# Modeling Atomic Interactions in Xenon Using the Morse and Lennard-Jones Potentials

Jonathan Bunton

December 13, 2017

**Abstract**

Several models exist for inter-atomic forces, the majority of which with the basis of weak attractive force at long range, countered by a strong repulsive force at close distance. Two such models are the Lennard-Jones and Morse potentials, each of which accomplish this behavior with differing mathematical models.

This paper analyzes the behavior of a lattice of Xenon atoms in both the Lennard-Jones and the Morse potentials by using Verlet integration to simulate atomic behavior using parameters for these potentials as experimentally determined from the second virial coefficient. [3, 6] Verlet integration shows excellent conservation of momentum and energy, with a maximum deviation on the order of 0.01%.[5] The data also exhibits the spring-esque behavior suggested by the potential curves, with the Morse potential ultimately showing a more oscillatory behavior due to its more flexible parameters. In addition, both potentials exhibit evaporation and decreased order with increased temperature.

## Introduction

Various mathematical models exist to approximate the shape of inter-atomic potentials. The fundamental idea behind each is to accurately represent both the attractive long-range forces and the repulsive short-range forces within the same model. Two of the most popular models for these interactions (which behave particularly well in the case of noble gases) are the Lennard-Jones and Morse potentials. Both potentials are characterized by a well-shaped curve, with various parameters to control the shape, which are dependent on the element at hand.

The Lennard-Jones potential is generally nicknamed the "6-12" potential, as it achieves the well-shaped potential behavior by using negative powers of $r$, shown in eq. 1. The

Lennard-Jones potential creates this with two terms:

$$U(r) = \varepsilon \left[ \left( \frac{r_m}{r} \right)^{12} - 2 \left( \frac{r_m}{r} \right)^{6} \right] \tag{1}$$

Here $r$ is the distance between particles, and the parameters $\varepsilon$ and $r_m$ determine the depth of the potential curve and its lowest point, respectively. [2] The parameters $\epsilon$ and $r_m$ are determined both computationally and experimentally, generally being derived from the second virial coefficient. [6] The Morse potential achieves a similar shape with a different mathematical approach:

$$U(r) = D_e \left(1 - e^{-a(r-r_e)}\right)^2 \tag{2}$$

In this potential, the parameters $D_e$ and $a$ control the depth and width of the potential well, while $r_e$ is the lowest point. [4] The inclusion of three parameters rather than the two in the Lennard-Jones potential gives the Morse potential more control over the well depth and width. These parameters are also often derived from a combination of computational and experimental methods. [3]

Both of these potentials can then be used to calculate conservative forces between two or more particles. The system in question in this paper is a lattice of Xenon atoms, with lattice spacing such that the system is in a relatively low potential energy state. Due to the shape of both potentials, this defines lattices with $r_m$ spacing for the Lennard-Jones potential and $r_e$ spacing in the Morse potential.

Because these potentials produce conservative vector fields, the fundamental conservation laws hold for the system of atoms. This means that as a measure of numerical accuracy, the linear momentum, angular momentum, and energy in the system should remain constant throughout. The linear momentum of the system (and therefore the linear momentum in each direction) must be constant, as well as the total energy, in order for the simulation to be considered accurate. This motivates the monitoring and plotting of these values over time, as shown below.

The simulation as a whole hinges on the initialization of a lattice of Xenon atoms, generation of initial velocities for each particle, and the calculation of the positions of each particle over time within a position-based potential function, all the while keeping total momentum and energy constant throughout. This paper does all this through the use of Verlet integration as a method of inteagral approximation with discrete timesteps.

## Methods

The Fortran code written to model this system uses Velocity Verlet, which is based on Verlet integration within a position-based potential. Recalling the equations of motion for a conservative system, $\vec{F} = m\vec{a}(t) = -\nabla V(\vec{r}(t))$, it is clear that within the discrete system, the negative gradient of the potential function will need to be recalculated at each time step. If this force is assumed roughly constant within the time step considered, the equations of motion are then:

$$\vec{v}(t) = \vec{v}(0) + \frac{1}{2}\vec{a}(t) \tag{3}$$

$$\vec{r}(t) = \vec{r}(0) + \vec{v}(t) + \frac{1}{2}\vec{a}(t)^2 \tag{4}$$

Verlet integration, or in this case the nicknamed algorithm Velocity Verlet, uses a central difference approximation to the second derivative to provide a value for the acceleration at a given point. Utilizing Velocity Verlet also requires a discretization of the system in time, where the equations of motion are instead calculated with a discrete timestep $\Delta t$ to yield:

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+\Delta t)}{2}\Delta t \tag{5}$$

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \tag{6}$$

This set of discrete equations indicates that at each time step, calculating forward a single timestep requires calculating the value of the acceleration at the current time, using the negative gradient of the potential. Because the Morse and Lennard-Jones potentials depend on the relative positions of particles with respect to each other, each timestep requires calculations of the force (and therefore acceleration) from each particle in the system, added together in a vector sum. Once the accelerations have been calculated at the current time $t$, the velocity followed by the position at time $t + \Delta t$ can be found for each particle in the system. This algorithm repeats recursively for as many timesteps forward as required for the time interval considered.

This algorithm is simple, but requires an initial velocity condition for the system. To maintain a realistic model, the velocities are sampled from the Maxwell velocity distribution in each dimension $i$ that is dependent on the temperature of the molecules within the system:

$$f_v(v_i) = \sqrt{\frac{m}{2\pi kT}} \exp\left(\frac{-mv_i^2}{2kT}\right), \tag{7}$$

where $m$ is the particle's mass and $k$ is Boltzmann's constant. This distribution allows for realistic temperature-dependent behavior within the system, which is evident based on the results shown below. To generate velocities from this distribution, it is worth noting that the Maxwell distribution is simply a normal distribution with mean $\mu = 0$ and a standard deviation $\sigma = \sqrt{\frac{kT}{m}}$. For convenience, this could be brought to the standard normal distribution with a z-transform $z = \frac{v}{\sigma}$. Because Fortran will only generate random numbers on the unit interval, the use of a Box-Muller transform is convenient. [1] The Box-Muller transform takes numbers $U_1, U_2$ from a uniform distribution and transforms them to numbers from the standard normal distribution through the equations:

$$Z_1 = \sqrt{-2\ln U_1}\cos(2\pi U_2) \tag{8}$$
$$Z_2 = \sqrt{-2\ln U_1}\sin(2\pi U_2). \tag{9}$$

