Modeling Atomic Interactions in Xenon Using the Morse and Lennard-Jones Potentials with Fourth-Order Runge-Kutta Integration

Jonathan Bunton

October 23, 2017

Abstract

Inter-atomic forces are often modeled using potential well models, characterized by an attractive long range force and a stronger repulsive short range force. Two such models are the Lennard-Jones and Morse potentials, each of which exhibit this behavior with differing mathematical premises. The Lennard-Jones potential accomplishes this with two terms:

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

Here r is the distance between particles, and the parameters ε and r_m determine the depth of the potential curve and its lowest point respectively. [2] The parameters for these values are characteristic to a material, in this paper's case, Xenon. The Morse potential achieves a similar shape using exponentials:

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

In the Morse potential, the parameters D_e and a control the depth and width of the potential well respectively, while r_e is the lowest point. [5] Again these coefficients are characteristic of the material, Xenon for this paper. Despite very different mathematical formulation, these potentials have very similar shapes.

This paper analyzes the behavior of a lattice of Xenon atoms in both the Lennard-Jones and the Morse potentials by using fourth-order Runge-Kutta (RK4) integration methods to numerically calculate the equations of motion for the system. [3] The coefficients for these potentials are experimentally determined from the second virial coefficient. [4, 6] Fourth-order Runge-Kutta shows high precision, at the cost of computation time. Compared to a simpler method, such as Verlet integration, our data shows an average of a 300% increase in calculation time from Verlet integration to RK4. As expected, the simulation data also exhibits the spring-esque behavior suggested by

the potential wells, as well as increased evaporation and decreased order with a higher temperature.

Results

Both the Lennard-Jones and Morse potentials were simulated using code developed in previous assignments, tailored to the RK4 algorithm. This code initializes the velocities in each direction using the Maxwell-Boltzmann distribution in each dimension.

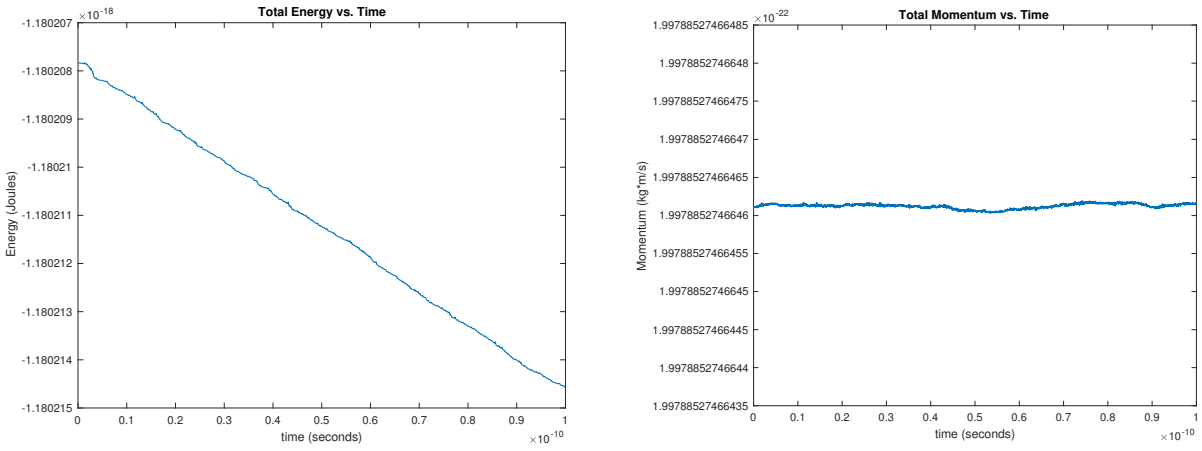
As was shown in the case of Verlet integration, the system proved semi-stable for both potentials, with a slight oscillatory behavior as expected from the potential shapes. Ultimately, the steady-state solution appeared to be a sphere, where each particle is spaced at an adequate distance to be "trapped" in the curve of the potential well. In addition, at lower temperatures clusters appear where groups of particles sit in the minimum potential position (r_m or r_e) in smaller groups after leaving the larger initial lattice.

