

**Molecular Dynamics Final Project**

**University of California, Berkeley E7**

**Ihsan Ahmed, Kaleb Savage, Liam Webster, Ning Pichitsurakij,**

**Tan Triyanond, Tomo Yoshino, Yenvy Nguyen**

**Professor Shaofan Li**

**12/17/2020**

# **Background Information:**

Dubbed as a computational microscope, molecular simulation is a method used to create simulations of atomic movement. Molecular simulations help us understand how these small particles interact individually as atoms and molecules. In addition, the simulation not only allows engineers to predict the location of the atoms but also allows scientists to study further how the forces of interaction vary with different sizes or types of particles.[[1]](#footnote-0) Molecular dynamics simulations, specifically, gives a closer look at intermolecular forces and energy.

In the 1950s, Alder and Wainwright introduced the idea of molecular dynamics. The model is based on Newton's second law of motion, which describes the atoms' movement for a given time interval. Hence, according to Newton, the applied force on each atom is equal to the rate of change of momentum. The model also utilizes the Lennard-Jones potential to approximate the interactions between a pair of uncharged atoms at an atomic scale. However, this was not until the late 1960s, when Rahman simulated the first MD simulation on liquid argon. A simulation followed the breakthrough on liquid water and proteins in the following decade. To this day, molecular dynamics simulations remain one of the most important discoveries in science and technology. Ultimately, MD helped researchers understand complex DNA-protein interactions, determine molecular structures using NMR and X-ray crystallography, etc[[2]](#footnote-1).

MD is used in fields such as materials science, biophysics, and biochemistry. With the advancements in computers and software, MD is able to aid in protein folding issues, protein-ligand complex stability, and protein docking constraints by providing visual energy analysis and allows scientists insight on how to optimize molecular structures to their advantage.[[3]](#footnote-2) Other applications of MD include creating visualization of drug and protein interaction, assembly of molecular systems, and polymer crystallization.

**Project Description:**

The project mission is to design and code a molecular dynamics simulation via MatLab. A generic simulation that approximates the intermolecular forces and energy of a group of 32 uncharged particles in a 1x1 box. Using mathematical functions and physics laws, the trajectory of each particle in the given box is animated in a 2 dimensional x y plane. To make the simulation more realistic, the expression of Lennard-Jones potential is applied, and a graph that compares intermolecular potential energy to intermolecular distance is displayed as a part of the program’s result. In addition to the animated simulation, the average kinetic energy, potential energy, temperature, and pressure variation of the system are also calculated and plotted as graphs.

This project is composed of six MATLAB functions, including “main.m”, “MD.m”, “LJ\_plot.m”, “compute\_force.m”, “Calculate\_Temperature.m”, and “animation.m”. The “main.m” function file serves as the baseline file. It initializes the simulation, setting up its parameters, and calls upon the other files. The “MD.m '' function is the largest script and sets the particles positions and calls the other functions to calculate the particle's average kinetic energy, potential energy, temperature, pressure variation. The rest of the function files begin to specialize and have very specific jobs. Each of the functions and its theory will be further explained later in this report. Generating the molecular dynamics simulation using MATLAB demonstrates how programming can integrate concepts and mathematical expressions to create a useful tool for scientists and engineers.

**Main Function Theory**

The main file initializes the parameters for the simulation. The simulation runs by applying random motion onto 32 atomic particles in a two dimensional 1.0m x 1.0m square box. The box has a density equivalent to the number of particles divided by the size of the box. Besides, the volume of the box is the box size to the power of its dimensions- in this case two. Another key aspect of the simulation is the ability to calculate the center of mass. The mass center is the sum of all the positions of all the particles at a particular instant divided by the total number of particles. These parameters help create restrictions to the simulation.

The main function also calls on other functions to parametrize the simulation. For instance, the function MD that contains information on the positions of all the particles at a particular time, derived from Newton’s second law, are called in order for the simulation to show the exact position of each particle on a plot. The main function also calls on other functions related to the temperature and force so that the plot considers the molecular interaction in the simulation.

