Molecular-Dynamics

May 14, 2020

1 Molecular Dynamics Simulation

1.1 Introduction

In this project, we simulate a solid-liquid phase transition of Argon using a simplified Lennard-Jones model. Each Ar atom is assumed to have no internal degrees of freedom, but is treated a pointlike particle interacting with the other particles in the simulation according to the Lennard-Jones "6-12" potential

$$U_{ij} = 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right]. \tag{1}$$

The simulation uses dimensionless parameters in which all fundamental physical constants have been absorbed. Quantities are measured in terms of the following:

$$\text{Length} : \sigma = 3.405 \times 10^{-10}\,\text{m}, \qquad \text{Time} : \tau = \sqrt{\frac{m_{\text{Ar}}\sigma^2}{\epsilon}} = 2.15 \times 10^{-12}\,\text{s}, \tag{2}$$

$$\text{Energy} : \epsilon = 1.65 \times 10^{-21}\,\text{J}, \qquad \text{Temperature} : \frac{\epsilon}{k_{\text{B}}} = 120\,\text{K}, \tag{3}$$

$$\text{Mass} : m_{\text{Ar}} = 6.63 \times 10^{-26}\,\text{kg}, \qquad \text{Entropy} : k_{\text{B}} = 1.38 \times 10^{-23}\,\text{J/K}, \tag{4}$$

and etc. for derived quantities such as velocities and forces. The data files output by `md-simulate` are all in dimensionless units, so they can be rescaled with the above numbers, or with some other set of numbers in order to represent any material that can be well-approximated with the Lennard-Jones potential. Note that the *time step* used in the simulation (and output in the `time_series.csv` files) is 0.005τ, so that 200 time steps make up one unit of time.

1.2 Setup

[16]:
```
%matplotlib inline
import os
import pandas as pd
import numpy as np
from mpl_toolkits import mplot3d
pd.plotting.register_matplotlib_converters()
```

```
import matplotlib.pyplot as plt
import seaborn as sns

dataset_path = 'data_cellcount_5'

density_labels = [d for d in os.listdir(dataset_path) if os.path.isdir(os.path.
 ↪join(dataset_path, d))]
print(density_labels)

temperature_labels = [t for t in os.listdir(os.path.join(dataset_path,␣
 ↪density_labels[0]))
                      if os.path.isdir(os.path.join(dataset_path,␣
 ↪density_labels[0], t))]
print(temperature_labels)
```

```
['rho_0.500', 'rho_0.550', 'rho_0.600', 'rho_0.650', 'rho_0.700', 'rho_0.750',
'rho_0.800', 'rho_0.850', 'rho_0.900']
['T_0.100', 'T_0.150', 'T_0.200', 'T_0.250', 'T_0.300', 'T_0.350', 'T_0.400',
'T_0.450', 'T_0.500', 'T_0.550', 'T_0.600', 'T_0.650', 'T_0.700', 'T_0.750',
'T_0.800', 'T_0.850', 'T_0.900']
```

Load the data files:

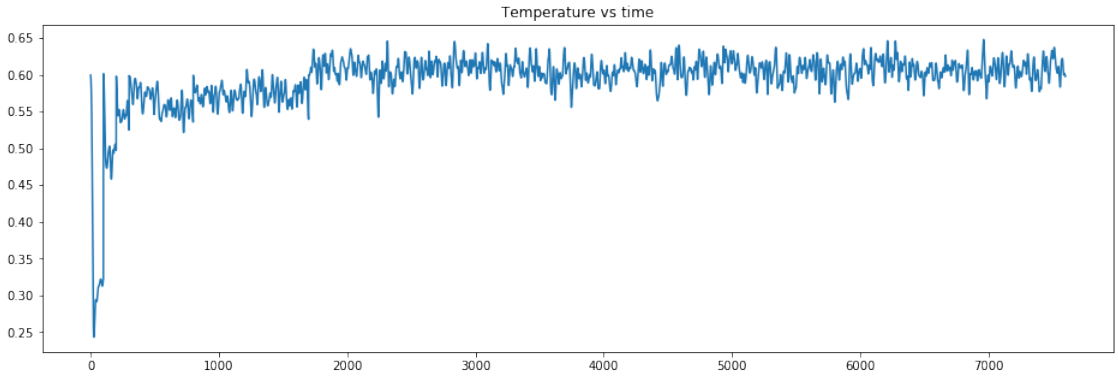
[17]:
```
thermo_meas_data = pd.read_csv(os.path.join(dataset_path, 'thermo_measurements.
 ↪csv'))

final_state_data = {d : {} for d in density_labels}
summary_info_data = {d : {} for d in density_labels}
time_series_data = {d : {} for d in density_labels}

for d in density_labels:
    for t in temperature_labels:
        final_state_data[d][t] = pd.read_csv(os.path.join(dataset_path, d, t,␣
 ↪'final_state.csv'))
        summary_info_data[d][t] = pd.read_csv(os.path.join(dataset_path, d, t,␣
 ↪'summary_info.csv'))
        time_series_data[d][t] = pd.read_csv(os.path.join(dataset_path, d, t,␣
 ↪'time_series.csv'),
                                             index_col='TimeStep')
```