Computationally, the data shows a more precise conservation of energy and momentum, both of which ought to remain constant throughout calculations. Variation in energy through 10000 timesteps of 10^{-14} is on the order of 0.001% and $< 10^{-10}$% in momentum. The error of the RK4 algorithm appears to lose total energy slightly through iterations, which is likely caused by low roundoff errors over the averaging period and numerical issues involving time step size selection. Because momentum and energy are conserved so incredibly precisely, we can consider RK4 a valid numerical approximation method for this system.

Lennard-Jones Potential

The Lennard-Jones (6-12) potential creates a semi-stable system. Our simulation results conserve energy and momentum while settling into a steady state orientation, 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 0.001% in total energy and $< 10^{-10}$% in total momentum. These percentages hold for timesteps of 10^{-14}, but smaller timesteps of course provide higher precision. Conversely, larger timesteps result in too drastic of changes in the system between timestep iterations for momentum and energy to be effectively conserved.

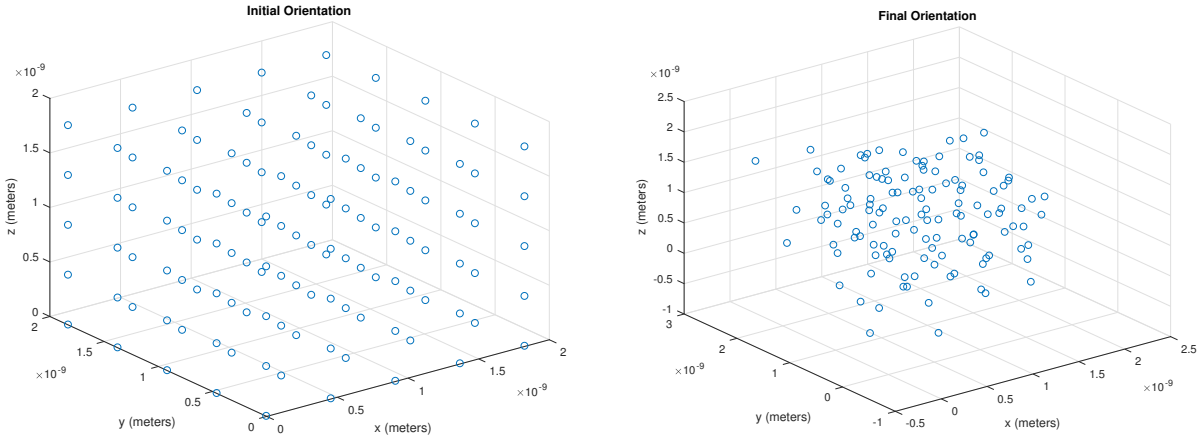
Shifting instead to the resulting position data, the initial and final states of the system are shown in figs. 2a and 2b. In this potential, the atoms are initialized to a lower velocity due to the low temperature (100K), and thus behave similarly to if they were to begin with zero velocity. In the initial lattice, each atom is at a minimum r_e distance apart, causing



(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.

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 clear through the plot in fig. 3, where the average displacement magnitude shows slight signs of oscillation over time as it steadily moves towards the stable orientation of particles.



(a) (b)

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, with very little evaporation.

