Long-term Conservation of Geometric Invariants by Newmark Beta methods

Numerical Computation Structured Projects

MMathCompSci Part B

Candidate Number: 1006833

Hilary Term, 2018

Contents

1 Introduction **4**

2 Newmark Beta Methods **6**
2.1 Introduction to Newmark Beta Methods 6
2.2 Computational Application of Newmark Beta Methods 7
2.3 Convergence Tests 8

3 Molecular Dynamics **10**
3.1 Hamiltonian Equation 10
3.2 Lennard-Jones Potential 10
3.3 Forces 11
3.4 Reduced Units 11
3.5 Initializing Positions 12
3.6 Initializing Velocities 13
3.7 Periodic Boundary Conditions and Minimum Image Convention 14
3.8 Cut-off Potential and Neighbour Lists 15
3.9 Computing the Jacobian 17

4 Experiment **19**
4.1 Time Reversibility 19
4.2 Total Linear Momentum 22
4.3 Total Angular Momentum 23
4.4 Total Energy 25

5 Conclusion **31**

A Appendix **32**
A.1 Lotka-Volterra Solutions Code 32
A.2 Convergence Tests on Pendulum Problem Code 34
A.3 Lennard-Jones Potential Graph Code 36
A.4 Molecular Dynamics: Implicit Methods Code 37

List of Figures

1.1 Solutions to the Lotka-Volterra Equation 4
2.1 Excerpt on applying the Newmark Beta methods computationally 7
2.2 Order of convergence of Newmark Beta methods for varying parameter values 9
3.1 Lennard-Jones Potential 10
3.2 Initial cubic lattice 13
3.3 Change in the position of particles after 500 steps of equilibration 14
3.4 Periodic Boundary Conditons in 2-d 15
4.1 Changes in position and velocities for $\beta = 0.00$ and $\gamma = 0.50$ after reversing 500 steps 21
4.2 Changes in position and velocities for $\beta = 0.00$ and $\gamma = 0.75$ after reversing 500 steps 21
4.3 Difference in behaviour of total energy for $\beta = 0.00$ and $\gamma = 0.75$ over different timesteps 21
4.4 Changes in position and velocities for $\beta = 0.25$ and $\gamma = 0.50$ after reversing 125 steps 22
4.5 Energy time series for two explicit Newmark Beta methods 26
4.6 Energy time series for Velocity Verlet method 27
4.7 Energy time series for implicit Newmark Beta methods with $\gamma = 0.50$ . . 27
4.8 Energy time series for Newmark Beta methods with $\beta = 0.25$ and $\gamma = 0.50$ under unmodified initial configurations 29
4.9 Energy time series for Newmark Beta methods with $\beta = 0.10$ and $\gamma = 0.50$ under unmodified initial configurations 29
4.10 Energy time series for Newmark Beta methods with $\beta = 0.40$ and $\gamma = 0.50$ under unmodified initial configurations 30

List of Tables

3.1 Reduced Units and their Values 12
3.2 Results of tests to find optimal N_n 16
4.1 Initial Configuration for molecular simulations 19
4.2 Total angular momentum of the system over the first 50 iterations 25

1 Introduction

Geometric Numerical Integration, when considered on the grand timeline of Mathematics, is a field still in its younger days. While work on the numerical treatment of differential equations had started off towards the end of the 19^{th} Century, it wasn't till the 1980s that numerical analysts shifted their focus onto Geometric Numerical Integration. A better part of the century had been involved in improving quantity - producting methods that could produce accurate results without being computationally strenuous. But the interest in running computer simulations over longer time intervals revealed that quality was equally important. Most methods could accurately reflect properties of a system, such as its asymptotic time behaviour or its sensitivity to changes in initial conditions; not all could preserve attributes such as total enery or momentum - properties that are invariants to the flow of the differential equation.

Consider, for example [4], the following set of Lotka-Volterra prey-predator equations

$$\frac{du}{dt} = u - uv, \qquad \frac{dv}{dt} = uv - 2v, \tag{1.1}$$

where $u(t)$ is the population of the prey over time, and $v(t)$ is that of the predator. Dividing the two equations in 1.1, we get that

$$\left(1 - \frac{2}{u}\right) du = \left(\frac{1}{v} - 1\right) dv.$$

Integrating the two sides of the equation, we get that

$$I(u, v) = u - 2\ln u + v - \ln v = const, \tag{1.2}$$

giving us that the solutions to 1.1 lie on the level curves of 1.2.

We solve the differential equation 1.1 with initial points $(u_0, v_0) = (2, 4)$ and $h = 0.02$ using three different numerical methods: Explicit Euler, Implicit Euler, and Symplectic Euler. The results can be found in Figure 1.1.