**Main Function Script**

The main function defines and uses the following variables:

| **Initializing the Particles** | | |
| --- | --- | --- |
| **Variable** | **Type** | **Description** |
| DIM | 1x1 double | Dimension of space to observe |
| N | 1x1 double | Number of particles |
| BoxSize | 1x1 double | Length of one side of the space to observe (2D space in this case) |
| volume | 1x1 double, global variable | Area of space to observe (since DIM=2 in this case) |
| density | 1x1 double, global variable | Particles per unit of space (particle/area unit) |
| position | 32x2 double | Starting position of particles (x,y) |
| pos (line 17) | 32x2 double | Position divided by box size, to scale to 1x1 space |
| MassCenter | 1x2 double | Average position of all particles combined |
| pos (line 22) | 32x2 double | Positions, with (0,0) located at the mass center of all particles |
| **Performing the Simulation** | | |
| **Variable** | **Type** | **Description** |
| NSteps | 1x1 double | Number of frames to take new measurements |
| deltat | 1x1 double | Time between each NStep |
| TRequested | 1x1 double | Reduced temperature (units not specified) |
| DumpFreq | 1x1 double | Frequency at which a frame is dumped for the animation (every 100 frames) |
| epsilon | 1x1 double | Depth of the potential well |
| ene\_kin\_aver | 1x10000 double | Avg kinetic energy for all 32 particles per frame |
| ene\_pot\_aver | 1x10000 double | Avg potential energy |
| pressure | 1x10000 double | Pressure at each frame |
| temperature | 1x10000 double | Temperature at each frame |
| pos (line 42) | 32x2 double | Position of the particles at the 10000th frame |

The first thing that must be done is to create the 2D environment in which we will run the molecular dynamics simulation. In order to do this, a 2D space with pre-defined constraints must be created, and the position of each of the particles in the 2D space must be determined. The dimension of the space, the number of particles, and the length of one side of the space are first defined as DIM, N, and BoxSize, respectively, and each given a value. The volume of the space and the density of particles within the space are then defined and calculated using the aforementioned variables.

To determine the position of each of the particles, data is loaded from a position.dat file, and it is scaled to the dimensions of the space. The center of mass of the particles is then calculated, and a for loop is used in order to adjust the assigned positions of the particles with the center of mass being the origin. Another for loop is used to plot the positions of the particles on a figure and place markers on those positions to represent the particles.

To start the simulation, the function defines a variable for the number of frames/steps before measurements are recorded, the time between each of the recording steps, the reduced temperature, the frequency at which frames are dumped for animation, and the dispersion energy. These variables are then inputted into the MD function, where the simulation is performed (more information on the MD function can be found in the **MD Theory** and **MD Script** sections), and the variables representing the average kinetic energy, the average potential energy, the pressure at every animation frame, the temperature at every animation frame, and the positions of the particles at the 10000th frame are defined.

At this point, the state of the particles has been recorded every 100th frame by the MD function, so the recorded animation frames are used to create an animation of the simulation using the animation function.

The output variables obtained from the MD function are then used to plot four graphs. Each of the graphs represents the time variation of its respective variable.

Below is a recap of the major lines of code and what each of them does:

| Lines 24-30:  figure  for i=1:N  plot(pos(i,1),pos(i,2),'o','MarkerSize',10,'MarkerFaceColor','b');  text(pos(i,1),pos(i,2),num2str(i));  hold on  end  axis off  This for loop plots the initial state of the particles in the 2D space. |
| --- |
| Lines 41-43:  [ene\_kin\_aver,ene\_pot\_aver,temperature,pressure,pos]= MD(pos,NSteps,deltat,TRequested,DumpFreq,epsilon,BoxSize,DIM);  animation(NSteps/DumpFreq);  These lines input the necessary values into the MD and animation functions. |
| Lines 46-50:  figure  subplot(4,1,1);  plot(ene\_kin\_aver,'k-');  ylim([0.485,0.514]);  ylabel('E\_{K}');  The code above plots a graph representing the time variation of the average kinetic energy. Note that lines 51-62 are repetitions of lines 47-50 above, and they create subplots representing the time variation of the average potential energy, the temperature, and pressure, respectively. |