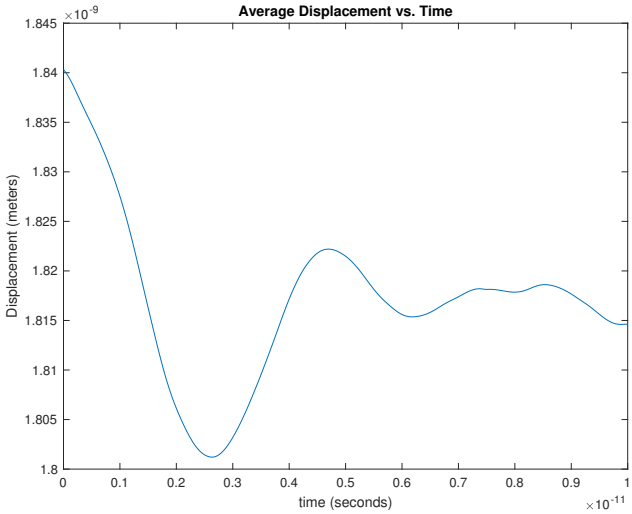
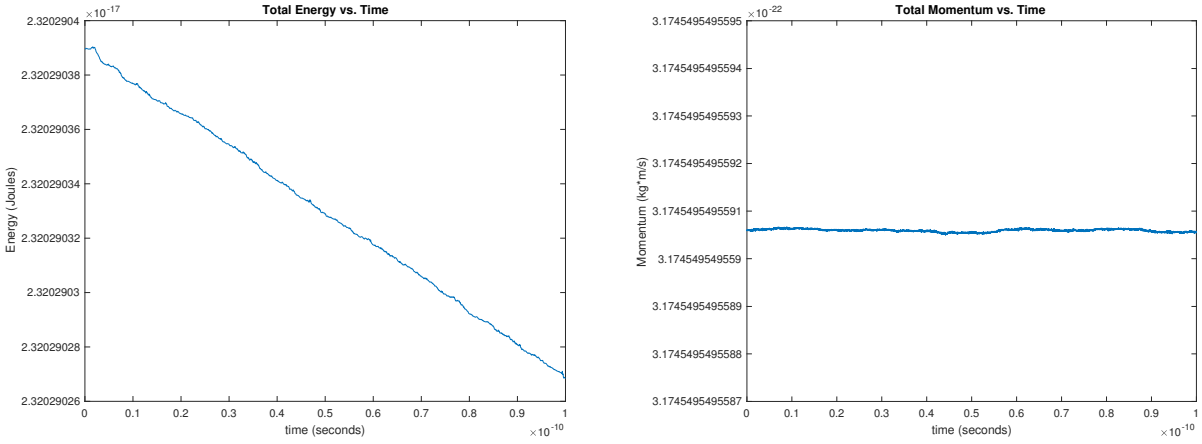


Figure 3: A plot of the average magnitude of the displacement from the origin over time for Xenon atoms in the Lennard-Jones potential. As the atoms repeatedly expand and contract, there is a clear damped oscillation in the average displacement as the system settles to its final orientation.

Morse Potential

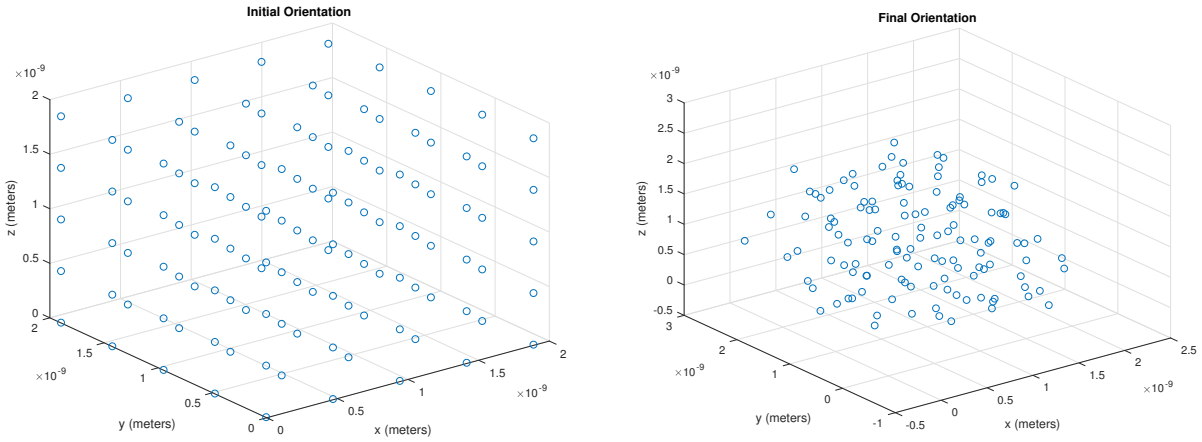
In the Morse potential, the spring-esque behavior within the potential is again very evident. The energy and momentum of the system are conserved with comparable accuracy on the order of $< 0.001\%$, as seen in figs 4a and 4b. This once again validates RK4, provided an acceptable timestep.