1.3 Walking through a simulation

We start by giving an example of what a simulation looks like. We fix the initial positions and velocities of the particles, and let them go. We are attempting to achieve a specific temperature of the system *in equilibrium*, but our system does not *start* in equilibrium. We must let the system relax for a bit, and if it is still the wrong temperature, we need to adjust it (by scaling the velocities), and then let it relax again. Eventually, when the temperature fluctuations stay within certain bounds, we declare the system to be at equilibrium, and we can begin doing measurements

on the system.

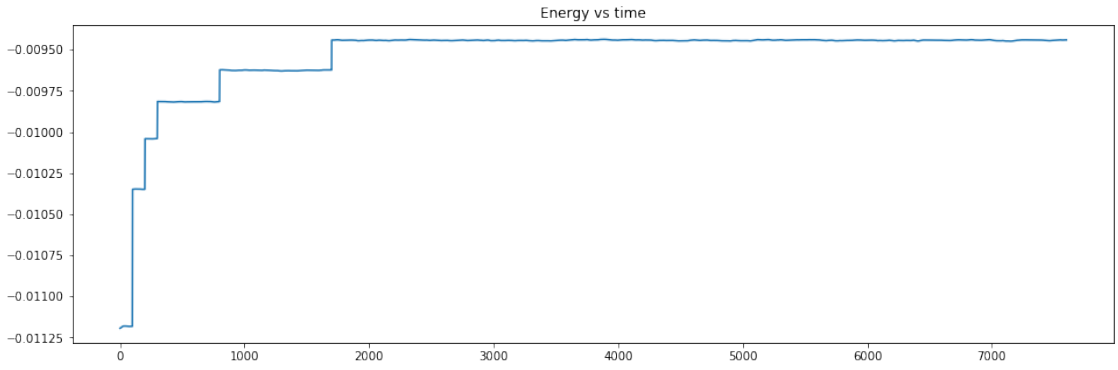
[21]:
```
plt.figure(figsize=(16,5))
plt.title("Temperature vs time")

sns.lineplot(data=time_series_data['rho_0.800']['T_0.600']['Temp'])
plt.show()
```



In this graph of temperature vs time, you can see this process of a series of "kicks" on the system until it finally settles to a roughly-constant temperature, near the nominal temperature of $0.6\,\epsilon/k_{\mathrm{B}}$.

One useful diagnostic of our integration algorithm is to plot the total energy over time. Since the system is closed, the total energy should be constant. This is achieved by using a symplectic integration method (in this case, velocity-Verlet integration).

[20]:
```
plt.figure(figsize=(16,5))
plt.title("Energy vs time")

sns.lineplot(data=time_series_data['rho_0.800']['T_0.600']['TotEnergy'])
plt.show()
```



You can see the initial adjustment "kicks" quite clearly in this graph. After the last kick, the