**MD Theory**

As stated in the introduction, MD is based on motion of particles described by Newton’s Second Law. Let’s assume that for a given atom, the acceleration is constant, such that . Acceleration is the first derivative of velocity with respect to time. Therefore,

Integrating both sides of the acceleration equation gives the velocity expression:

Velocity is the first derivative of position with respect to time. Therefore,

For constant velocity,

Integrating both sides of the velocity equation gives the particle’s position expression:

As a result, from Newton’s second law, the position and velocity of the particles can be predicted.

From the derivation above, the velocity of the particles can be expressed as However, the velocity can also be affected by the temperature according to the thermal velocity equation:

Therefore, this relationship between temperature and velocity needs to be accounted for when calculating a position of a particle at a given time.

Factors that affect the velocity, such as temperature, are related to the pressure of the system. Hence, pressure needs to be calculated. The expression used to calculate the pressure can be obtained from the ideal gas law equation.

**MD Script**

The MD function has 8 inputs and 5 outputs:

| function [ene\_kin\_aver, ene\_pot\_aver, temperature, pressure, pos] = **…**  MD(pos, NSteps, deltat, TRequested, DumpFreq, epsilon, BoxSize, DIM) | | |
| --- | --- | --- |
| **Input** | **Type** | **Description** |
| pos | 32x2 double | Initial position of all particles |
| NSteps | 1x1 double | Number of frames to calculate, 10000 in this case |
| deltat | 1x1 double | Number of time between each frame, 0.0032 in this case |
| TRequested | 1x1 double | Reduced temperature, 0.5 in this case |
| DumpFreq | 1x1 double | Frequency to save frames of particles. Save every 100 frames as an image. |
| epsilon | 1x1 double | Depth of the potential well (dispersion energy), basically an energy scaling constant that says how strongly two particles attract each other. 1.0 in this case |
| BoxSize | 1x1 double | Size of observed space, 10.0 |
| DIM | 1x1 double | Dimension of observed space, 2 |
| **Output** |  |  |
| ene\_kin\_aver | 1x10000 double | Avg kinetic energy for all 32 particles per frame |
| ene\_pot\_aver | 1x10000 double | Avg potential energy |
| pressure | 1x10000 double | Pressure of every frame |
| temperature | 1x10000 double | Temperature at every frame |
| pos | 32x2 double | Position of particle at final frame |

The MD function does several tasks. It generates the position of particles at every frame using a for loop. New positions are found using the previous frame’s velocities and accelerations. For each frame, the average kinetic energy, average potential energy, pressure, and temperature are also calculated. Kinetic energy and temperature are found through the Calculate\_Temperature function. Average potential energy is found using the compute\_force function. Pressure is calculated using a modified ideal gas law equation.

The first 21 lines of code sets up space for the outputs and randomly generates the first frame’s velocity and acceleration. The built-in function, **randn** is used to determine the initial vel and acc. **randn** selects numbers based on a normal distribution centered at 0, so 68% of generated numbers fall between -1 and 1, 95% of numbers fall between -2 and 2, and 99.7% of numbers fall between -3 and 3. From each vel and acc component, 0.5 is subtracted. This is most likely to show how non-random disturbances in the air, like a leftward push, can affect the molecular dynamics of a system.