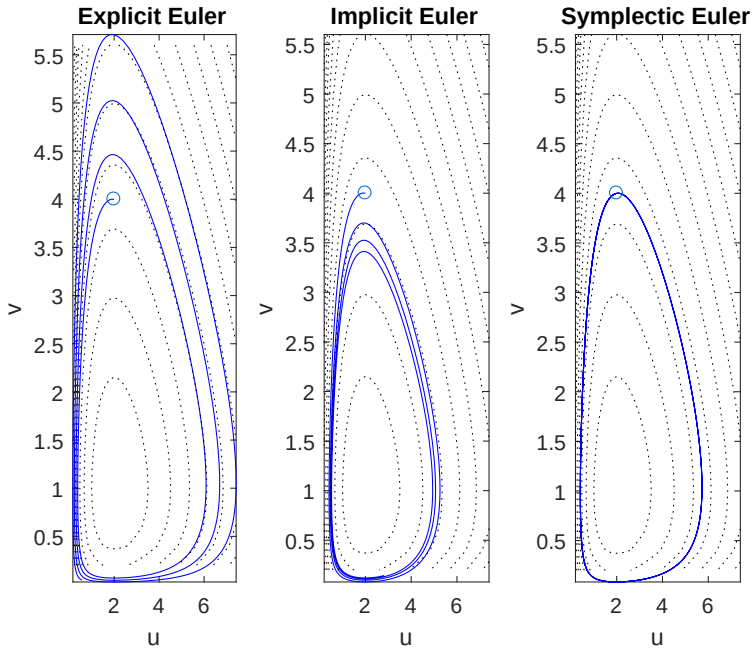

Figure 1.1: Solutions to the Lotka-Volterra Equation

The phase planes in Figure 1.1 (denoted by the dotted lines) convey that the solutions must be a closed loops and periodic - but only the solution using the Symplectic Euler scheme satisifies this criterion. The other two methods are qualitatively incorrect - the solution either spirals inwards or outwards.

Hence, it is important to study methods that preserve qualitative attributes of a system as well. Since these attributes often occur in differential geometry (the study of differential equations in geometry), we call them geometric invariants - thus lending the name 'Geometric Numerical Integration' to their study.

In this paper, we will focus on a specific class of geometric invariants - nonlinear, highly oscillatory Hamiltonian problems. We will devote our attention to the problem of molecular dynamics - specifically a simulation of argon molecules as done by A. Rahman [10], and L. Verlet [14]. We will first set up the molecular dynamics simulation, focusing on various techniques to improve upon its speed. We will then investigate the preservation of geometric invariants of this system in in the long run by the Velocity Verlet algorithm (the current industry standard for molecular dynamics simulation). We will conclude by extending this investigation to the family of Newmark-Beta methods, of which the Velocity Verlet algorithm is a special case.

2 Newmark Beta Methods

In this section, we look at a family of numerical methods used to solve second order ordinary differential equations. We look at the general formula, the algorithm to implement them, and an experimental verification of convergence order for different values of the parameters.

2.1 Introduction to Newmark Beta Methods

The Newmark Beta methods were presented by Nathan M. Newmark in a paper [8] published in 1959. They were intended to be used for find solutions to problems in structural dynamics, *"capable of application to structures of any degree of complication, with any relationship between force and displacement"*, with considerations for *"any type of dynamic loading such as that due to shock or impact, vibration, earthquake motion, or blast from a nuclear weapon"*. However, they can also be used in less apocalyptical scenarios - a special case known as the 'Velocity Verlet' algorithm is the standard for solving the equations of motion in molecular dynamics simulations. Writing $\mathbf{x}_i$, $\mathbf{v}_i$, and $\mathbf{a}_i$ for the displacement, velocity, and acceleration at timestep t_i respectively, the Newmark Beta methods are of the form

$$\begin{aligned} \mathbf{v}_{i+1} &= \mathbf{v}_i + h\left[(1-\gamma)\,\mathbf{a}_i + \gamma \mathbf{a}_{i+1}\right], \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\left[(1-2\beta)\,\mathbf{a}_i + 2\beta \mathbf{a}_{i+1}\right], \end{aligned} \tag{2.1}$$

where $\gamma \in [0,1]$ and $2\beta \in [0,1]$. Note that the Newmark Beta methods are implicit if $\beta > 0$, as $\mathbf{a}_{i+1}$ depends upon the values of $\mathbf{x}_{i+1}$ and $\mathbf{v}_{i+1}$. If $\beta = 0$, the methods can be either explicit or implicit, depending on the situation in question. We consider this in greater detail in the next subsection.

These methods are second order accurate if and only if $\gamma = \frac{1}{2}$, and they are conditionally stable if and only if $2\beta \geq \gamma \geq \frac{1}{2}$. The proof of these attributes is beyond the scope of this paper, and can be found in [7]. However, we provide numerical evidence of the order of convergence in Subsection 2.2.

Some of the common choices of parameters β and γ (and the resulting schemes) are:

- Average Acceleration Method $\left(\gamma = \frac{1}{2},\, \beta = \frac{1}{4}\right)$:

$$\begin{aligned} \mathbf{v}_{i+1} &= \mathbf{v}_i + h\left[\frac{1}{2}\mathbf{a}_i + \frac{1}{2}\mathbf{a}_{i+1}\right], \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\left[\frac{1}{2}\mathbf{a}_i + \frac{1}{2}\mathbf{a}_{i+1}\right]. \end{aligned}$$

- Linear Acceleration Method $\left(\gamma = \frac{1}{2},\, \beta = \frac{1}{6}\right)$:

$$\begin{aligned} \mathbf{v}_{i+1} &= \mathbf{v}_i + h\left[\frac{1}{2}\mathbf{a}_i + \frac{1}{2}\mathbf{a}_{i+1}\right], \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\left[\frac{2}{3}\mathbf{a}_i + \frac{1}{3}\mathbf{a}_{i+1}\right]. \end{aligned}$$

- Velocity Verlet Method $\left(\gamma = \frac{1}{2},\, \beta = 0\right)$:

$$\begin{aligned} \mathbf{v}_{i+1} &= \mathbf{v}_i + h\left[\frac{1}{2}\mathbf{a}_i + \frac{1}{2}\mathbf{a}_{i+1}\right], \\ \mathbf{x}_{i+1} &= \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\mathbf{a}_i. \end{aligned} \tag{2.2}$$

2.2 Computational Application of Newmark Beta Methods

Figure 2.1 shows an excerpt from the original manuscript [8] where Newmark outlines an algorithm to calculate the next values. The method suggested belongs to a class of methods known as 'predictor-corrector' methods - an initial 'prediction' of the quantity at t_{i+1} is calculated using some function, and then this value is 'corrected' by using the prediction as the initial value for some other function to calculate the value of the quantity at the same point in time, t_{i+1}. The example below depicts the Heun method for an ODE of the form $y' = f(t,\, y)\,, \quad y(t_0) = y_0$.

