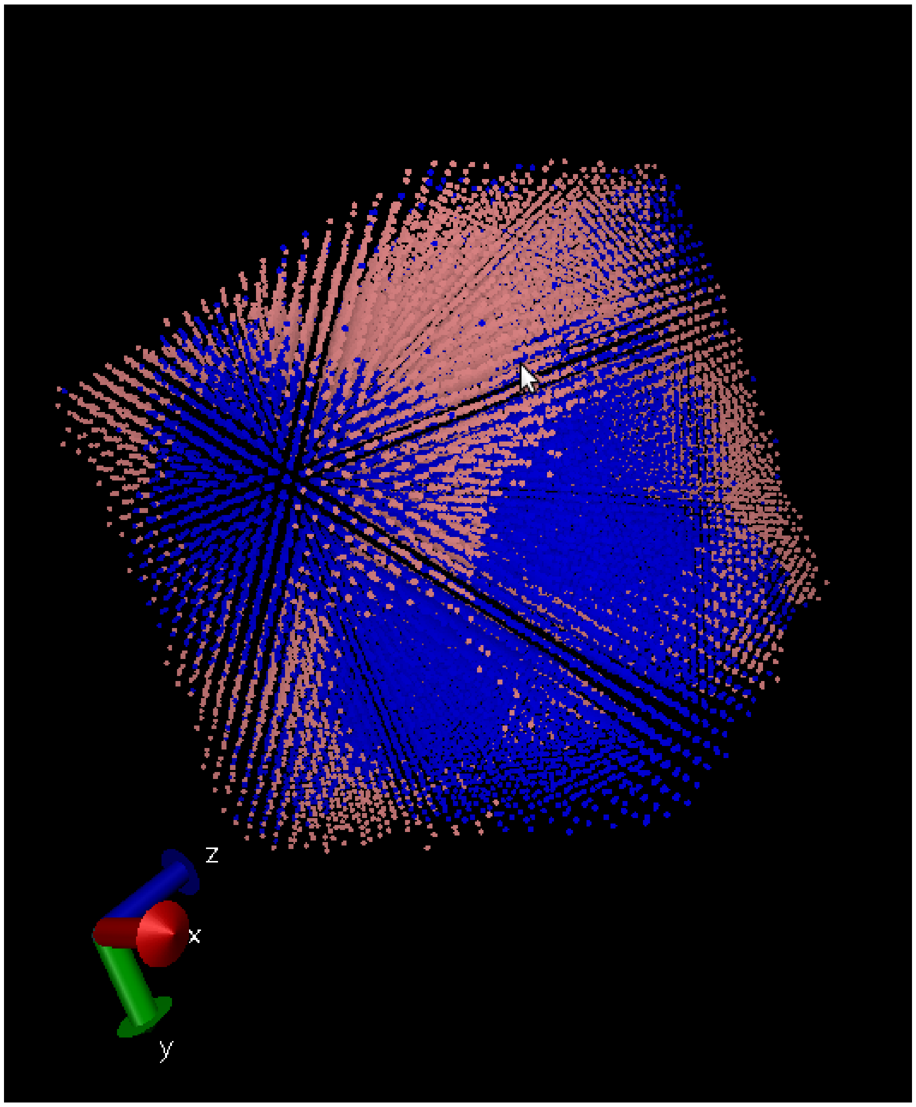


Figure 4: The figure shows a nanoporous material, created as discussed in the text.