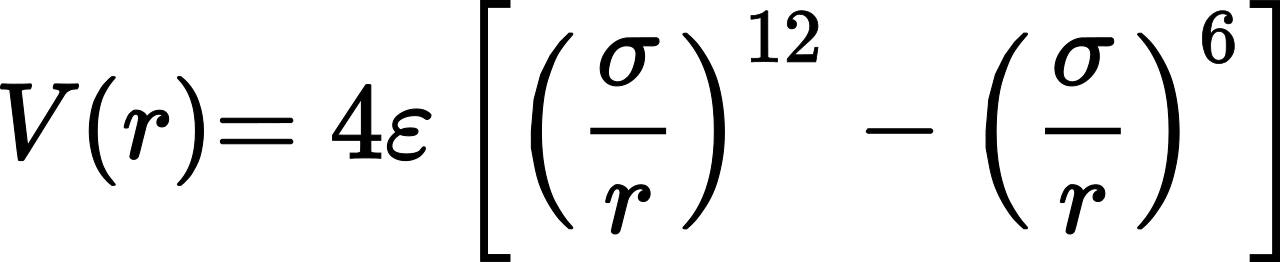
The second part of the code is a for loop that calculates the 10,000 outputs frame by frame. Since all of the code has comments, only the important or more complicated lines will be brought up here. The vel array changes twice based on temperature and molecular interactions, which will be discussed further in this table.

| Line 31:  pos = pos + deltat•vel + 0.5•(deltat^2.0)•acc  This is from the physics equation:  **xt = x0 + v0•t + (½)•a•t2** |
| --- |
| Lines 38 and 39:  chi = sqrt(TRequested/temperature)  vel = chi•vel + 0.5•deltat•acc  The simple physics equation would be: **vt = v0 + a•t**. Temperature increases the velocity of particles. The expression for chi is related to the thermal velocity equation: . Chi scales the temperature effects to TRequested, and applies it to the velocity. |
| Line 45:  vel = vel + 0.5•deltat•acc  The new velocity accounts for the accelerations caused by particle-particle interactions. This is similar to the physics equation: **vt = v0 + a•t**, except half of the acceleration already exists from the previous for loop (which is the acc used in Line 39), and the other half is calculated from intermolecular forces using compute\_force. |

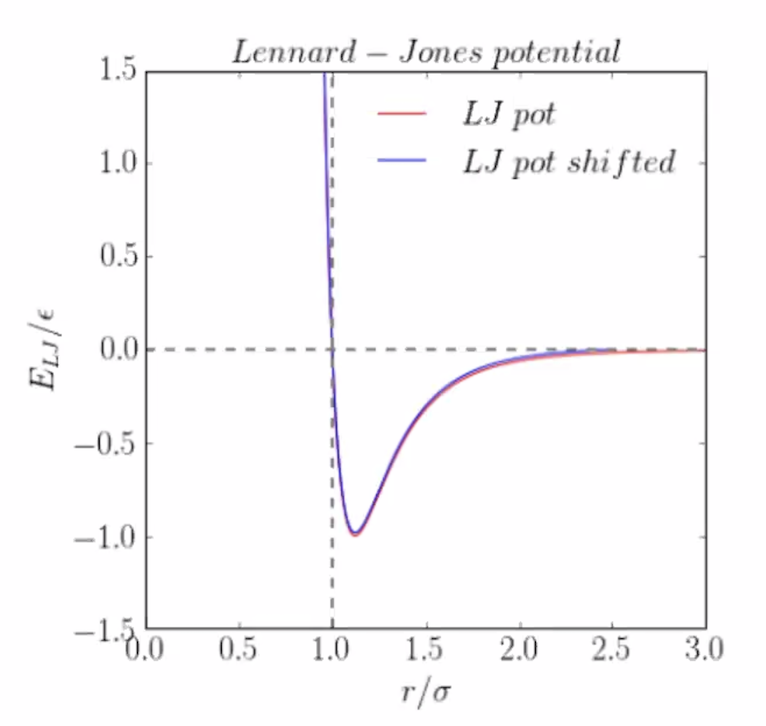
Lines 54 to 74 of the loop plot the particles at every 100th frame and stores it in an image folder.