$\tilde{y}_{i+1} = y_i + hf(t_i,\, y_i)\,,$ prediciting using Forward Euler Method

$y_{i+1} = y_i + \frac{h}{2}\left(f(t_i, y_i) + f(t_{i+1}, \tilde{y}_{i+1})\right).$ correcting using Trapezium Rule

Application of the General Procedure

In general unless β is 0 we may proceed with our calculation as follows:

(1) Assume values of the acceleration of each mass at the end of the interval.

(2) Compute the velocity and the displacement of each mass at the end of the interval from Eqs. (4) and (3), respectively. (Unless damping is present it is not necessary to compute the velocity at the end of the interval until step (5) is completed.)

(3) For the computed displacements at the end of the interval compute the resisting forces R which are required to hold the structural framework in the deflected configuration.

(4) From Eq. (1) and the applied loads and resisting forces at the end of the interval recompute the acceleration at the end of the interval.

(5) Compare the derived acceleration with the assumed acceleration at the end of the interval. If these are the same the calculation is completed. If these are different, repeat the calculation with a different value of assumed acceleration. It will usually be best to use the derived value as the new acceleration for the end of the interval.

Figure 2.1: Excerpt on applying the Newmark Beta methods computationally

However, this raises a question: how many times should the cycle of prediction and correction be run to get an accurate result? We can avoid this question by simplifying our assumptions. Since the goal of this paper is to apply the Newmark Beta methods to a molecular dynamics simulation, we can tune the method accordingly. In the absence of external forces, the molecular dynamics simulation is a conservative system. So the force

(and by Newton's Second Law, the acceleration) is dependent only on the displacement of the particles. Thus, we can re-write Equation 2.1 as

$$\mathbf{v}_{i+1} = \mathbf{v}_i + h\left[(1-\gamma)\,\mathbf{a}_i + \gamma \mathbf{a}_{i+1}\left(\mathbf{x}_{i+1}\right)\right], \tag{2.3}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\left[(1-2\beta)\,\mathbf{a}_i + 2\beta \mathbf{a}_{i+1}\left(\mathbf{x}_{i+1}\right)\right]. \tag{2.4}$$

as we will already have computed the values of $\mathbf{x}_i, \mathbf{v}_i,$ and $\mathbf{a}_i$.

Thus, we can see that the equation is implicit only for $\mathbf{x}_{i+1}$. However, it is explicit for $\mathbf{v}_{i+1}$, as we can calculate $\mathbf{a}_{i+1}$ once we have the value of $\mathbf{x}_{i+1}$. We can use Newton-Raphson Iteration to solve for $\mathbf{x}_{i+1}$ in Equation 2.4

$$\begin{aligned}
\tilde{\mathbf{F}}\left(\mathbf{x}_{i+1}\right) &= \mathbf{x}_{i+1} - \mathbf{x}_i - h\left(\mathbf{v}_i + \frac{h}{2}\left((1-2\beta)\,\mathbf{a}_i + 2\beta \mathbf{a}_{i+1}\left(\mathbf{x}_{i+1}\right)\right)\right), \\
D_{\mathbf{x}_{i+1}}\tilde{\mathbf{F}}\left(\mathbf{x}_{i+1}\right) &= \mathbf{I} - \beta h^2 \mathbf{J}_{\mathbf{a}_{i+1}}, \\
\mathbf{x}_{i+1} &= \mathbf{x}_i - D\tilde{\mathbf{F}}\left(\mathbf{x}_i\right)^{-1}\tilde{\mathbf{F}}\left(\mathbf{x}_i\right).
\end{aligned} \tag{2.5}$$

where $\mathbf{J}_{\mathbf{a}_{i+1}}$ is the Jacobian of the acceleration function at time t_{i+1}. Note that this equation requires that the inverse of $D\tilde{\mathbf{F}}\left(\mathbf{x}_i\right)$ exists. If this is satisified, then we use the following algorithm to find the next set of points:

Algorithm 2.1: Newmark-Beta Method Algorithm

input : $\mathbf{x}_i, \mathbf{v}_i, \mathbf{a}_i, h, \gamma, \beta$
output: $\mathbf{x}_{i+1}, \mathbf{v}_{i+1}, \mathbf{a}_{i+1}$

$\mathbf{v}_{i+1} \leftarrow \mathbf{v}_i + h\left(1-\gamma\right)\mathbf{a}_i$; // Updating the explicit part of the equation
Calculate $\mathbf{x}_{i+1}$ using the Newton-Raphson Method
$\mathbf{a}_{i+1} \leftarrow \frac{1}{m}\mathbf{F}\left(\mathbf{x}_{i+1}\right)$
$\mathbf{v}_{i+1} \leftarrow \mathbf{v}_{i+1} + \gamma h \mathbf{a}_{i+1}$; // Updating the 'implicit' part of the equation

Now that we have the algorithm laid out, we can look at applying it to an example and test the convergence of Newmark Beta methods.

2.3 Convergence Tests

We consider the case of a pendulum, governed by the equation:

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) \tag{2.6}$$

We test for convergence in the following way: we fix the values of β and γ, set $T = 5$, and consider $N = 16, 32, 64, 128, 256, 512,$ and 1024 $\left(\text{hereby varying } h := \frac{T}{N}\right)$. Then, for each value of h, we calculate the position and velocity of the pendulum at $T = 5$, and compare these values to the reference solution calculated using MATLAB's `ode45` function (with absolute and relative tolerances of 10^{-12}). We define the overall error in the estimation at time t_i as a function of the error in displacement and velocity at t_i:

$$err_i = \sqrt{\left(err_{x_i}\right)^2 + \left(err_{v_i}\right)^2}$$

Looking at the Taylor series expansion of the analytical solution, we get that $err_i \approx \mathcal{O}\left(h^a\right)$ for some $a \in \mathbb{Z}^+$. Hence, we get that a - the algebraic order of convergence - is the slope

of the straght line when $\log(err_i)$ is plotted against $\log(h)$. We can see that Figure 2.2 corroborates the statement in Subsection 2.1 - the order of a Newmark Beta method is 1 unless $\gamma = 0.50$, when it is 2.