Considering an initial z-transform, to generate random numbers from the Maxwell distribution from random uniformly distributed numbers $U_1, U_2$:

$$v_1 = \sqrt{\frac{-2m\ln(U_1)}{kT}}\cos(2\pi U_2) \tag{10}$$
$$v_2 = \sqrt{\frac{-2m\ln(U_1)}{kT}}\sin(2\pi U_2). \tag{11}$$

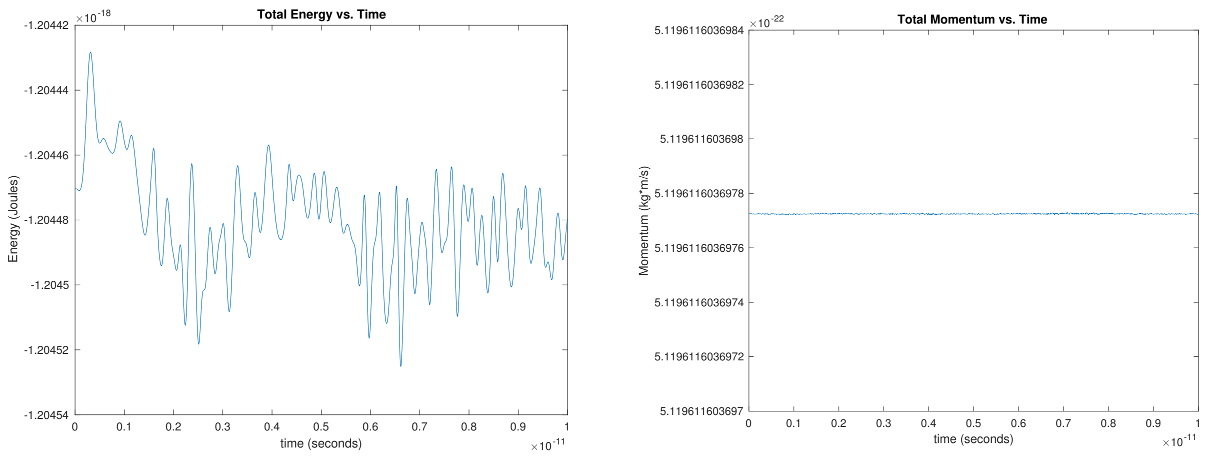
These velocities are initialized alongside the lattice position at the start of the program, which then undergoes 1000 time steps of $\Delta t = 10^{-14}$ seconds. Because of the dependence on temperature, several simulations were performed with various temperature conditions to reveal some temperature-dependent behaviors inherent in the system.

## Results

Both the Lennard-Jones and Morse potentials were simulated using a Fortran code. In order to provide the most realistic system, the code initializes the velocities in each direction using the Maxwell-Boltzmann distribution in each dimension. The system proved semi-stable for both cases, with a slight oscillatory behavior as expected from the potential shapes. Ultimately, the steady-state solution appeared to be a sphere, where each particle is spaced at an adequate distance to be "trapped" in the curve of the potential well.

### Lennard-Jones Potential

The Lennard-Jones (6-12) potential creates a semi-stable system. Our simulation remains stable throughout the movement of the atoms, provided a small enough time-step, in this case $10^{-14}$ seconds. This is shown most clearly by the stability of the graphs in figs. 1a and 1b, which exhibit a deviation on the order of 0.01% in total energy and $< 0.01\%$ in total momentum. This is expected in an acceptable model for the system, proving Verlet integration as a viable method of calculation in this code. These percentages hold for timesteps of $10^{-14}$, but smaller timesteps provide more accuracy. Conversely, larger timesteps result in too drastic of changes in the system for momentum and energy to be effectively conserved.



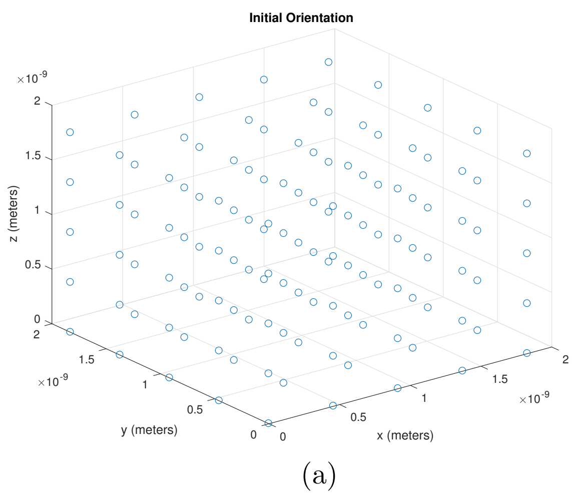
(a) (b)

Figure 1: Plots of (a) total system energy, and (b) total momentum magnitude vs. time for a lattice of Xenon atoms in the Lennard-Jones potential. These graphs were simulated with a temperature of 100K. The extremely stable results show that there is only a variation in energy on the order of 0.01% and $< 0.01\%$ for momentum.

Shifting instead to the resulting position data, the initial and final states of the system are shown in figs. 2a and 2b. In this potential, the atoms are initialized to a lower velocity due to the low temperature, and thus behave similarly to if they were to begin with zero velocity. In this situation, each atom is at a minimum $r_e$ distance apart, causing the long-range attractive force to dominate. After this attractive force acts on the particles, the atoms move closer and experience the abrupt short-range repulsive forces. This causes a slight contraction-expansion cycle as the atoms readjust to a more energetically favorable position. This is visualized through the plot in fig. 3, where the average displacement magnitude shows slight signs of oscillation over time.

## Morse Potential

In the Morse potential, the oscillatory behavior within the potential is again very evident. The energy and momentum of the system are conserved with comparable accuracy on the order of $< 0.01\%$, as seen in figs 4a and 4b. This once again validates the use of Ver-



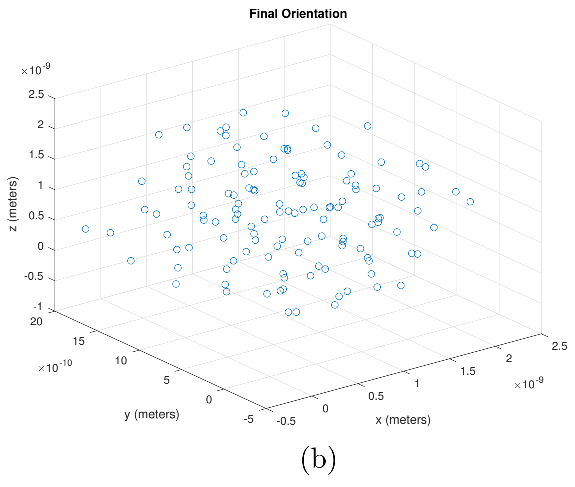


Figure 2: 3D scatter plots of (a) initial system position, and (b) final system position for Xenon atoms in the Lennard-Jones potential. These graphs were simulated with a temperature of 100K. The system shifts from a lattice to a more energetically favorable spherical shape, as expected when below its boiling point.

let integration to simulate this system, provided an appropriate time step is used for the potential.