**Compute force with Lennard-Jones Potential Theory**

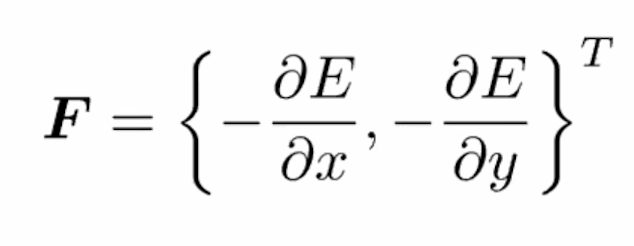
The Lennard-Jones potential is a mathematical model for the interaction between electronically neutral atoms and molecules, where ‘r’ is equal to the distance between two particles, ‘ε’ is energy of minimum interaction, and ‘σ’ is the Van der Waals radius.[[4]](#footnote-3)



The Van der Waals radius is half the minimum distance between the center of two particles. As seen by the graph of the Lennard-Jones potential, when the particles are closest to each other the positive potential energy, where positive means repulsion, has a high magnitude. As ‘r’ and ‘σ’ become equal, potential energy becomes 0 since all that energy is now kinetic energy. When the particles move farther away from each other, the magnitude of potential energy starts to increase in the negative direction, where negative energy means attraction.[[5]](#footnote-4) However, the increase in magnitude only goes so far because the growing distance makes energy more negligible. Eventually, the potential energy is very close to 0 because of distance, so for the purposes of this project we use a cutoff of r/σ = 2.5. Because of the cutoff, there is an unrealistic jump in energy that is solved by shifting the graph so that potential energy is 0 at r/σ = 2.5.

****

Force can be described in two dimensions using energy in terms of x and y:



We use the Lennard-Jones potential to calculate force by differentiating the function. Work (Force\*displacement) is equal to the change in potential energy. By differentiating the work function with respect to displacement, we are left with force, where force = dW/dx = dV/dr.

**Compute\_force Script**

The compute\_force function has 5 inputs and 3 outputs:

| function [acc, ave\_ene, virial\_coeff] = compute\_force(pos, epsilon, BoxSize, DIM, N) | | |
| --- | --- | --- |
| **Input** | **Type** | **Description** |
| pos | 32x2 double | Position of particles for a single frame |
| epsilon | 1x1 double | Constant that describes how strongly two particles interact with each other. epsilon=1 |
| BoxSize | 1x1 double | Size of observed space, 10.0 |
| DIM | 1x1 double | Dimension of space, 2 |
| N | 1x1 double | Number of particles, 32 |
| **Output** |  |  |
| acc | 32x2 double | Acceleration on each particle, caused by intermolecular interactions |
| ave\_ene | 1x1 double | Average potential energy in observed frame |
| virial\_coeff | 1x1 double | Virial stress for that frame |

The compute\_force function calculates the intermolecular forces felt by each particle and uses that information to find the outputs.

The function starts by setting up variables, Sij and Rij. These variables store the distance between the two points being compared. Sij and Rij reset when new particles are being compared. In line 11, Rcutoff is set at 2.5, which means the particles need to be within 2.5 units of each other to feel intermolecular forces. In line 12, phicutoff is calculated using the Lennard-Jones equation with r=Rcutoff and σ=1. Phicutoff is basically the potential energy we can ignore, as it is outside of Rcutoff.

The intermolecular interactions are calculated within the for loops from lines 15-44. The outer two for loops are used to compare a particle to every particle that is listed after it. After all loops run to completion, every particle would have been compared to every other particle. The forces felt on each particle would be the sum of the forces exerted by all particles within Rcutoff.