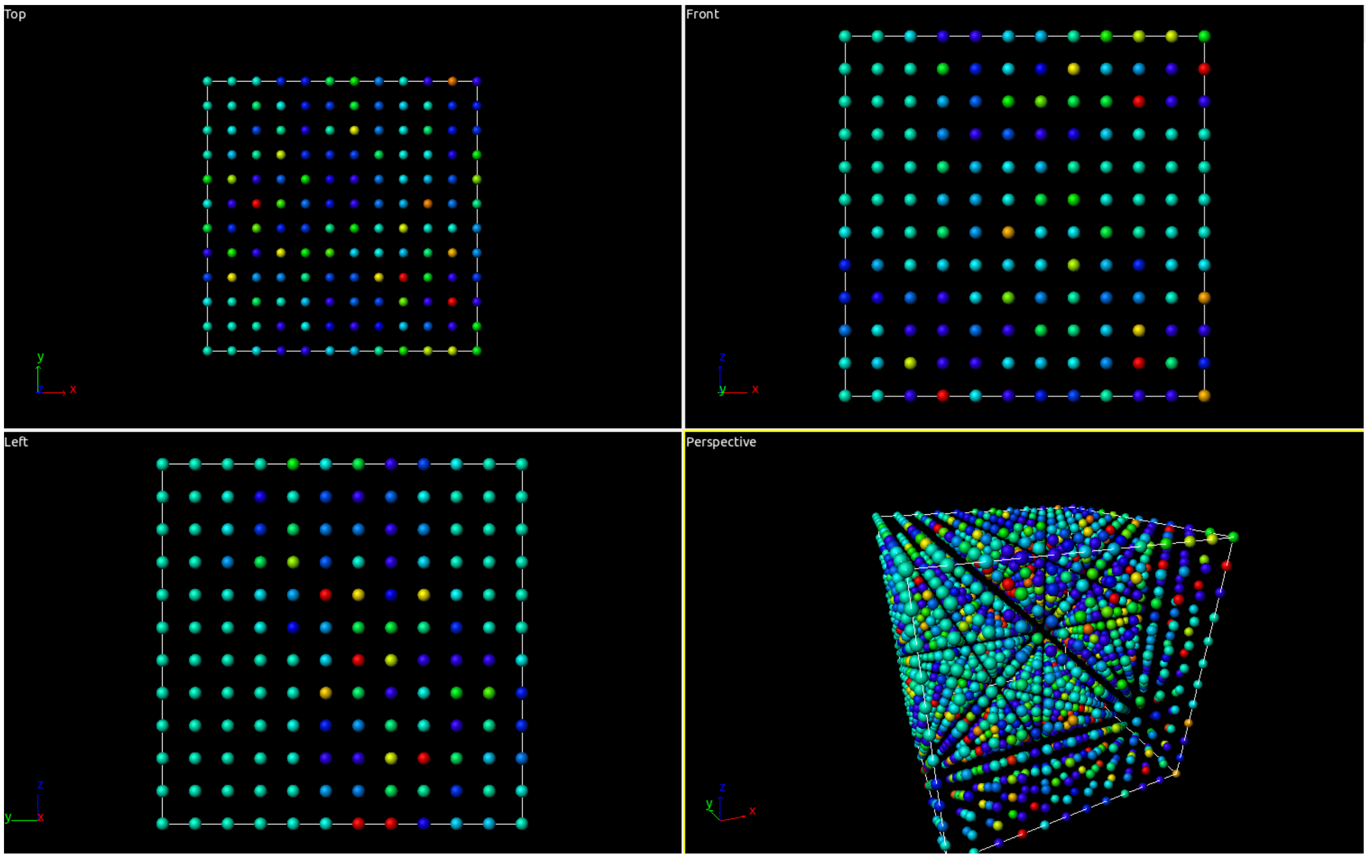


Figure 5: The figure the pressure distribtion of the system. The pressures are in the range -50 to 100 [MD units], with red denoting the highest pressures.