After validating the model through acceptable conserved quantities, the results of the Morse potential are very similar to those in the Lennard-Jones. Because each atom is initialied at a minimum distance $r_m$ from its neighbors, the long-range attractive force dominates. As this force pulls the atoms inward, the short-range repulsive force acts, causing a nearly identical contraction-expansion cycle to that seen in the Lennard-Jones potential. The Morse potential, however, exhibits noticeably larger oscillations, perhaps owed somewhat to the flexibility in the width parameter $a$ that does not exist in the Lennard-Jones. This pattern is shown in fig. 6, which displays heavy oscillations in the average magnitude of the displacement.

In addition to the oscillatory behavior shown in fig. 6, the 3D scatter plots in figs. 5a and 5b show that the Morse potential tends to pull particles slightly closer together than the Lennard-Jones. Because the Morse potential has more control over the potential well width, this is somewhat expected.

## Temperature Considerations

In both potentials, the initial velocity in each direction for each atom was drawn from the Maxwell velocity distribution. This distribution inherently depends on temperature, and therefore as we increase the temperature, the standard deviation of our initial velocity also

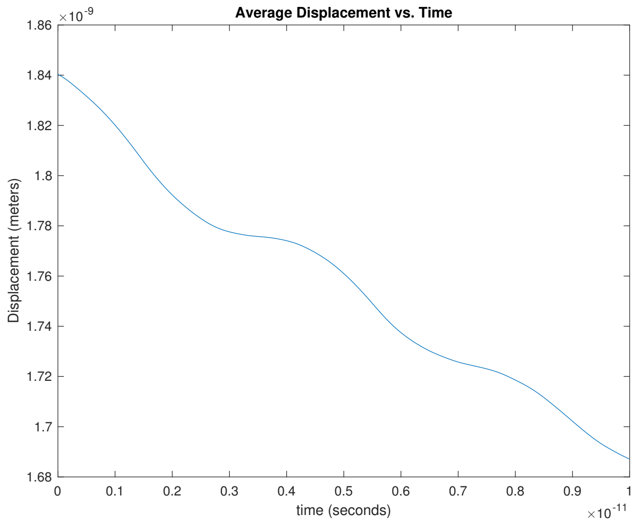
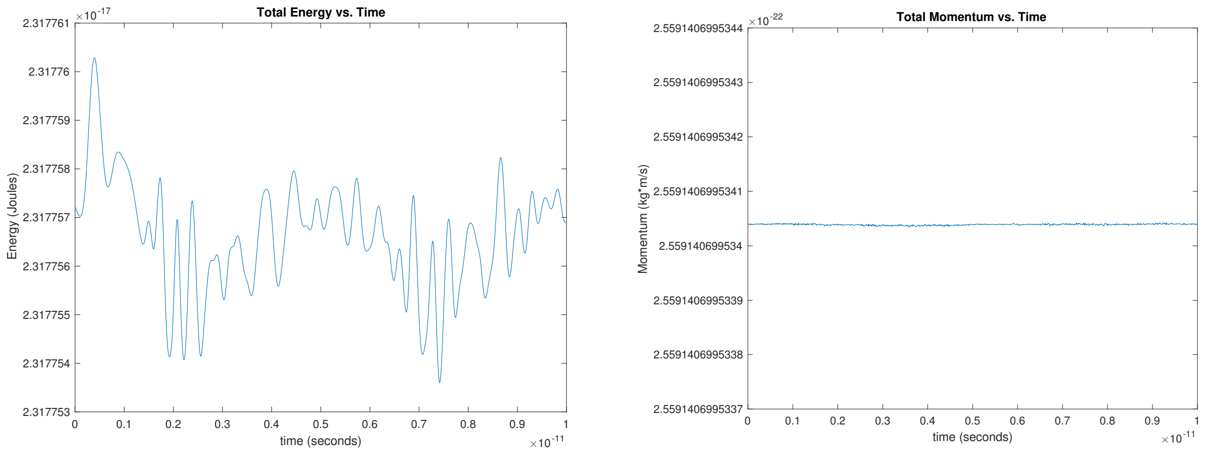


Figure 3: A plot of the average magnitude of the displacement from the origin over time for Xenon atoms in the Lennard-Jones potential. As the atoms repeatedly expand and contract, there is a definite trend in the average displacement magnitude as it increases and decreases. It appears to stabilize over time, but trend slightly closer to the origin as the particles arrange themselves more spherically (and therefore closer to the axes' origin).



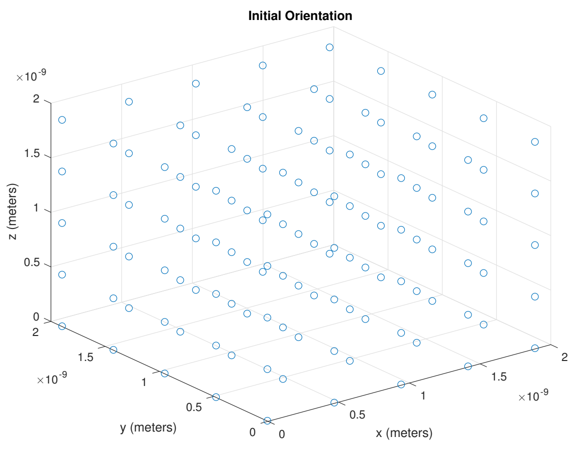
(a)

(b)