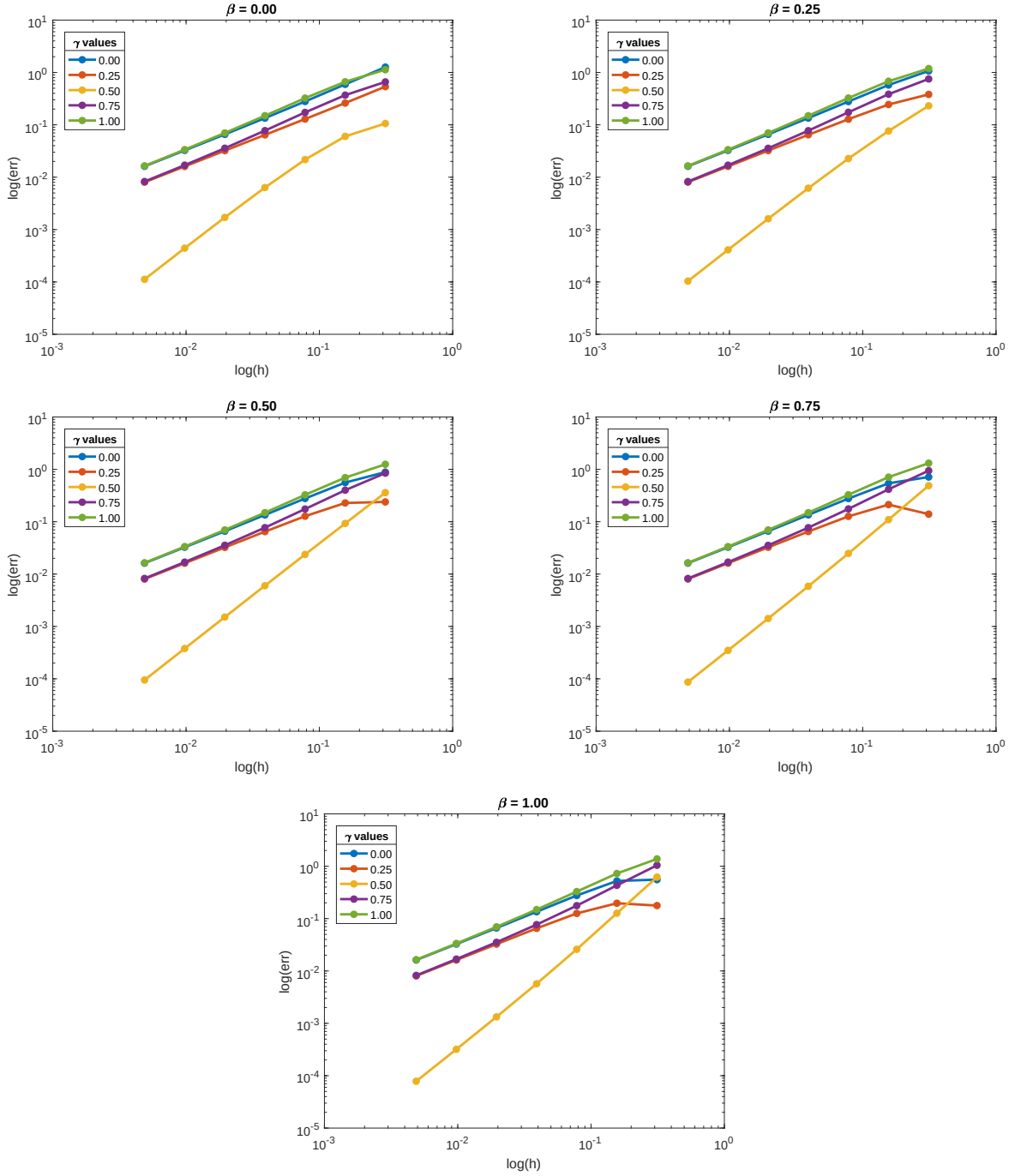

Figure 2.2: Order of convergence of Newmark Beta methods for varying parameter values

3 Molecular Dynamics

One of the most seminal works in Molecular Dynamics was Dr. Aneesur Rahman's 1964 paper [10] "Correlations in the Motion of Atoms in Lquid Argon", where a system of 864 argon atoms was simulated on a CDC 3600 computer using a predictor-corrector method and the computed quantities were matched against the empirical values. This was followed by a paper in 1967 [14] by Dr. Loup Verlet, which introduced the Verlet Integration method for the same model of 864 argon atoms.

In this section, we look at setting up the molecular dynamics simulation as described in the aforementioned papers. We focus consider the basic foundations of the simulation such as inter-particle potentials and interaction, and then introduce certain techniques that help us implement these foundations in a manner that would not be computationally expensive.