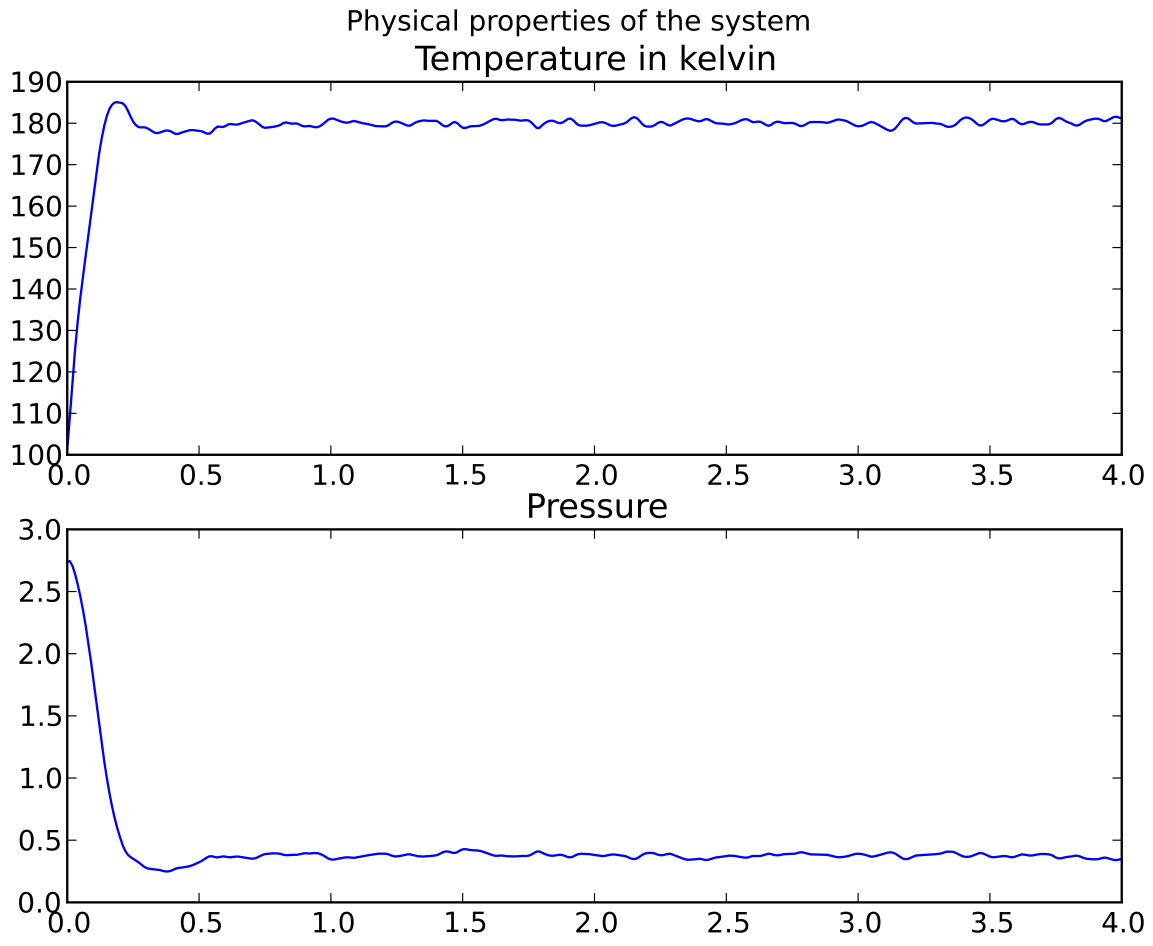


Figure 6: The figure shows the pressure of the low density fluid as a function of time. Both pressure and time are in MD units.

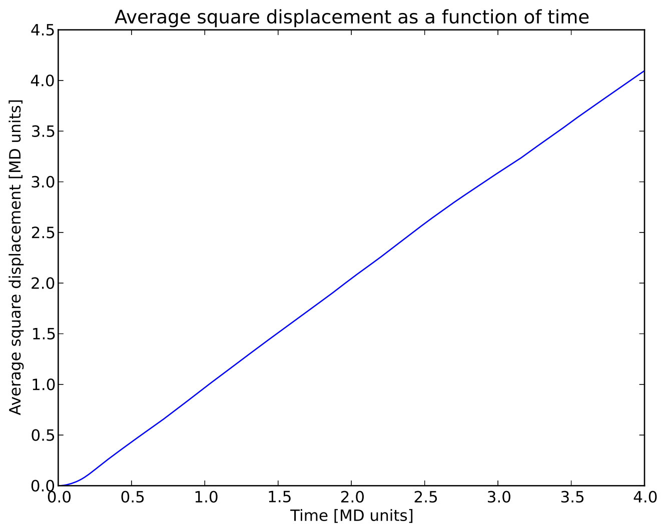


Figure 7: The figure shows the average square displacement of the low-density fluid when thermalized at $T = 1.5$.

usually formulated as

$$U = \frac{k}{\mu}(\nabla P - \rho g) \tag{1}$$

For a system like flow thorugh a cylinder, we can ignore the ∇P-term and note that $\rho g = Nmg/V = nF$, where $n = N/V$ is the number density of particles and $F = mg$ is the force exerted by the gravitational potential.

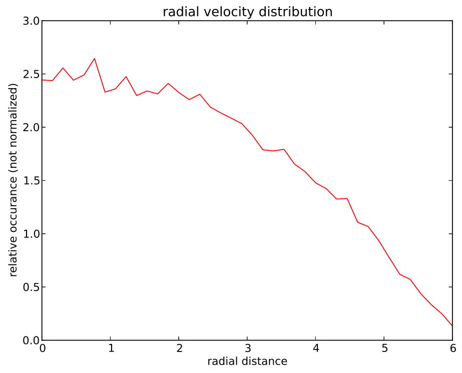
It is interesting to measure the flow profile of our liquid and cylinder system. From continuum models, the flow profile should be $u(r) \propto a^2 - r^2$ where a is the radius of the cylinder, and the density profile should be constant. Figure 8 shows these properties for a normal density fluid, while 9 show these for a low density fluid, where we have set $F = 1.0\epsilon/\sigma$. The results seem to be in good agreement with the theory, with the note that for high density liquids, there will be some structure in the density due to the particle interactions. We can use these results to measure the viscosity of the fluid, as for a cylindrical pipe, the velocity profile can be shown to be