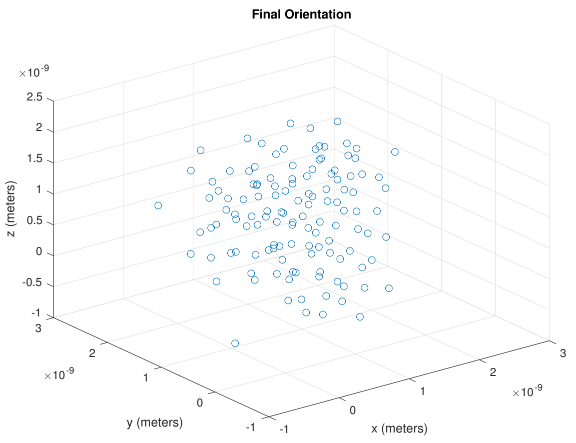
Figure 4: Plots of (a) total system energy, and (b) total momentum magnitude vs. time for a lattice of Xenon atoms in the Morse potential. These graphs were simulated with a temperature of 100K. The extremely stable results show that there is a variation $< 0.01\%$ for both energy and momentum.

increases (see figs. 7a and 7b). This is to be expected, as the Maxwell velocity distribution is merely a normal distribution with $\mu = 0$ and $\sigma = \sqrt{\frac{K_b T}{m}}$.

The direct result of this distribution is the evaporation of more atoms in the system, as shown in the figs. 8a, 8b, **??**, and **??**. The upward trend in the average displacement



(a)



(b)

Figure 5: 3D scatter plots of (a) initial system position, and (b) final system position. These graphs were simulated with a temperature of 100K. The system shifts from a lattice to a more energetically favorable spherical shape, as expected when below its boiling point.

indicates a trend of general evaporation upon initialization, where some atoms are initialized with enough energy to effectively escape the long-range attractive force of the other atoms in the system. This happens regardless of potential, although the Morse potential appears to exhibit evaporation at a slightly faster rate.

Another noteworthy effect of temperature is the order that appears in the structure as temperature is lowered and the Xenon shifts from gas, to liquid, to solid. To analyze this order, histograms of the final distance to neighbors in multiple simulations have been plotted in figs. 10a, 10b, 11a, and 11b. The sharp peaks throughout at low temperatures is caused by the presence of long range order in the solid state of the material, and then as the temperature rises above the melting and boiling points, the peaks flatten and the long range order dissipates. This follows naturally, as when the lattice is held at a lower temperature, it lacks the necessary kinetic energy to overcome the potential well and escape. As it heats, some atoms have a higher probability of overcoming that well and evaporating.

## Conclusions

The results of these simulations point to the viability of Verlet integration as a method of modeling the interactions between these atoms, provided an adequate time step. This is proved by the conservation of energy and momentum within a variance of 0.01% Both the Lennard-Jones and Morse potentials are well-shaped potential curves that cause slight oscillatory behavior in the lattice, as seen in the expansion-contraction cycles in the simulation results. [4, 2] The introduction of temperature variance in the lattice of Xenon atoms shows

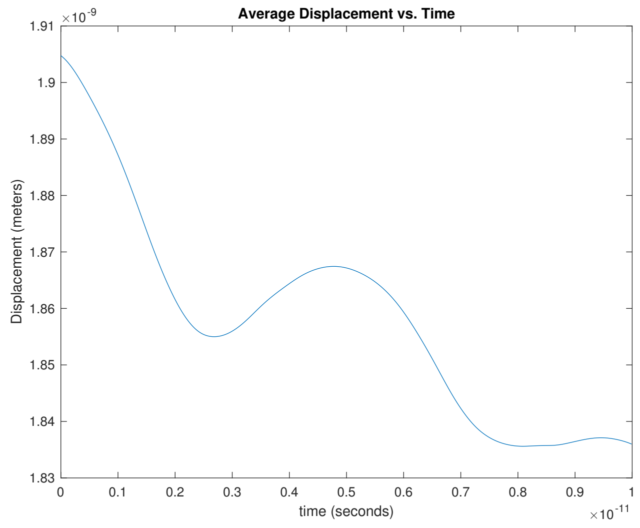


Figure 6: A plot of the average magnitude of the displacement from the origin over time for Xenon atoms in the Morse potential. As was seen in the Lennard-Jones potential, the atoms repeatedly expand and contract, creating a trend in the average displacement magnitude. It appears to stabilize over time, but trend slightly closer to the origin as the particles arrange themselves more spherically (and therefore closer to the axes' origin).

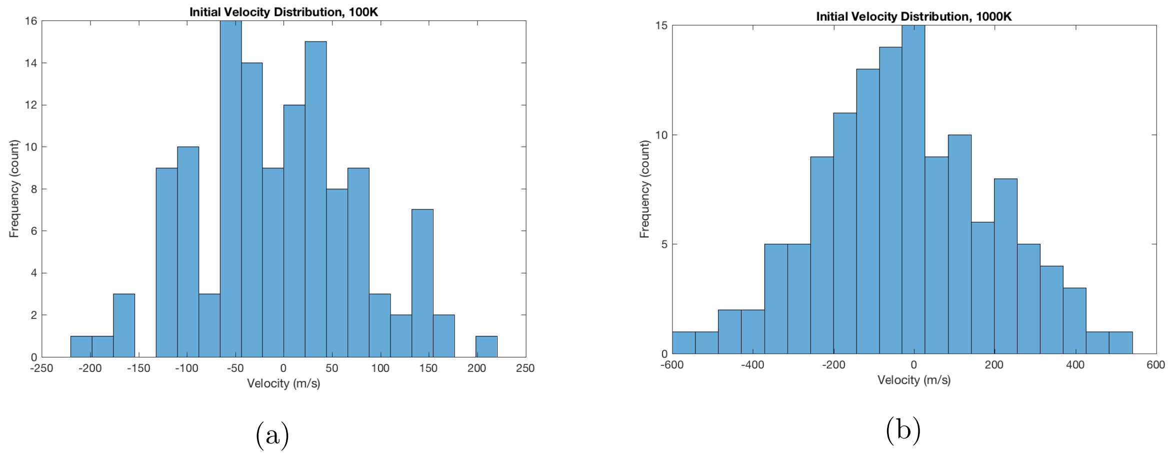
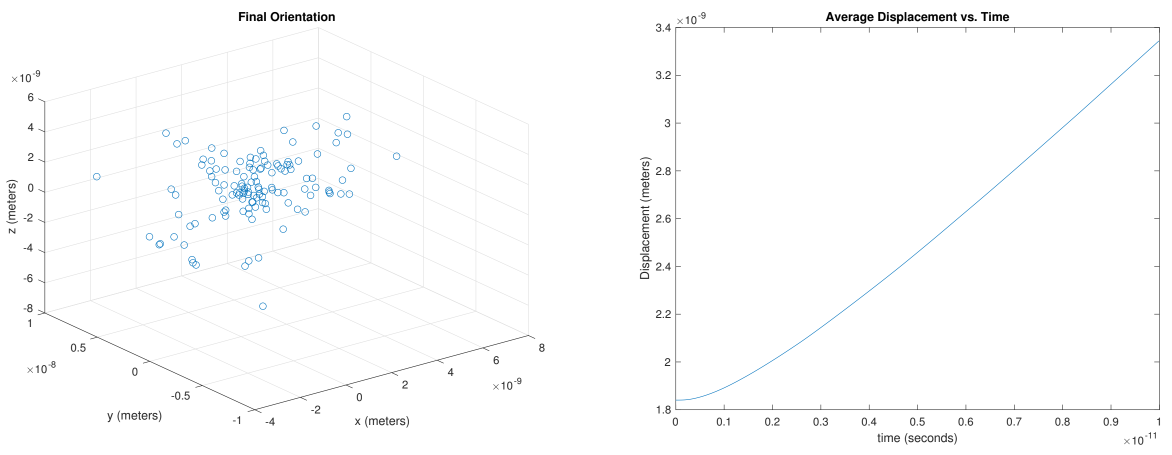