(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 $< 0.001\%$ for both energy and momentum in 10000 iterative steps.

The results of the Morse potential simulation are strikingly similar to those in the Lennard-Jones, as would be expected by the similar shapes. Because each atom is initialized 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 the contraction-expansion cycle 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 clear in fig. 6, which displays the average magnitude of the displacement through the 10000 iterative steps.



(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 with little evaporation at this temperature, as 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 increases. 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 is the evaporation of more atoms in the system as temperature increases, as shown in the figs. 7a, 7b, **??**, and **??**. The upward trend in the average displacement indicates a trend of general evaporation upon initialization, where most atoms are initialized with enough kinetic energy to effectively "escape" the long-range attractive force of the other atoms in the system. Both Morse and Lennard-Jones exhibit this behavior, however, the

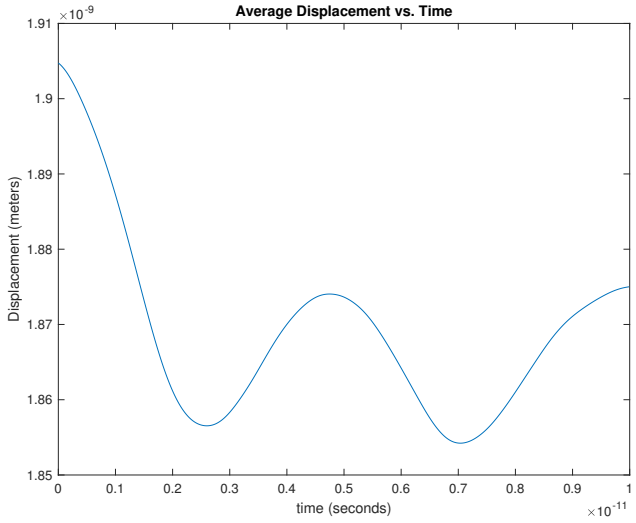
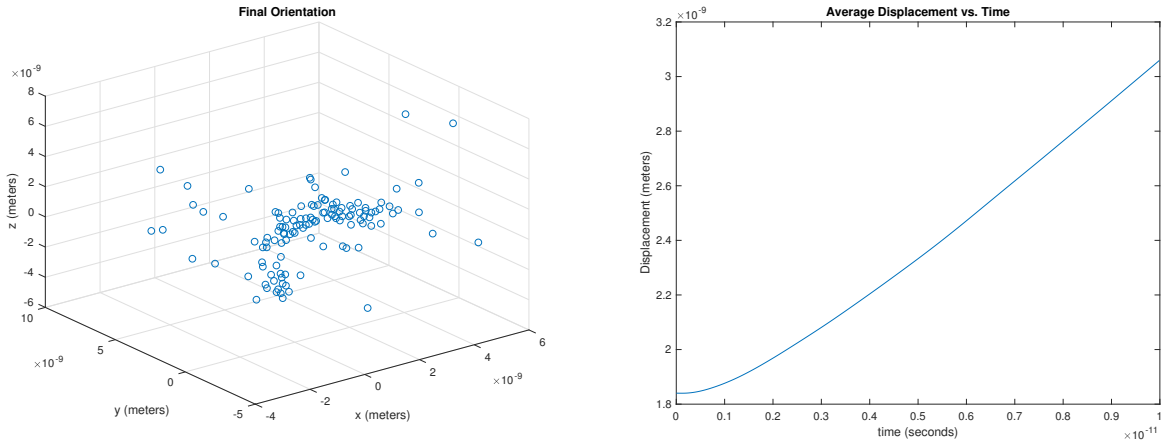


Figure 6: A plot of the average magnitude of the displacement from the origin over time for Xenon atoms in the Morse potential. As was seen in the Lennard-Jones potential, the atoms repeatedly expand and contract, creating an oscillation in the average displacement magnitude.