$$u(r) = \frac{\Delta P}{L}\frac{1}{4\mu}(a^2 - r^2) \tag{2}$$

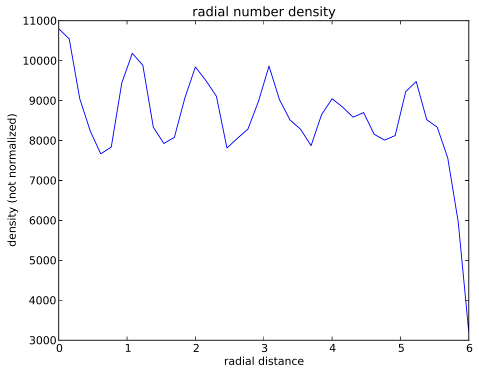
where ΔP is the pressure difference in the pipe, L is the length of the pipe, μ is the viscosity of the fluid, a is the radius of the pipe and r is the distance from the radius. Using $u(0) = 2.5\sigma/\tau$ for the high density fluid and $u(0) = 3.5\sigma/\tau$ for the low density fluid, and in our MD units, $a = 5.9\sigma$. The pressure difference is simply $\Delta P = \rho g \Delta h = nFL = NFL/V$ inside the cylinder where N is the total number of particles, V is the volume of the cyllinder. The number density is $n = 0.84\sigma^{-3}$ for the high density and $n = 0.42\sigma^{-3}$ for the low density.

$$\mu = \frac{nFa^2}{4u(0)} \tag{3}$$

which gives $\mu = 11.7\frac{m}{\sigma\tau}$ for the high density fluid and $\mu = 5.8\frac{m}{\sigma\tau}$ for the low density fluid, where m is the mass of the Argon atom (i.e. our unit of mass).

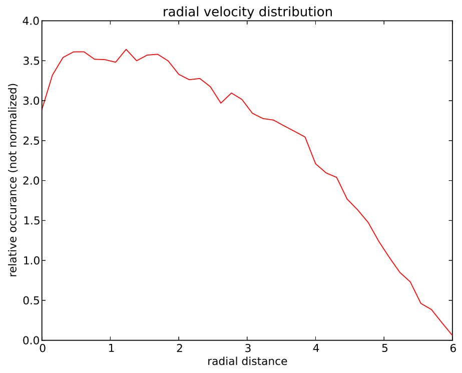


(a) velocity distribution

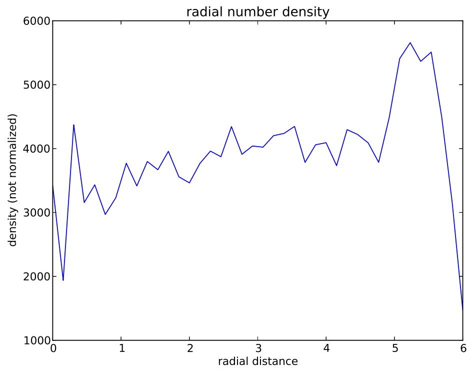


(b) density distribution

Figure 8: The plots show the distribution of the velocity and the distribution of particles in a cylinder filled with a dense liquid subject to a constant force at 500 K.



(a) velocity distribution



(b) density distribution

Figure 9: The plots show the distribution of the velocity and the distribution of particles in a cylinder filled with a low-density liquid subject to a constant force at 500 K.

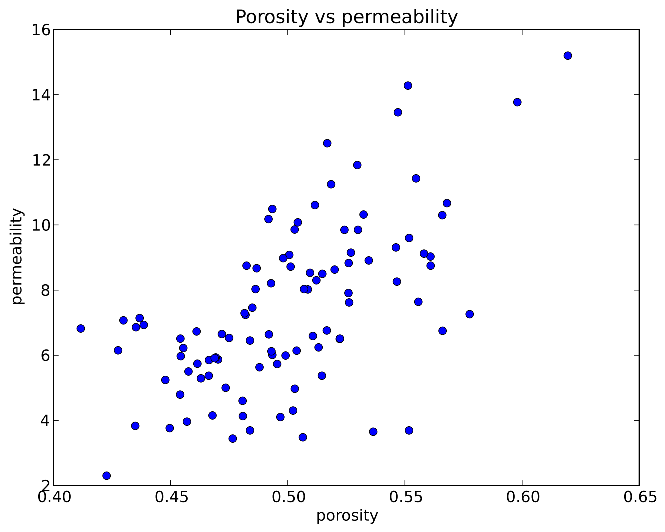


Figure 10: Permeability and porosity

5 Permeability

Finally, we will investigate the permeability of our nanoporous materials with varying porosities. To do this. we simply generate a large collection of systems to simulate with different porosities. We can find the permeability from Darcy's law;

$$Q = \frac{kA}{\mu}\frac{\Delta P}{L}. \tag{4}$$

The results of many experiments is shown in figure 10