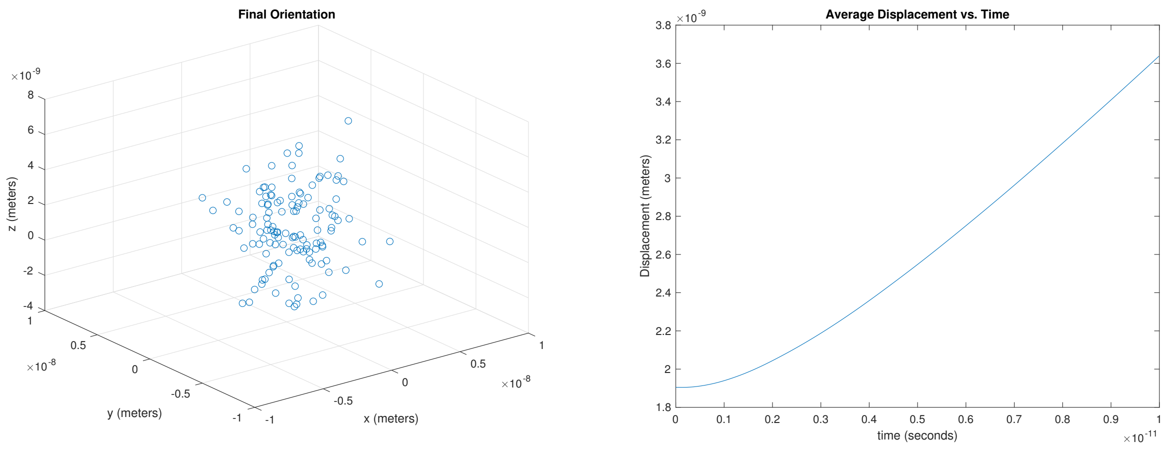
Figure 7: Histograms displaying the initial distributions of the initial x-component of the velocity at (a) 100K and (b) 1000K. These distributions are similar in shape, but as the temperature increases, the atoms are permitted to move faster initially. This higher initial velocity causes more evaporation of atoms.

the transition between solid, liquid, and gas phases. As temperature increases, more atoms evaporate, and the long range order evident in the solid dissipates.



(a) (b)

Figure 8: A 3D scatter plot of the final position (after $10^{-11}$ seconds) of a Xenon lattice in the Lennard-Jones potential (a) and the average magnitude of the displacement over time (b) when initialized to 1000K. It is clear from both plots the increased temperature causes more atoms to have enough energy to leave the initial lattice and evaporate.



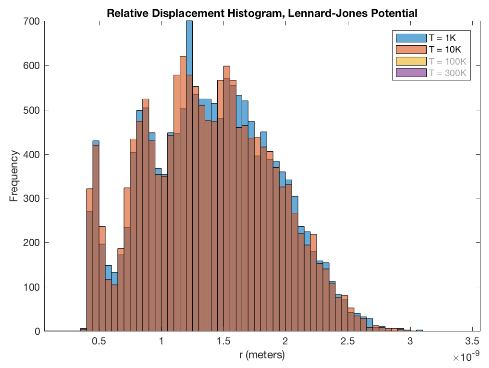
(a) (b)

Figure 9: A 3D scatter plot of the final position (after $10^{-11}$ seconds) of a Xenon lattice in the Morse potential (a) and the average magnitude of the displacement over time (b) when initialized to 1000K. It is clear from both plots the increased temperature causes more atoms to have enough energy to leave the system entirely and evaporate.

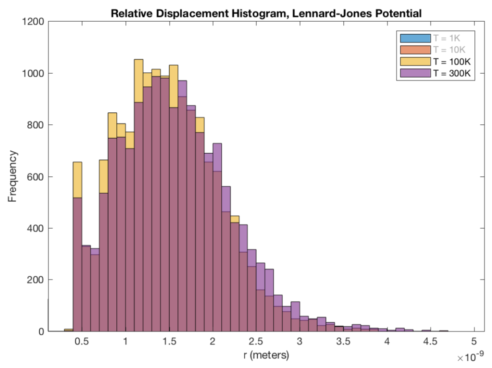
## Acknowledgements

I would like to thank Dr. Justin Oelgoetz for assistance formulating the simulation code, and for providing insightful explanations and troubleshooting along the way.