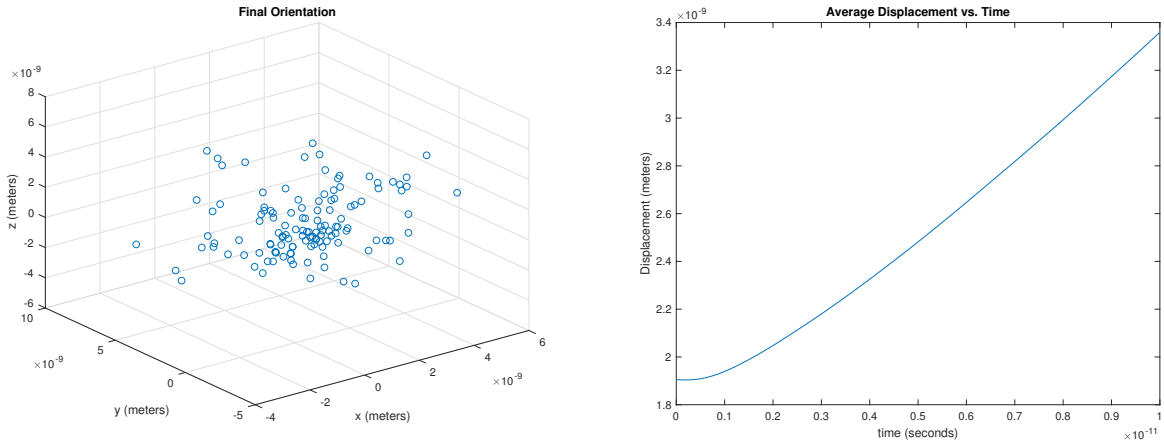
Morse potential shows faster evaporation, likely caused by the flexibility in width parameter a.



(a) (b)

Figure 7: 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.

Another noteworthy effect of temperature is the structural order that appears at lower temperature. As Xenon shifts from gas, to liquid, to solid, it gains more structural order, and



(a) (b)

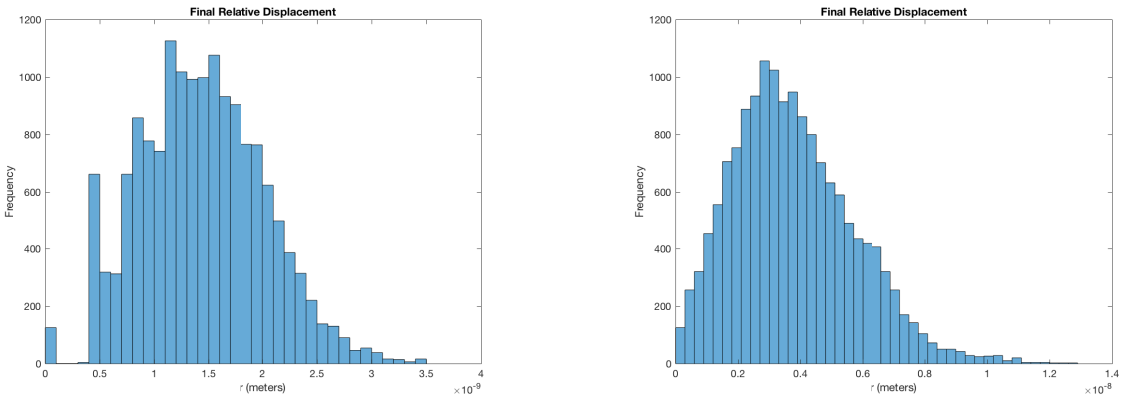
Figure 8: A 3D scatter plot of the final position (after 10^{-11} seconds) of a Xenon lattice in the Morse potential (a) and the average magnitude of the displacement over time (b) when initialized to 1000K. It is clear from both plots the increased temperature causes more atoms to have enough energy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

therefore has an inverse relationship with temperature. To analyze this order, histograms of the final distance to neighbors in multiple simulations have been plotted in figs. 9a, 9b, 10a, and 10b. These histograms effectively plot the distribution of distances between each atom and its neighbors. The sharp peaks throughout at low temperatures are caused by lower temperatures creating lower kinetic energies and therefore a lower probability of escaping the potential wells. As the system is heated, some atoms have a higher probability of enough kinetic energy (velocity) to overcome the potential well and evaporate, eliminating the short range order at higher temperatures..