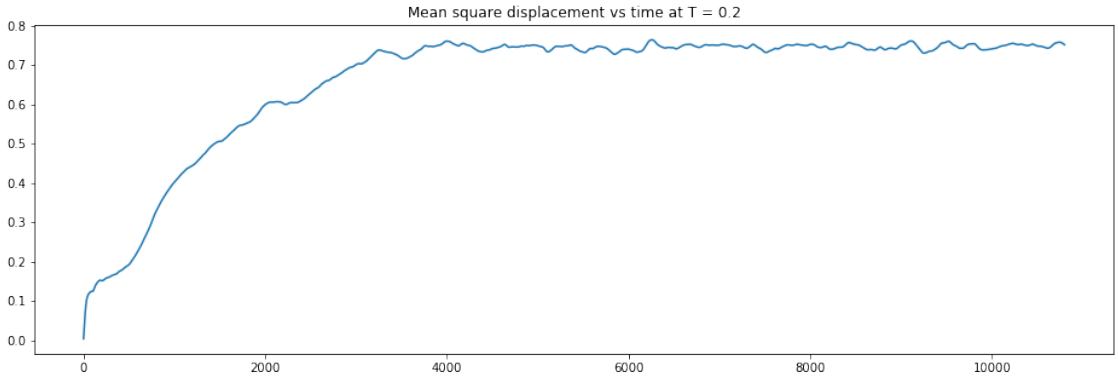
energy remains very nearly constant, which indicates the integration algorithm is at least producing physically correct results.

1.4 Distinguishing solid from liquid

The goal of the simulation is to see whether something like the Lennard-Jones potential gives us complex enough behavior to see phase transitions. A very good indicator of the transition between solid and liquid is the *mean square displacement* of the particles, which measures, effectively, the rate of diffusion in the material. A particle undergoing Brownian motion will tend to wander from its original starting point at a rate proportional to the *square root* of t, and therefore its *square* displacement will tend to grow linearly. The slope is the rate of diffusion.

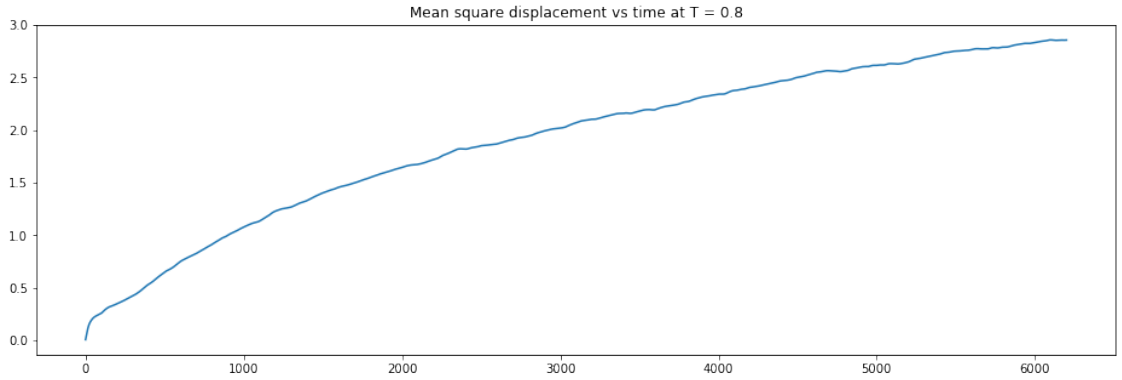
In a solid, the rate of diffusion is practically zero, so we expect the mean square displacement to stay roughly constant, once the system has settled in equilibrium. In a liquid, by contrast, we should see linear growth. We give two examples.

[22]:
```
plt.figure(figsize=(16,5))
plt.title("Mean square displacement vs time at T = 0.2")

sns.lineplot(data=time_series_data['rho_0.800']['T_0.200']['MeanSqDisp'])
plt.show()
```



We see quite clearly that at low temperatures (for $T = 0.2\,\epsilon/k_{\mathrm{B}}$), the mean square displacement is constant at equilibrium, and thus we have a solid.