The "random_number" subroutine within the code was seeded using an algorithm pro-

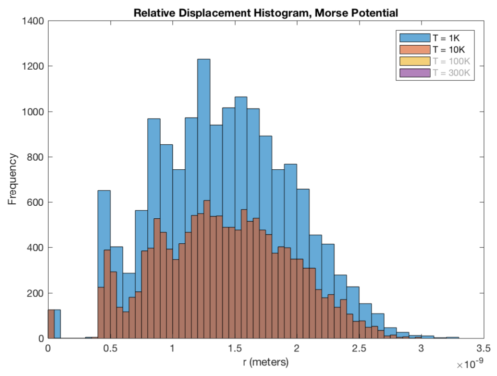


(a)

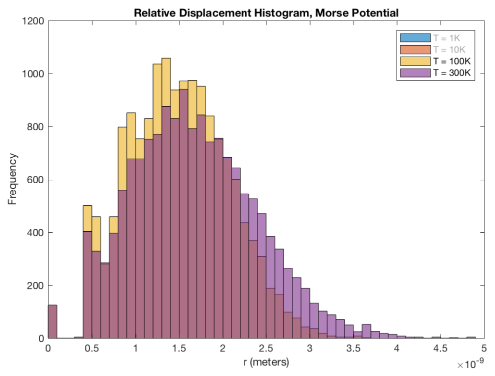


(b)

Figure 10: Histograms of the relative distance to all other particles in the system at (a) 1K and 10K, and (b) 100K and 300K in the Lennard-Jones potential. Definite peaks occur at low temperatures across short and long range, and as the element transitions to liquid and gaseous phases, the long range order dissipates.



(a)



(b)

Figure 11: Histograms of the relative distance to all other particles in the system at (a) 1K and 10K, and (b) 100K and 300K in the Lennard-Jones potential. Definite peaks occur at low temperatures across short and long range, and as the element transitions to liquid and gaseous phases, the long range order dissipates.

vided by Justin Oelgoetz, as sourced from open source code online. This code seeds Fortran's random number generator to create pseudo-random numbers that are unique to each run.

All graphs and plots made in this paper were produced using MATLAB computational software.

## References

[1] G. E. P. Box and Mervin E. Muller. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29(2):610–611, 06 1958.

[2] J. E. Jones. On the determination of molecular fields. ii. from the equation of state of a gas. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 106(738):463–477, 1924.

[3] Akira Matsumo. Parameters of the morse potential from second virial coefficients of gases. *Z. Naturforsch*, 42a, 1987.

[4] Philip M. Morse. Diatomic molecules according to the wave mechanics. ii. vibrational levels. *Phys. Rev.*, 34:57–64, Jul 1929.

[5] Loup Verlet. Computer ”experiments” on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.

[6] E. Whalley and W.G. Schneider. Intermolecular potentials of argon, krypton, and xenon. *The Journal of Chemical Physics*, 23, 1955.

## Appendix

Attached is the source code for this project, written in Fortran.

### Main

```
program nbody
use constants
use morse
use lennardjones
implicit none

real*8, parameter :: pi = acos(-1.0)
real*8, parameter :: kb = 1.38065D-23
real*8, parameter :: T = 100, dt = 1D-14
integer, parameter :: edgex = 5, edgey = 5, edgez = 5, tsteps =
   1000
integer, parameter :: nbodies = edgex*edgey*edgez
real*8, dimension(1:3,1:nbodies,0:tsteps+1) :: r = 0, v = 0, F = 0
real*8, dimension(3,0:tsteps) :: p = 0
real*8, dimension(0:tsteps) :: E = 0
integer :: i, j, k, it, counter, start, finish, persecond, time
real*8 :: average
real, dimension(3,2) :: rand
character :: pottype

! -O3 for third level optimization
!call system_clock(count_rate=persecond)
print*,'Lennard-Jones or Morse?'
print*,'(L/M)'
read(*,*) pottype

! PERFORM LENNARD-JONES POTENTIAL SIMULATION
if(pottype == 'L') then
print*,'Performing Lennard-Jones potential simulation...'
!do time = 1,10
```

```
! INITIALIZE r(:,:,0)
counter = 1
do i = 1, edgex
   do j = 1, edgey
      do k = 1, edgez
      r(1,counter,0) = (i-1)*rm
      r(2,counter,0) = (j-1)*rm
      r(3,counter,0) = (k-1)*rm
      counter = counter + 1
      end do
   end do
end do
print*,'Position␣initialized...'

! INITIALIZE v(:,:,0)
do i = 1, nbodies
   call random_number(rand)
   v(:,i,0) = sqrt(-2.0*Kb*T/m*log(rand(:,1)))*cos(2*pi*rand(:,2))
      *0.95
end do
print*,'Velocity␣initialized...'
!call system_clock(count=start)
! CALCULATE INITIAL FORCE
call ljforce(r(:,:,0), F(:,:,0)) !subroutine calculates all forces
    on all particles at time 0
call ljenergy(r(:,:,0), v(:,:,0), E(0))
call ljmomentum(v(:,:,0),p(:,0))
print*,'Force,␣energy,␣momentum␣initialized...'

! TIME LOOP
do it = 1,tsteps
r(:,:,it) = r(:,:,it-1) + v(:,:,it-1)*dt + (0.5/m)*F(:,:,it-1)*dt
   **2
call ljforce(r(:,:,it), F(:,:,it))
v(:,:,it) = v(:,:,it-1) + (F(:,:,it-1) + F(:,:,it))/(2*m)*dt
call ljenergy(r(:,:,it), v(:,:,it), E(it))
call ljmomentum(v(:,:,it),p(:,it))
end do ! End timesteps loop
!call system_clock(count=finish)
!average = average + (float(finish)-float(start))/float(persecond)
!end do
!print*,'Average computation time (seconds):', average/10


!PERFORM MORSE POTENTIAL SIMULATION
else if (pottype == 'M') then
print*,'Performing␣Morse␣potential␣simulation...'
!do time = 1,10

! INITIALIZE r(:,:,0)
counter = 1
do i = 1, edgex
   do j = 1, edgey
      do k = 1, edgez
      r(1,counter,0) = (i-1)*re
      r(2,counter,0) = (j-1)*re
      r(3,counter,0) = (k-1)*re
      counter = counter + 1
      end do
   end do
end do
print*,'Position␣initialized...'

! INITIALIZE v(:,:,0)
do i = 1, nbodies
   call random_number(rand)
   v(:,i,0) = sqrt(-2.0*Kb*T/m*log(rand(:,1)))*cos(2*pi*rand(:,2))
      *0.95
end do
print*,'Velocity␣initialized...'

!call system_clock(count=start)
```

```
! CALCULATE INITIAL FORCE, ENERGY, MOMENTUM
call morseforce(r(:,:,0), F(:,:,0)) !subroutine calculates all
   forces on all particles at time 0
call morseenergy(r(:,:,0), v(:,:,0), E(0))
call morsemomentum(v(:,:,0),p(:,0))
print*,'Force, energy, momentum initialized...'

! TIME LOOP
do it = 1,tsteps
r(:,:,it) = r(:,:,it-1) + v(:,:,it-1)*dt + (0.5/m)*F(:,:,it-1)*dt
   **2
call morseforce(r(:,:,it), F(:,:,it))
v(:,:,it) = v(:,:,it-1) + (F(:,:,it-1) + F(:,:,it))/(2*m)*dt
call morseenergy(r(:,:,it), v(:,:,it), E(it))
call morsemomentum(v(:,:,it),p(:,it))
end do ! End timesteps loop
!call system_clock(count=finish)
!average = average + (float(finish)-float(start))/float(persecond)
!end do
!print*,'Average computation time (seconds):', average/10
end if

! OUTPUT DATA
print*,'Writing data...'
call outputdata(r, v, F, E, P, nbodies, tsteps, dt, pottype)

end program nbody


!DATA OUTPUT SUBROUTINE
subroutine outputdata(r, v, F, E, P, nbodies, tsteps, dt, pottype)
integer :: nbodies, tsteps, it
real*8, dimension(3,1:nbodies,0:tsteps+1) :: r, v, F
real*8, dimension(3,0:tsteps) :: P
real*8, dimension(0:tsteps) :: E
real*8 :: dt
```

```
character :: pottype

open(unit = 100, file = 'pos.txt')
open(unit = 200, file = 'vel.txt')
open(unit = 300, file = 'force.txt')
open(unit = 400, file = 'E.txt')
open(unit = 500, file = 'P.txt')
open(unit = 600, file = 'lasttype.txt')

do it = 0, tsteps
do i = 1, nbodies
write(100,*) i, r(:,i,it), norm2(r(:,i,it)), it*dt !particle #, x,
     y, z, time
write(200,*) i, v(:,i,it), norm2(v(:,i,it)), it*dt !VEL
write(300,*) i, F(:,i,it), norm2(F(:,i,it)), it*dt !F
end do
write(400,*) it*dt, E(it) ! total E
write(500,*) it*dt, p(:,it), norm2(p(:,it)) !total P in each
    direction, and mag
end do
write(600,*) pottype
write(600,*) nbodies
write(600,*) tsteps

close(100)
close(200)
close(300)
close(400)
close(500)
close(600)

end subroutine

!RANDOM SEED SUBROUTINE
subroutine init_random_seed()
      integer :: i, n, clock
      integer, dimension(:), allocatable :: seed

      CALL RANDOM_SEED(size = n)
      allocate(seed(n))

      CALL SYSTEM_CLOCK(COUNT=clock)

      seed = clock + 37 * (/ (i - 1, i = 1, n) /)
      CALL RANDOM_SEED(PUT = seed)

      deallocate(seed)
end subroutine

!RANDOM INDEX FUNCTION
integer function randomindex(length)
      real :: random
    call random_number(random)
      randomindex = floor(random*length + 1.0)
return
end
```