Comparison to Verlet Integration

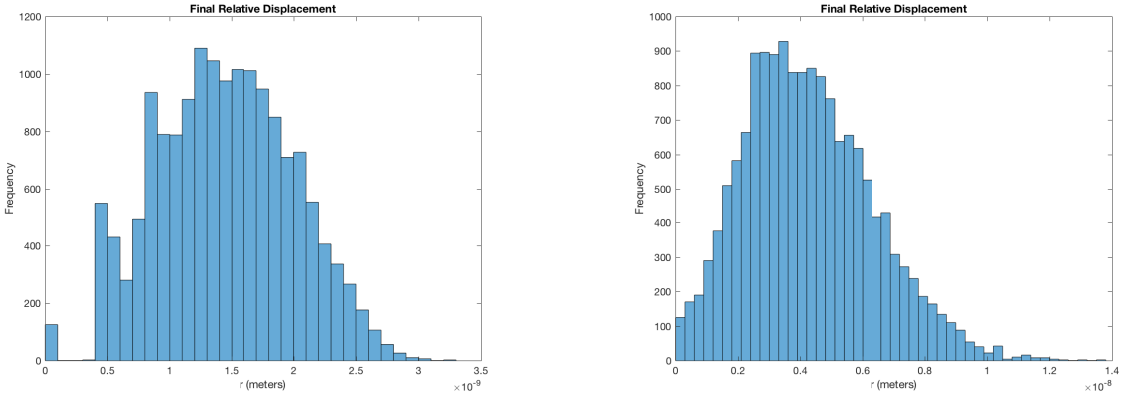
Fourth-order Runge-Kutta, while highly precise, is incredibly computationally exhaustive. Where simpler methods such as Verlet integration only involve performing full iterative calculations of the force and positions twice per timestep, each run through the RK4 algorithm involves averaging three additional values each with new force and position calculations.

This repetition in mind, the average runtime of the RK4 algorithm is approximately three times longer than the runtime of the previously written Verlet integration method for solving the same system's equations of motion. [1] Both codes have been modified to output the total time used in calculation before and after the algorithms begin by calling the 'system_clock' subroutine before and after calculation. The subroutine also provides the number of 'ticks' per second, and this value is used to find the total runtime of each program



(a) (b)

Figure 9: Histograms of the relative distance to all other particles in the system at (a) 100K, and (b) 1000K 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 10: Histograms of the relative distance to all other particles in the system at (a) 100K, and (b) 1000K in the Morse 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.

in seconds.

The average runtimes are compared in **??**. The Morse potential generally takes longer to compute, likely because of the double-precision exponential functions that are required in the calculation. As expected, the RK4 algorithm takes on the order of 300% the amount of time as the Verlet integration algorithm. Arguably the improvement in precision does not merit the time in this scenario, however, in cases where small error can quickly propogate over longer iteration times, RK4 may be necessary.

	Lennard-Jones Runtime (s)	Morse Runtime (s)
RK4	23.334	25.926
Verlet	8.422	7.322

Table 1: A table detailing the average runtime of n-body simulations using both RK4 and Verlet integration methods. The averages were collected over 10 runs each using 10000 timesteps of 10^{-14}. Simulation runs were performed back to back, with no change in background programs running on system between simulations.

Discussion

The results of these simulations justify the use of RK4 as an integration method, provided an adequate time step. There is of course numerical error associated with the calculations, however the variance in energy below 0.01% gives evidence to the method's overall accuracy. RK4 also provides more precise results than Verlet integration, at the cost of high computation times. [1] Both the Lennard-Jones and Morse potentials induce the expansion-contraction cycles in the simulation results, causing a similar system to bodies connected by springs. [5, 2] The introduction of temperature dependence of the initial velocities for the lattice of Xenon atoms shows the transition between ordered (solid) and unordered (liquid, gaseous) phases of the material.