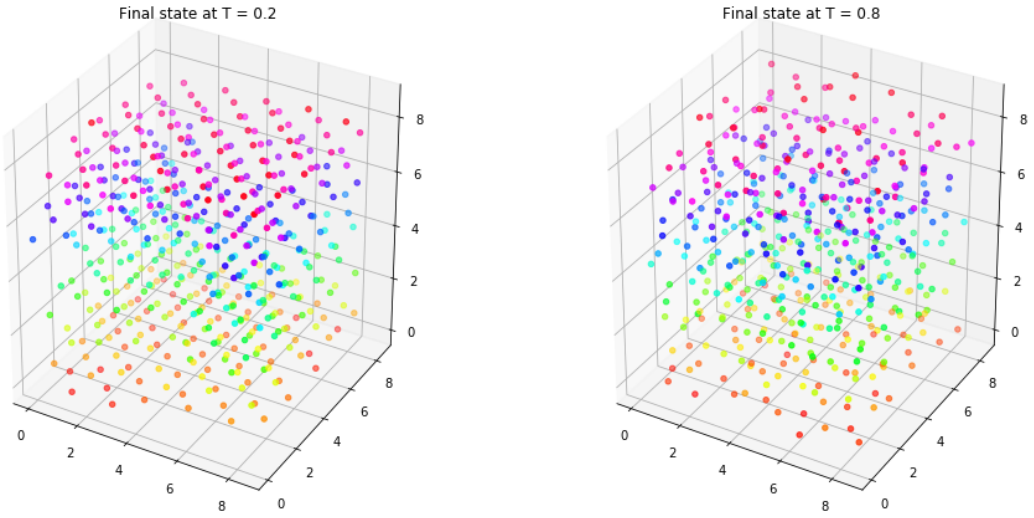
[23]:
```
plt.figure(figsize=(16,5))
plt.title("Mean square displacement vs time at T = 0.8")

sns.lineplot(data=time_series_data['rho_0.800']['T_0.800']['MeanSqDisp'])
plt.show()
```



At high temperatures ($T = 0.8\,\epsilon/k_{\mathrm{B}}$), the mean square displacement continues to increase, showing that the particles are free to move, and therefore we have a liquid.

To further convince us that the system is in different phases under these choices of parameters, we can literally look at the final configuration of atoms, and we'll see a striking difference.

[34]:
```
fig = plt.figure(figsize=(16,8))

axL = fig.add_subplot(1, 2, 1, projection='3d')
axL.set_title('Final state at T = 0.2')

axL.scatter3D(final_state_data['rho_0.800']['T_0.200']['PosX'],
              final_state_data['rho_0.800']['T_0.200']['PosY'],
              final_state_data['rho_0.800']['T_0.200']['PosZ'],
              c=final_state_data['rho_0.800']['T_0.200']['PosZ'],
              cmap='hsv')

axH = fig.add_subplot(1, 2, 2, projection='3d')
axH.set_title('Final state at T = 0.8')

axH.scatter3D(final_state_data['rho_0.800']['T_0.800']['PosX'],
              final_state_data['rho_0.800']['T_0.800']['PosY'],
              final_state_data['rho_0.800']['T_0.800']['PosZ'],
              c=final_state_data['rho_0.800']['T_0.800']['PosZ'],
              cmap='hsv')


#ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');

plt.show()
```

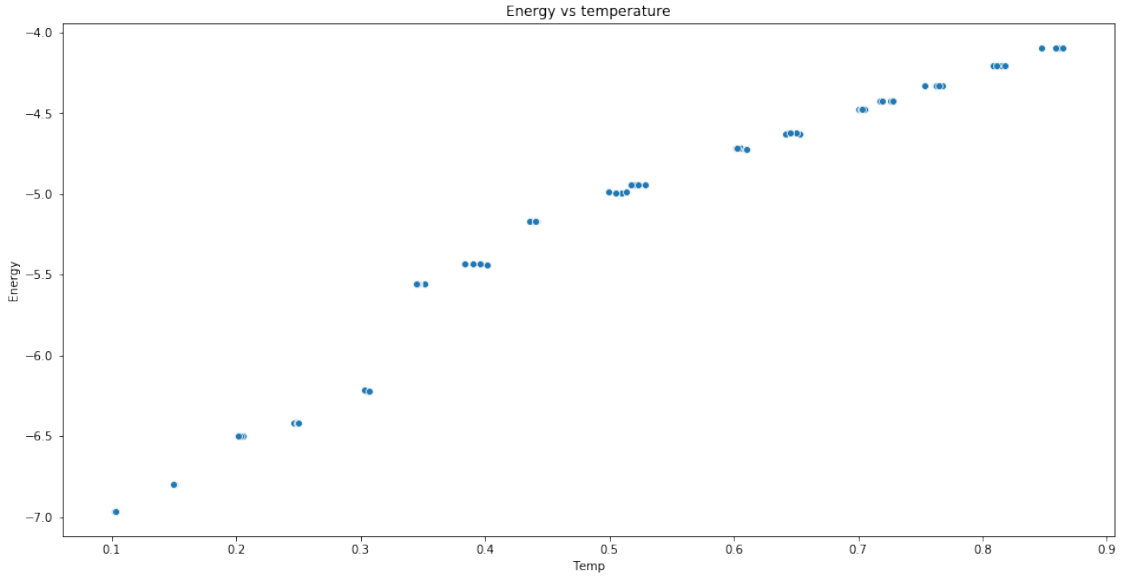


As we can see, at low temperatures, the equilibrium state is a crystalline configuration of atoms, whereas at high temperatures, the atoms are more randomly distributed.