## Morse Potential Module

```
module morse
!MORSE POTENTIAL MODULE
use constants
implicit none
contains

!FORCE SUBROUTINE
!At a given timestep, calculates the total force in each direction
    for all particles in system
subroutine morseforce(r, F)
integer :: nbodies
real*8, intent(in), dimension(:,:) :: r
real*8, intent(out), dimension(:,:) :: F ! first dimension is x, y
   , z, second is particle #
```

```
integer ::  ipart, jpart, i
real*8 :: length, fmag

nbodies = size(r,2)

do ipart = 1, nbodies
   do jpart = ipart+1, nbodies
   !DISTANCE BETWEEN IPART and JPART
   length = norm2(r(:,ipart) - r(:,jpart))
   !MAGNITUDE of FORCE from POSITIONS
   fmag = -2.0*D*a*dexp(a*(re-length))*(1-dexp(a*(re-length)))
   !COMPONENTS of FORCE on IPART from JPART
   F(:,ipart) = F(:,ipart) + fmag*(r(:,ipart)-r(:,jpart))/length
   F(:,jpart) = F(:,jpart) - F(:,ipart)
   end do
end do
end subroutine

!POTENTIAL ENERGY SUBROUTINE
!At a given timestep, calculates all potential energy
   contributions for all particles in system
subroutine morsepotential(r, U)
real*8, dimension(:,:) :: r ! first dimension is x, y, z, second
   is particle #
integer :: ipart, jpart, nbodies
real*8 :: length, U
nbodies = size(r,2)
do ipart = 1, nbodies
   do jpart = ipart+1, nbodies
   length = norm2(r(:,ipart) - r(:,jpart))
   U = U + D*(1-dexp(-a*(length-re)))**2
   end do
end do
end subroutine

!MOMENTUM SUBROUTINE
```

```
!At a given timestep, sums momentum from all particles in system
subroutine morsemomentum(v, p)
real*8, dimension(:,:) :: v
real*8, dimension(3) :: p
integer :: nbodies, ipart
nbodies = size(v,2)
do ipart = 1,nbodies
   p(:) = p(:) + m*v(:, ipart)
end do

end subroutine

!ENERGY SUBROUTINE
!At a given timestep, calculates the total energy in the system
subroutine morseenergy(r, v, E)
real*8, dimension(:,:) :: r, v
integer :: nbodies,  i
real*8 :: E
nbodies = size(v,2)

call morsepotential(r, E) !Stores potential of system in E

do i = 1,nbodies
   E = E + 0.5*m*norm2(v(:,i))**2 !Adds KE of each particle
end do



end subroutine


end module
```