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 provided 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] Jonathan Bunton. Modeling interatomic interactions using the morse and lennard-jones potentials. Previous Submission, October 2017.

[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] Martin W. Kutta. Beitrag zur näherungsweisen integration totaler differentialgleichungen. *Zeitschrift für Methematik und Physik*, 1901.

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

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

[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 nbodyrk4
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 =
   10000
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
real*8, dimension(3,1:nbodies,4) :: KR = 0, KV = 0
real*8 :: average
integer :: i, j, k, it, counter, start, finish, persecond, time
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 #', time
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 ENERGY and MOMENTUM
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
   call ljforce(r(:,:,it-1),F(:,:,it-1))
   !Calculate K1s
   KV(:,:,1) = dt*F(:,:,it-1)/m
   KR(:,:,1) = dt*V(:,:,it-1)
   !Calculate K2s
   call ljforce((r(:,:,it-1)+KR(:,:,1)/2),KV(:,:,2))
```

```
    KV(:,:,2) = dt*KV(:,:,2)/m
    KR(:,:,2) = dt*(V(:,:,it-1)+KV(:,:,1)/2)
    !Calculate K3s
    call ljforce((r(:,:,it-1)+KR(:,:,2)/2),KV(:,:,3))
    KV(:,:,3) = dt*KV(:,:,3)/m
    KR(:,:,3) = dt*(V(:,:,it-1)+KV(:,:,2)/2)
    !Calculate K4s
    call ljforce((r(:,:,it-1)+KR(:,:,3)),KV(:,:,4))
    KV(:,:,4) = dt*KV(:,:,4)/m
    KR(:,:,4) = dt*(V(:,:,it-1)+KV(:,:,3))
    !Average all four Ks
    v(:,:,it) = v(:,:,it-1) + (1.0/6.0)*(KV(:,:,1)+2*KV(:,:,2)+2*KV
        (:,:,3)+KV(:,:,4))
    r(:,:,it) = r(:,:,it-1) + (1.0/6.0)*(KR(:,:,1)+2*KR(:,:,2)+2*KR
        (:,:,3)+KR(:,:,4))
    KV(:,:,:) = 0
    KR(:,:,:) = 0
    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 ENERGY and MOMENTUM
call morseenergy(r(:,:,0), v(:,:,0), E(0))
call morsemomentum(v(:,:,0),p(:,0))
print*,'Energy, momentum initialized...'

! TIME LOOP
do it = 1,tsteps
    call morseforce(r(:,:,it-1),F(:,:,it-1))
    !Calculate K1s
    KV(:,:,1) = dt*F(:,:,it-1)/m
    KR(:,:,1) = dt*V(:,:,it-1)
    !Calculate K2s
    call morseforce((r(:,:,it-1)+KR(:,:,1)/2),KV(:,:,2))
    KV(:,:,2) = dt*KV(:,:,2)/m
    KR(:,:,2) = dt*(V(:,:,it-1)+KV(:,:,1)/2)
    !Calculate K3s
    call morseforce((r(:,:,it-1)+KR(:,:,2)/2),KV(:,:,3))
    KV(:,:,3) = dt*KV(:,:,3)/m
```

```
    KR(:,:,3) = dt*(V(:,:,it-1)+KV(:,:,2)/2)
    !Calculate K4s
    call morseforce((r(:,:,it-1)+KR(:,:,3)),KV(:,:,4))
    KV(:,:,4) = dt*KV(:,:,4)/m
    KR(:,:,4) = dt*(V(:,:,it-1)+KV(:,:,3))
    !Average all four Ks
    v(:,:,it) = v(:,:,it-1) + (1.0/6.0)*(KV(:,:,1)+2*KV(:,:,2)+2*KV
        (:,:,3)+KV(:,:,4))
    r(:,:,it) = r(:,:,it-1) + (1.0/6.0)*(KR(:,:,1)+2*KR(:,:,2)+2*KR
        (:,:,3)+KR(:,:,4))
    KV(:,:,:) = 0
    KR(:,:,:) = 0
    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-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 nbodyrk4


!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*(re-length)))**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 = 0; %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');
```