3.1 Hamiltonian Equation

For a system of molecules, we have to solve the Hamiltonian [2]

$$H(p,q) = \frac{1}{2}\sum_{i=1}^{N} m_i^{-1} p_i^T p_i + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} V_{ij}\left(\|q_i - q_j\|\right), \tag{3.1}$$

where V_{ij} is a potential function, and q_i, p_i and m_i represent the position, momentum and mass of molecule i respectively. The first term of the equation can be seen as the kinetic energy of the system, while the second term is the potential energy. We re-write Equation 3.1 as

$$\dot{q_i} = \frac{1}{m_i} p_i, \qquad \dot{p_i} = \sum_{j=1}^{N} \nu_{ij}\left(q_i - q_j\right), \tag{3.2}$$

where for $i < j$, $\nu_{ij} = \nu_{ji} = -\frac{V'_{ij}(r_{ij})}{r_{ij}}$ and $\nu_{ii} = 0$, with $r_{ij} = \|q_i - q_j\|_2$.

3.2 Lennard-Jones Potential

Before populating the argon atoms, we need to establish how the molecules will interact. We use the Lennard-Jones Potential to approximate the interaction with two atoms

$$V\left(\mathbf{r}_{ij}\right) = 4\varepsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right], \tag{3.3}$$

where $\mathbf{r}_{ij}$ is the displacement between the i^{th} and the j^{th} atoms, $r_{ij} = \|\mathbf{r}_{ij}\|_2$, ε is the potential well depth (a measure of strength of attraction of two atoms), and σ is the distance at which the potential between two atoms is 0. The Lennard-Jones potential is a very simple model for the interactive forces between atoms; nevertheless it

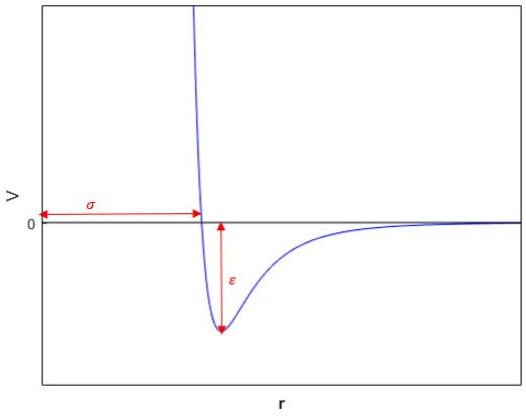

Figure 3.1: Lennard-Jones Potential

was found to suffice for modelling argon atoms in [10]. In fact, the simplicity of the Lennard-Jones potential contributes to speed of the simulation - given a system of n atoms, the potential needs to be evaluated $\frac{n(n+1)}{2}$ times (as $r_{ij} = r_{ji}$).

For an argon atom, $\varepsilon = 1.65 \times 10^{-21}$ J and $\sigma = 3.40 \times 10^{-10}$ m. The computer is prone to making errors when operating on such small numbers, so we look at a technique that handles this problem in Subsection 3.4.

3.3 Forces

We have that force $\mathbf{f}$ along the vector $\mathbf{r}_{ij}$ is given by

$$\mathbf{f}\left(\mathbf{r}_{ij}\right) = -\frac{dV}{d\mathbf{r}_{ij}}.$$

Now,

$$\frac{dV}{dr_{ij}} = 4\varepsilon\left[-12\sigma^{12}\left(\frac{1}{r_{ij}}\right)^{13} + 6\sigma^{6}\left(\frac{1}{r_{ij}}\right)^{7}\right],$$

and

$$\frac{dr_{ij}}{d\mathbf{r}_{ij}} = \frac{\mathbf{r}_{ij}}{r_{ij}}.$$

By the chain rule, we get that

$$\begin{aligned}\mathbf{f}\left(\mathbf{r}_{ij}\right) &= -4\varepsilon\left[-12\sigma^{12}\left(\frac{1}{r_{ij}}\right)^{13} + 6\sigma^{6}\left(\frac{1}{r_{ij}}\right)^{7}\right] \times \frac{\mathbf{r}_{ij}}{r_{ij}} \\ &= 48\varepsilon\left[\sigma^{12}\left(\frac{1}{r_{ij}}\right)^{14} - \frac{\sigma^{6}}{2}\left(\frac{1}{r_{ij}}\right)^{8}\right]\mathbf{r}_{ij}. \end{aligned} \tag{3.4}$$

Hence, we get that the force exterted on particle i, $\mathbf{F}_i$ is given by

$$\mathbf{F}_i = \sum_{i \neq j} \mathbf{f}_{ij}.$$

Note that we can reduce the number of computations of $\mathbf{f}_{ij}$ by half due to Newton's Third Law Motion: every action has an equal and opposite reaction implies that $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$.

3.4 Reduced Units

As explained earlier, limits on a computer's precision can introduce floating point errors when expressing miniscule quantities such as potential in terms of the SI units. Hence, we scale - or "reduce" - these SI unis such that the quantities involved in the simulation have an order of magnitude of 1. This makes simplifies the calculations, and has the added benefit of making erroneous results easier to spot (as extreme values are now unlikely). We start off by expressing lengths in terms of σ^*, where $1\,\sigma^* = 3.40 \times 10^{-10}$ m, and energy in terms of ε^*, where $1\,\varepsilon^* = 1.65 \times 10^{-21}$ J. Then, the Lennard-Jones potential can be calculated as

$$V\left(\mathbf{r}_{ij}\right) = 4\left[\left(\frac{1}{r_{ij}}\right)^{12} - \left(\frac{1}{r_{ij}}\right)^{6}\right], \tag{3.5}$$

where $\mathbf{r}_{ij}$ and r_{ij} have the same meaning as before, but are expressed as a multiple of σ^*. Then, we can rewrite Equation 3.4 as

$$\mathbf{f}\left(\mathbf{r}_{ij}\right) = 48\left(\frac{1}{r_{ij}}\right)^8\left[\left(\frac{1}{r_{ij}}\right)^6 - \frac{1}{2}\right]\mathbf{r}_{ij}. \tag{3.6}$$

As the SI unit for force is $\mathrm{J{\cdot}m^{-1}}$, the reduced unit for force is $\varepsilon \cdot \sigma^{-1}$.