## Lennard-Jones Potential Module

```
module lennardjones
!LENNARD JONES POTENTIAL MODULE
use constants
implicit none
```

```
contains

!FORCE SUBROUTINE
!At a given timestep, calculates all components of force between
   all particles in system
subroutine ljforce(r, F)
real*8, dimension(:,:) :: r, F ! first dimension is x, y, z,
   second is particle #
integer :: nbodies, ipart, jpart
real*8 :: length, fmag
nbodies = size(r,2)

do ipart = 1, nbodies
   do jpart = ipart+1, nbodies
   !DISTANCE BETWEEN IPART and JPART
   length = norm2(r(:,ipart) - r(:,jpart))
   !MAGNITUDE of FORCE from POSITIONS
   fmag = eps*(12/rm)*((rm/length)**13-(rm/length)**7)
   !COMPONENTS of FORCE on IPART from JPART
   F(:,ipart) = F(:,ipart) + fmag*(r(:,ipart)-r(:,jpart))/length
   F(:,jpart) = F(:,jpart) - F(:,ipart)
   end do
end do
end subroutine

!POTENTIAL ENERGY SUBROUTINE
!At a given timestep, calculates all potential energy
   contributions for all particles in system
subroutine ljpotential(r, U)
real*8, dimension(:,:) :: r ! first dimension is x, y, z, second
   is particle #
integer :: ipart, jpart, nbodies
real*8 :: length, U
nbodies = size(r,2)

do ipart = 1, nbodies
```

```
   do jpart = ipart+1, nbodies
   length = norm2(r(:,ipart) - r(:,jpart))
   U = U + eps*((rm/length)**12-2.0*(rm/length)**6)
   end do
end do
end subroutine



subroutine ljmomentum(v, p)
real*8, dimension(:,:) :: v
real*8, dimension(3) :: p
integer :: nbodies, ipart
nbodies = size(v,2)

do ipart = 1,nbodies
   p(:) = p(:) + m*v(:, ipart)
end do

end subroutine



subroutine ljenergy(r, v, E)
real*8, dimension(:,:) :: r, v
integer :: nbodies, i
real*8 :: E
nbodies = size(r,2)

call ljpotential(r, E) !Stores potential of system in E

do i = 1,nbodies
   E = E + 0.5*m*norm2(v(:,i))**2 !Adds KE of each particle
end do

end subroutine

end module
```

## Constants Module

```
module constants
real*8, parameter :: re = 4.73D-10 !Meters
real*8, parameter :: D = 3.13222D-21 !Joules
real*8, parameter :: a = 1.099D10  !1/Meters
!PARAMETERS for XENON in L-J POTENTIAL
real*8, parameter :: rm = 4.57D-10
real*8, parameter :: eps = 3.106D-21
real*8, parameter :: m = 2.1807D-25

end module constants
```

## MATLAB Plotting Script

Code can be run while MATLAB is in the directory containing result files and will create a .gif file of positions, total energy and momentum plots, an initial velocity sampling histogram, average displacement magnitude vs. time, and initial and final scatter plots of position and save them to the directory.

```
limits = [-0.5;1;-0.5;1;-0.5;1];
limits = (limits)*0.5E-8;
lastrun = importdata('lasttype.txt');  %Which potential data?
pottype = cell2mat(lastrun.textdata);
pnum = lastrun.data(1);
tsteps = lastrun.data(2);
dogif = 1;  %Do the gif?

if pottype == 'L'
fprintf('Plotting L-J Potential \n')
filename = 'LJposition.gif';
else
fprintf('Plotting Morse Potential \n')
filename = 'Mposition.gif';
end
R = importdata('pos.txt');
tsteps = 1000;
```

```
initial = figure;
scatter3(R(1:pnum,2),R(1:pnum,3),R(1:pnum,4));
title('Initial␣Orientation');
xlabel('x␣(meters)');
ylabel('y␣(meters)');
zlabel('z␣(meters)');
if pottype == 'L'
saveas(initial,'LJinitial','epsc');
else
saveas(initial,'Minitial','epsc');
end

final = figure;
scatter3(R((tsteps)*pnum + 1:(tsteps+1)*pnum,2),R(tsteps*pnum +
    1:(tsteps+1)*pnum,3),R(tsteps*pnum + 1:(tsteps+1)*pnum,4));
title('Final␣Orientation');
xlabel('x␣(meters)');
ylabel('y␣(meters)');
zlabel('z␣(meters)');
if pottype == 'L'
saveas(final,'LJfinal','epsc');
else
saveas(final,'Mfinal','epsc');
end

if dogif == 1
h = figure;
for i = 1:tsteps
%figure()
scatter3(R(((i-1)*pnum + 1):(i*pnum),2),R(((i-1)*pnum + 1):(i*pnum
    ),3),R(((i-1)*pnum + 1):(i*pnum),4));
title(num2str(i))
axis(limits)
drawnow
% Capture the plot as an image
frame = getframe(h);
```

```
im = frame2im(frame);
[imind,cm] = rgb2ind(im,256);
% Write to the GIF File
if i == 1
imwrite(imind,cm,filename,'gif', 'Loopcount',inf,'DelayTime',0.05)
    ;
else
imwrite(imind,cm,filename,'gif','WriteMode','append','DelayTime'
    ,0.05);
end
end
end

E = importdata('E.txt');
P = importdata('P.txt');
V = importdata('vel.txt');

eplot = figure;
plot(E(:,1),E(:,2));
title('Total␣Energy␣vs.␣Time');
xlabel('time␣(seconds)');
ylabel('Energy␣(Joules)');
if pottype == 'L'
saveas(eplot,'LJEvsT','epsc');
else
saveas(eplot,'MEvsT','epsc');
end

pplot = figure;
plot(P(:,1),P(:,5));
title('Total␣Momentum␣vs.␣Time');
xlabel('time␣(seconds)');
ylabel('Momentum␣(kg*m/s)');
if pottype == 'L'
saveas(pplot,'LJPvsT','epsc');
else
```

```
saveas(pplot,'MPvsT','epsc');
end

vhist = figure;
histogram(V(1:125,2),20);
title('Initial␣Velocity␣Distribution');
xlabel('Velocity␣(m/s)');
ylabel('Frequency␣(count)');
saveas(vhist,'initialvelocity','epsc');

rave = zeros(tsteps,2);
for i = 1:tsteps
rave(i,2) = sum(R(((i-1)*pnum + 1):(i*pnum),5))/pnum;
rave(i,1) = R((i-1)*pnum + 1,6);
end
raver = figure;
plot(rave(:,1),rave(:,2));
title('Average␣Displacement␣vs.␣Time');
xlabel('time␣(seconds)');
ylabel('Displacement␣(meters)');
if pottype == 'L'
saveas(raver,'LJaverager','epsc');
else
saveas(raver,'Maverager','epsc');
end

adj = zeros(pnum,pnum);
for i = 1:pnum
for j = 1:pnum
    adj(i,j) = sqrt((R(tsteps*pnum+i,2)-R(tsteps*pnum+j,2))^2 + (R(
        tsteps*pnum+i,3)-R(tsteps*pnum+j,3))^2 + (R(tsteps*pnum+i,4)
        -R(tsteps*pnum+j,4))^2);
end
end
figure()
histogram(adj);
```

```
title('Final Relative Displacement');
xlabel('r (meters)');
ylabel('Frequency');
```