Some modifications to Rij and Sij are done before being applied to the LJ potential equation. Rij is the true distance between particles, and Sij is all when all positions are scaled to a 1x1 board. Sij is the distance between two points, and could be [1 1] if the particles are at the opposite corners of the board. However, we want the particle to be within [0.5 0.5] of each other because particles at the edge of the board “wrap around” to the other side. Using this logic, no particle should be more than [0.5 0.5] board units away from each other. Rij is squared into Rsqij to get rid of negatives. Rsqij is basically the argument inside the distance formula; (x2-x1)2 + (y2-y1)2.

The LJ potential equation is only applied to particles within Rcutoff, and the potential energy between two particles is represented as phi in the code. Half of this potential energy is added to one of the particles, and the other half is added to the second particle. Potential energy accumulates for each particle as the interactions with the remaining particles are calculated.

To calculate acc and virial\_coeff, we need to use dphi, which would be (1/r)•(dV/dr) from the LJ equation. Dphi is the rate potential energy changes as distance-squared changes. The equation for dphi differs from the direct dV/dr equation (assuming σ=1 and ε=1):

↓ 1) multiply by 1/r

↓ 2) switch signs

The first modification to dV/dr is done to help calculate virial. Virial theorem helps convert potential energy into kinetic energy, and the virial variable has the same units as potential energy. While virial theorem is mainly used in astrophysics when relating gravitational potential energy to kinetic energy, it can also be used for intermolecular interactions. In this case, we only want to deal with positive distances when accumulating virial stress, so we multiply by Rsqij (r2) for instead of r. To have the same units as ene\_pot, dphi has to be (1/r)•(dV/dr), and the (1/r) term would cancel out the extra power in Rsqij. The second modification, switching the signs of the terms, is reversed in line 38: virial = virial -dphi•Rsqij. Dphi would lead to mainly negative values since the signs on dV/dr are switched.

Acceleration is also calculated using dphi. As mentioned above, dV/dr should represent force, as it is the derivative of potential energy, V(r). In this case, mass of the particles is irrelevant, so per *F=ma*, dV/dr should also represent acceleration in this code. Dphi is (1/r)•(dV/dr), so multiplying by Sij (which has the same units as r) should leave us with dV/dr and help scale all accelerations to the same relative units.

**Calculate\_Temperature Theory**

The Calculate\_Temperature function has 4 inputs and 2 outputs.

| function [ene\_kin\_aver,temperature] = Calculate\_Temperature(vel,BoxSize,DIM,N) | | |
| --- | --- | --- |
| **Input** | **Type** | **Description** |
| vel | 32x2 double | Velocity of each particle |
| BoxSize | 1x1 double | Size of observed space, 10.0 |
| DIM | 1x1 double | Dimensions of space, 2 |
| N | 1x1 double | Number of particles, 32 |
| **Output** |  |  |
| ene\_kin\_aver | 1x10000 double | Average kinetic energy of each particle. |
| temperature | 1x10000 double | Proportional to the average kinetic energy.  Since it is 2D, T=2\*KE/2 |

The theory behind this function is not extensive. In quick summary the function takes the velocities for each particle, and calculates the kinetic energy for each particle and adds it on top of each other. Then the function takes that total kinetic energy in the boxed region and divides it by the number of particles. Which gives the average kinetic energy of each particle which is proportional to the temperature. The physics behind this function is quite simple. Dealing with kinetic energy of particles or objects and relating that to temperature. If an object experiences a force that is greater than the friction or gravitational forces the object experiences, then the object will begin to move. As the object begins to move it acquires kinetic energy; the energy an object possesses due to its motion. The force exerted on said object is said to do work on the object. It causes the object to accelerate which means there is a change in the object's velocity. And a change in velocity means there is a change in its kinetic energy.[[6]](#footnote-5)

Every atom has kinetic energy but not a temperature. Temperature is a concept we have created to relate the average kinetic energy of a group of molecules to a number we can easily understand and grasp an intuition upon its physical properties. This is an important distinction, in this function the individual kinetic energies of each particle are summed and then divided by the number of particles which results in the average kinetic energy, which is proportional to the temperature; but for all of the particles in the boxed region some will have a kinetic energy below and some above the value of the group’s temperature. From calculating the average kinetic energy and thus temperature one can make important inferences about the systems physical properties, and can further make calculations using this information.