Similarly, we define a reduced unit for mass, m^* for the mass of an argon atom (6.69×10^{-23} g). In fact, we can simplify the calculation $\mathbf{a} = \frac{\mathbf{F}}{m}$ by setting $1\ \mathrm{m}^* = 48\varepsilon^*{\sigma^*}^{-2}{\mathrm{s}^*}^2$, where s^* is the yet-to-be-calculated reduced unit for time (this definition follows from $1\ \mathrm{J} = 1\ \mathrm{kg{\cdot}m^{-2}{\cdot}s^2}$). Plugging in the values, we get that $1\ \mathrm{s}^* = 3.125 \times 10^{-13}$ s. Then a step-size of 10 femtoseconds is equivalent to $h = 0.032$. We use such a small step size as we assume during numerical differentiation that the velocity and the acceleration of an argon atom are constant in the interval of one time step. Lastly, we will see in a later subsection that the distribution of velocities in a system of particles depends upon the Boltzmann constant, $k_B = 1.381 \times 10^{-23}\mathrm{J \cdot K^{-1}}$. We set this to 1 in our reduced units system, by defining $1\ \mathrm{K}^* = 119.6$ K. These results are summarised in Table 3.1.

Physical Quantity	Reduced Unit	Value
Length	σ^*	3.40×10^{-10} m
Energy	ε^*	1.65×10^{-21} J
Mass	m^*	6.69×10^{-23} kg
Time	s^*	3.125×10^{-13} s
Temperature	K^*	119.6 K

Table 3.1: Reduced Units and their Values

3.5 Initializing Positions

We initialize the atoms as face-centered cubic lattice, i.e., a cube consisting of periodic images of a unit lattice cell where the atoms are placed at the eight corners, and at the center of each face. It is essential to start off the simulation with the atoms in this structure irrespective of what state of matter we want to study the simulation in. This is because randomly distributing the particles could lead to some of them being placed in close proximity of each other, leading to strong repulsive forces that would introduce errors in the simulation. Hence, if we want to study argon atoms in liquid or gas state, we start off with a face-centered cubic lattice, and increase the temperature of the system while the simulation is running.

We take as input the density of the cube to set the length of the cubic lattice. Thus, for 864 atoms at a density of $1.374\ \mathrm{g \cdot cm^{-3}}$, we get the following cube lattice

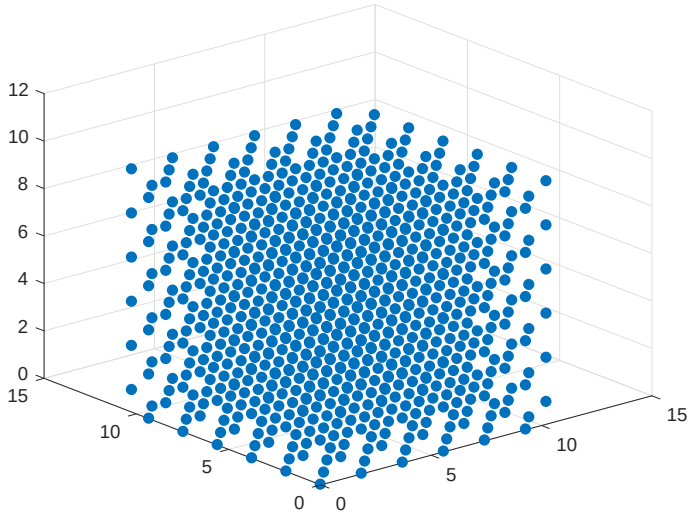

Figure 3.2: Initial cubic lattice

3.6 Initializing Velocities

As argon is an ideal gas, we can set the initial temperature to above boiling point (87.8996 K or 0.7356 K*) and use the Maxwell-Boltzmann distibution to generate the initial set of velocities. The probablity density function for velocities under this distribution is given by

$$f_{\mathbf{v}}\left(v_x, v_y, v_z\right) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{3}{2}} \exp\left[-\frac{m\left(v_x^2 + v_y^2 + v_z^2\right)}{2k_B T}\right].$$

Assuming that the velocity in a direction is independent of the velocities in the other directions, we get that the probability density function for velocity v_i in direction i is

$$f_{\mathbf{v}}\left(v_i\right) = \left(\frac{m}{2\pi k_B T}\right)^{\frac{1}{2}} \exp\left[-\frac{m v_i^2}{2k_B T}\right].$$

So, $v_i \sim \mathcal{N}\left(0, \frac{k_B T}{m}\right)$. We use the `normrnd` function of MATLAB to generate the velocities, and then correct them so that there is no overall linear momentum.

Now, we are using the Maxwell-Boltzmann distribution for ideal gases to initialize the velocities, but the particles have been initialized as a solid. Hence, we will 'boil' the solid for a fixed number of steps at the start of the simulation by scaling the velocities

$$\mathbf{v}_{new} = \sqrt{\frac{T_{target}}{T_{old}}} \cdot \mathbf{v}_{old}.$$

This method is known as equilibration.