1.5 Energy vs temperature and latent heat of fusion

It is very informative to plot the energy vs temperature curve. A discontinuity in the E-T curve represents a first-order phase transition. There is a jump in energy which represents the energy released by the atoms fusing into a crystal lattice (or alternatively, it is the extra heat we need to *add* to the system in order to melt it, if we are heating it up).

[53]:
```
plt.figure(figsize=(16,8))
plt.title("Energy vs temperature")

#densities = thermo_meas_data['Density'].unique()
#thermo_meas_dict = {d : thermo_meas_data.loc[thermo_meas_data['Density'] == d]␣
 ↪for d in densities}

thermo_meas_slice = thermo_meas_data.loc[abs(thermo_meas_data['Density'] - 0.8)␣
 ↪< 0.01]

sns.scatterplot(x=thermo_meas_slice['Temp'], y=thermo_meas_slice['Energy'])
plt.show()
```

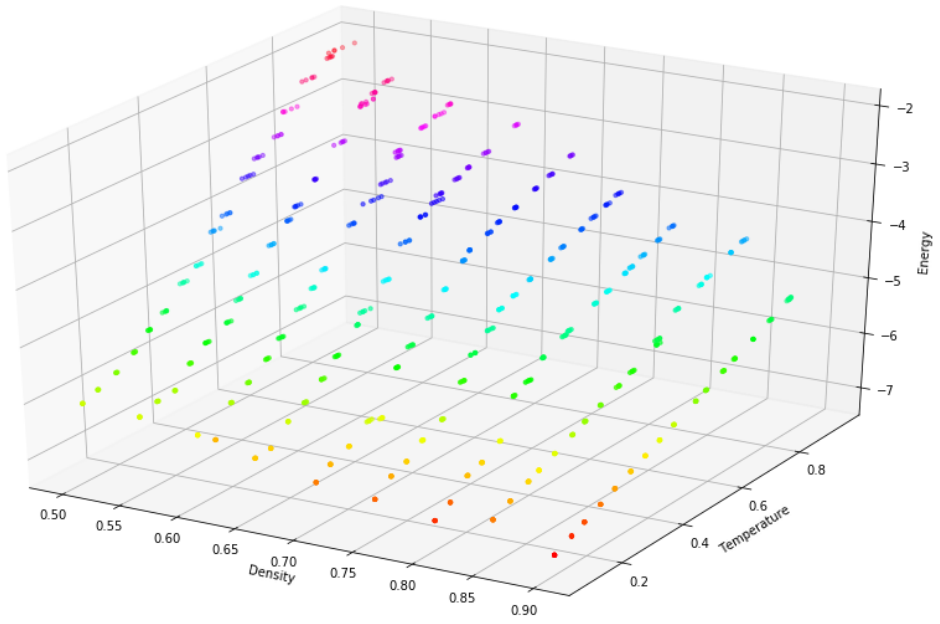


As we can see, this does appear to show a discontinuity around $T = 3.5\,\epsilon/k_{\mathrm{B}}$. However, this is a very small simulation of atoms (only about 500). A bigger simulation should be expected to give a more accurate phase transition.

1.6 Energy vs temperature and density

By plotting energy over two axes of temperature and density, we can begin to visualize what the phase diagram of this system looks like.

[57]:
```
fig = plt.figure(figsize=(16,10))

ax = fig.add_subplot(1, 1, 1, projection='3d')
#ax.set_title('Energy vs density and temperature')

ax.scatter3D(thermo_meas_data['Density'],
             thermo_meas_data['Temp'],
             thermo_meas_data['Energy'],
             c=thermo_meas_data['Energy'],
             cmap='hsv',
             s=10)

ax.set_xlabel('Density')
ax.set_ylabel('Temperature')
ax.set_zlabel('Energy')

plt.show()
```



The system is too small to see an obvious region of phase transitions. We can try running with a larger system.