**Calculate\_Temperature Script**

The Calculate\_Temperature function has 22 lines of code. It takes the calculated velocity, the box size, and the number of particles, and outputs the temperature and average kinetic energy of each particle. In the first line of code, the variable ene\_kin is initialized at zero. In the second line, a for loop is created that runs from 1 to N, in our case N=32 particles. Inside of the for loop the real velocity or the total velocity is calculated by scaling the previous velocity in accordance with the box size, 10. Then the total kinetic energy in the box is calculated using the real velocity and the equation KE = 0.5mv^2 or written KE = 0.5m(v\*v). After the for loop runs for each iteration, the total kinetic energy is divided by the number of particles, 32. We then have the first output the average kinetic energy of each particle. In the next line temperature is calculated by dividing the average kinetic energy by DIM and then multiplying by 2. Which makes temperature and the average kinetic energy of each particle proportional, which is in accordance with the Kinetic Theory of Gases.

**Animation Function**

The animation portion of the code has only a single input, which is the amount of frames used in the animation. In total, 100 images were created and strung together in sequence to form the animation. The code is explained below.

| Lines 3-4:  nFrames = n;  mov(1:nFrames) = struct('cdata',[],'colormap',[]);  These lines assign the function input n as the amount of frames. The value for n is called as Nsteps/DumpFreq, which has a value of 100. Then, a struct array is made with fields for cdata and colormap for later use. |
| --- |
| Lines 5-17:  for i=1:nFrames  Img = imread(sprintf('images/%d.bmp',i\*100));  imshow(Img,[]);  frame = getframe(gcf);  im = frame2im(frame);  [I,map]=rgb2ind(im,256);  mov(i)= getframe(gcf);  if(i==1)  imwrite(I,map,'movefig.gif','DelayTime',2.0,'LoopCount',Inf)  else  imwrite(I,map,'movefig.gif','WriteMode','append','DelayTime',2.0)  end  end  First, the function assigns the first image file (100.bmp) and assigns it to the variable Img. Img is displayed using imshow and then is captured as a frame using the getframe function. That frame is then stored under the variable im as the image data taken from the movie frame. After that, the rgb2ind function converts im to an indexed image with 256 colors and the corresponding slot of the mov structure array is assigned the proper frame. Lastly, the first indexed image is used to create a graphics file, and every following image is added to that graphics file. |
| Lines 18-24:  v=VideoWriter('collide.avi','Uncompressed AVI');  v.FrameRate = 10;  open(v);  writeVideo(v,mov);  close(v)  end  The last part of this function creates the actual video file. First v is named and specified as an uncompressed AVI. Then the frame rate is set, and v is opened. Then, v is filled with the frames stored in the previously created mov structure array. Lastly, v is closed and the animation is complete. |

**Lennard-jones Potential Energy Plot**

Below are the variables used in the script used to plot the Lennard-jones potential energy:

| **Variable** | **Type** | **Description** |
| --- | --- | --- |
| r | 1x500 double | Various distances between two particles |
| epsilon | 1x1 double | Depth of the potential well |
| sigma | 1x1 double | measure of how close atoms can get without bonding; distance where potential energy is zero |
| E\_LJ | 1x500 double | Lennard-jones potential energy calculated for each value of r |
| Rcutoff | 1x1 double | Closest distance at which particles would show interaction |
| phicutoff | 1x1 double | Potential energy felt by particles at Rcutoff distance apart |
| E\_LJ\_shift | 1x500 double | Shifted form of the Lennard-jones potential energy |