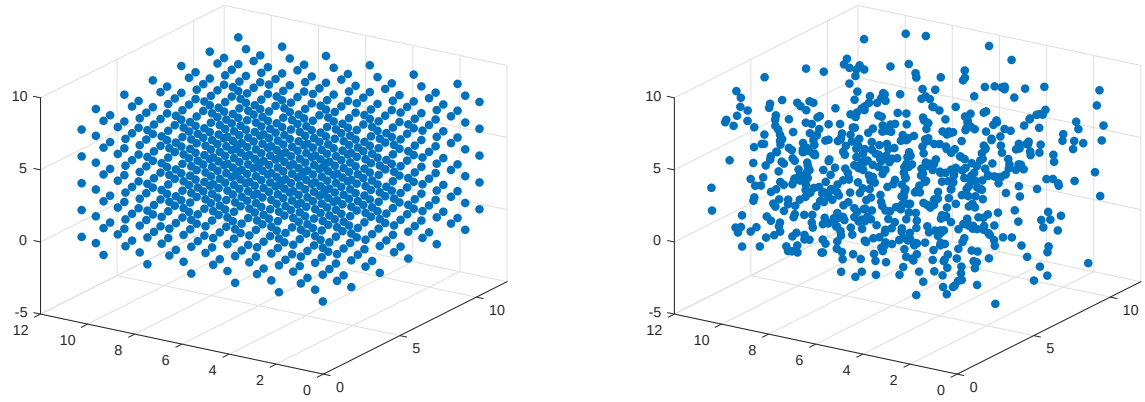

Figure 3.3: Change in the position of particles after 500 steps of equilibration

3.7 Periodic Boundary Conditions and Minimum Image Convention

Typically, we'd imagine the argon gas to be in a container, with particles colliding against the walls and moving off in the opposite direction. In the real world, the number of particles colliding against the wall would be miniscule - for a liquid with 10^{21} particles, only 1 in 10^7 would be close to the walls [11]. In comparison, our system of 846 particles has almost half of them at the edges. The simulation would, thus, not reflect the internal state of the system accurately. We get rid off the walls through a technique known as periodic boundary conditions. Given a face-centered cubic lattice with side L, we consider it to consist of 27 identical cubes of side $\frac{L}{2}$ (one central box and 26 images of it), and we maintain this idea of images throughout the simulation, as shown in Figure 3.4. This has two effects:

- Say we have two boxes beside each, Box A and Box B (where Box B is an image of Box A). Then, as a particle moves out of Box A (say from the left), its image in Box B also moves out of Box B from the left, and enters Box A. This maintains that all the boxes in the cube are images of each other.
- A particle in a box will interact not only with the particles in the same box, but also the particles in the other boxes - leading to repeated calculation and effect of the same force.

We consider for this 'wrap-around' effect through the Minimum Image Convention - each particle interacts only with the particles in its box. Given x-components x_i for particle i, x_j for particle j, and the length of the box in the x-component, L_x, we implement

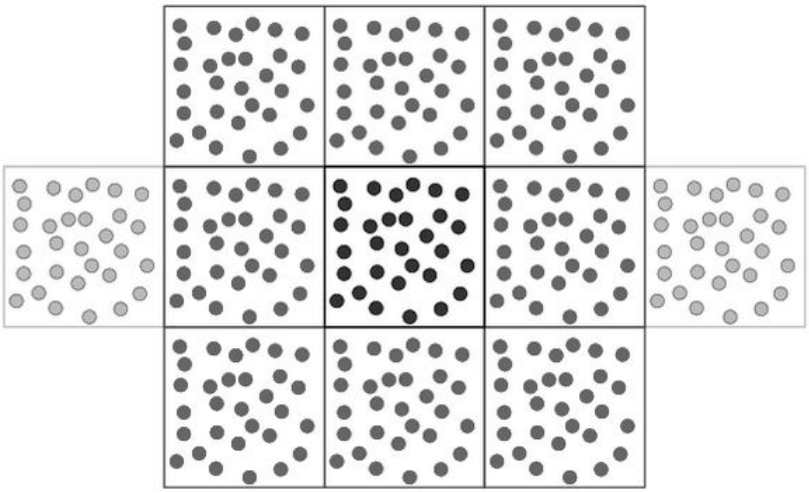

Figure 3.4: Periodic Boundary Conditons in 2-d

the Minimum Image Criterion using the algorithm below.

Algorithm 3.1: Minimum Image Convention in 1-d

input : x_i, x_j, L_x
output: x_{ij}

$x_{ij} \leftarrow x_i - x_j$;
$half\_L_x \leftarrow \frac{L_x}{2}$;
if $x_{ij} > half\_L_x$ **then**
  $x_{ij} \leftarrow x_{ij} - L_x$;
else
  if $x_{ij} \leq -half\_L_x$ **then**
    $x_{ij} \leftarrow x_{ij} + L_x$;
  end
end

This is repeated for all three dimensions to find the correct image of the particle. We shorten the above algorithm by making use of MATLAB's vector operations: `diff_r = diff_r - L*round(diff_r/L)`. The elimination of the conditional checks gives us a 10x reduction in time spent on implementing the minimum image convention.

3.8 Cut-off Potential and Neighbour Lists

A final consideration for our molecular dynamics simulation is the number of times we are evaluating the forces, and how many of these evaluations are effective. Looking at the graph of the Lennard-Jones Potential in Figure 3.1, we see that the potential tends to zero as the inter-particle distance increases. Furthermore, the potential does not change much for over larger distances - implying that it is not efficient to consider interactions over all inter-particle distances. Rather, we can define a cut-off distance, r_c beyond which the potential is zero. We neeed to now shift the Lennard-Jones Potential; otherwise the potential will have a discontinuity at r_c and Newton's Laws of Motion do not hold at

discontinuities. So, we get the shifted Lennard-Jones potential

$$V(\mathbf{r}_{ij}) = \begin{cases} 4\left[\left(\left(\frac{1}{r_{ij}}\right)^{12} - \left(\frac{1}{r_{ij}}\right)^{6}\right) - \left(\left(\frac{1}{r_c}\right)^{12} - \left(\frac{1}{r_c}\right)^{6}\right)\right] & , \quad \text{if } r_{ij} \leq r_c \\ 0 & , \quad \text{otherwise} \end{cases} \tag{3.7}$$

What is the optimal value for r_c? Periodic boundary conditions enforce that $r_c < \frac{L}{2}$ (as particles can only interact with particles in the same box). Generally, $r_c \leq \frac{L}{4}$[3]. In our simulations, we have $L = 10.229\,\sigma^*$; so, we define $r_c = 2.5\,\sigma^*$. We substitute this into Equation 3.5 and confirm that this is a good choice: $V(2.5) = -0.0163$, $V(2.6) = -0.0129$, and $V(3) = -0.005479$. The loss in potential by setting $r_c = 2.5\,\sigma^*$ is minimal. The effect on computing time is phenomenal - only 6.3% of all pairwise distances are smaller than $2.5\,\sigma^*$, thus eliminating 93.7% of the computationally intensive force calculations.