To begin, the script defines r, epsilon, and sigma, and the script then uses those variables in a for loop in order to calculate the Lennard-jones potential energy for each value of r. This is done using the the Lennard-jones equation . That same equation is then used with a predetermined variable, Rcutoff, replacing r in order to calculate phicutoff, the potential energy when particles are a distance of Rcutoff apart. The variable phicutoff is subtracted from each element of E\_LJ in order to find the shifter form of the Lennard-jones potential energies, defined as the variable E\_LJ\_shift. Having obtained both E\_LJ and E\_LJ\_shift, both of these variables are plotted with respect to .

Below are the important lines of code in the Lennard-jones plot script, and what they do:

| Lines 7-9:  for i=1:length(r)  E\_LJ(i) = 4\*epsilon\*((sigma/r(i))^12-(sigma/r(i))^6);  end  This code calculates the Lennard-jones potential energy for each value of r, using the equation . |
| --- |
| Line 13:  E\_LJ\_shift = E\_LJ - phicutoff;  This code calculates the shifted form of the Lennard-jones potential energy by subtracting the potential energy at r = Rcutoff from the calculated potential energy values. |
| Lines 15-22:  plot(r,E\_LJ,'r-','LineWidth',1);  hold on  plot(r,E\_LJ\_shift,'b-','LineWidth',1);  xlim([0,3.0]);  ylim([-1.5,1.5])  legend('LJ potential','shift LJ potential')  xlabel('r/ \sigma')  ylabel('E\_{LJ}/\epsilon')  This code plots the values of the Lennard-jones potential energy, and the shifted Lennard-jones potential energy with respect to . |

1. Embo. (2016, June 04). *An Introduction to Molecular Dynamics.* Embo. <https://aidanbudd.github.io/ppisnd/trainingMaterial/natashaWood/What_is_MD.pdf> [↑](#footnote-ref-0)
2. Embnet. (n.d.). *Theory of Molecular Dynamics Simulations.* Embnet. <https://embnet.vital-it.ch/MD_tutorial/pages/MD.Part1.html> [↑](#footnote-ref-1)
3. Perricone, U., Gulotta, M., Lombino, J., Parrino, B., Cascioferro, S., Diana, P., Cirrincione, G., Padova, A.,. (2018 April 19). *An Overview of Recent Molecular Dynamics Applications as Medicinal Chemistry Tools for the Undruggable Site Challenge.* MedChemComm [↑](#footnote-ref-2)
4. Chemistry LibreTexts. (2020 August 15). *Lennard-Jones Potential*. Chemistry LibreTexts. <https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Physical_Properties_of_Matter/Atomic_and_Molecular_Properties/Intermolecular_Forces/Specific_Interactions/Lennard-Jones_Potential> [↑](#footnote-ref-3)
5. Vitz, E., Moore, J., Shorb, J., Prat-Resina, X., Wendorff, T., Hahn, A., (2020 August 17). *Gravitational and Electrostatic Potential Energy*. Chemistry LibreTexts. <https://chem.libretexts.org/Bookshelves/Ancillary_Materials/Exemplars_and_Case_Studies/Exemplars/Biology/Gravitational_and_Electrostatic_Potential_Energy> [↑](#footnote-ref-4)
6. Holzner, S., & About the Book Author Steven Holzner. (n.d.). How to Calculate the Kinetic Energy of an Object. Retrieved December 14, 2020, from https://www.dummies.com/education/science/physics/how-to-calculate-the-kinetic-energy-of-an-object/

   Libretexts. (2019, June 5). Kinetic Energy and Temperature. Retrieved December 14, 2020, from https://chem.libretexts.org/Bookshelves/General\_Chemistry/Book:\_CLUE\_(Cooper\_and\_Klymkowsky)/5:\_Systems\_Thinking/5.1:\_Temperature/Kinetic\_Energy\_and\_Temperature [↑](#footnote-ref-5)