As we saw earlier, repeated conditional checks are not very efficient and consume a lot of time. This is especially expensive if each iteration of the simulation involved calculating the inter-particle distance and checking if that distance is lesser than $2.5\sigma^*$ - around $400,000$ checks per iteration! Assuming that the positions of the particles do not vary much in each successive iteration, we can calculate a list of 'neighbours' for each particle at the start of the simulation. However, it cannot be guaranteed that this list will be accurate for the entire run of the simulation. In fact, tests conducted with the Velocity Verlet algorithm show that this list is only 80% accurate after 10 iterations, and 50% accurate after 35 iterations. Clearly, we can't use the same list for the entire simulation.

To find the optimal number of iterations after which the list of neighbours should be refreshed (N_n), we run the simulation for 100 equilibration iterations (N_e) followed by 600 iterations (N_f), calculating the energy values every 10 iterations (N_s) after equilibration. We compute the time taken for each run and the percentage error in the final energy values, using $N_n = 1$ as our reference (neighbours lists are updated on each iteration). The results summarised in Table 3.2 make the case for using the same list for over multiple iterations (5× improvement in speed), but updating them regularly enough to reduce the overall error. There's trade-off between speed and accuracy, and setting $N_n = 15$ seems to be a good compromise.

N_n	% Error in Final Energy Value	Time taken (s)
1	0.00	945.86
10	3.18	164.55
15	4.43	146.90
20	5.37	127.55
35	7.67	99.58
100	8.08	85.08
No update	-1.21	62.18

Table 3.2: Results of tests to find optimal N_n

3.9 Computing the Jacobian

Recall from Equation 2.5 that we need to compute the Jacobian of $\mathbf{a}(\mathbf{x}_i)$. For this, we need to store the coordinates of the molecules in a column vector of size $3N$, where N is the number of molecules in the system - rather than a more intuitive $N \times 3$ matrix. We fix the order of coordinates as

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ \vdots \\ x_{3N} \\ y_{3N} \\ z_{3N} \end{pmatrix}$$

where x_i, y_i, z_i are the x,y, and z coordinates of particle i respectively (we will continue to use an $N \times 3$ matrix for the explicit Newmark Beta methods, as this orientation is proven to be faster during tests). The Jacobian is then a $3N \times 3N$ matrix, which would be very expensive to compute for large N.

$$J = \begin{pmatrix} \frac{\partial \mathbf{f}_1}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{f}_1}{\partial \mathbf{p}_2} & \cdots & \frac{\partial \mathbf{f}_1}{\partial \mathbf{p}_n} \\ \frac{\partial \mathbf{f}_2}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{f}_2}{\partial \mathbf{p}_2} & \cdots & \frac{\partial \mathbf{f}_2}{\partial \mathbf{p}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{f}_n}{\partial \mathbf{p}_1} & \frac{\partial \mathbf{f}_n}{\partial \mathbf{p}_2} & \cdots & \frac{\partial \mathbf{f}_n}{\partial \mathbf{p}_n} \end{pmatrix},$$

where $\mathbf{f}_i = \sum_{i=1}^{N} \mathbf{f}_{ij}$, $\mathbf{p}_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}$, and

$$\frac{\partial \mathbf{f}_i}{\partial \mathbf{p}_j} = \begin{pmatrix} \frac{\partial f_{i_x}}{\partial x_j} & \frac{\partial f_{i_x}}{\partial y_j} & \frac{\partial f_{i_x}}{\partial z_j} \\ \frac{\partial f_{i_y}}{\partial x_j} & \frac{\partial f_{i_y}}{\partial y_j} & \frac{\partial f_{i_y}}{\partial z_j} \\ \frac{\partial f_{i_z}}{\partial x_j} & \frac{\partial f_{i_z}}{\partial y_j} & \frac{\partial f_{i_z}}{\partial z_j} \end{pmatrix}, \tag{3.8}$$

where f_{i_x} is the x-component of the force $\mathbf{f}_i$.
Now, we have from Equation 3.6

$$\begin{aligned} f_{i_x} &= \sum_{j=1}^{N} f_{ij_x} \\ &= \sum_{j \in ngbrs(i)} 48 \left[\left(\frac{1}{r_{ij}} \right)^{14} - \frac{1}{2} \left(\frac{1}{r_{ij}} \right)^{8} \right] x_{ij}, \end{aligned}$$

where $r_{ij} = \sqrt{x_{ij}^2 + y_{ij}^2 + z_{ij}^2}$, and $ngbrs(i)$ is the list of neighbors of particle i.
Focusing on just f_{ij_x} for some i and j, we write $f_{ij_x} = 48 \cdot s_{ij} \cdot x_{ij}$. We can see that

$$\begin{aligned} \frac{ds_{ij}}{dx_i} &= -14 (x_i - x_j) \left(\frac{1}{r_{ij}} \right)^{16} + 4 (x_i - x_j) \left(\frac{1}{r_{ij}} \right)^{10}, \\ \frac{ds_{ij}}{dx_j} &= 14 (x_i - x_j) \left(\frac{1}{r_{ij}} \right)^{16} - 4 (x_i - x_j) \left(\frac{1}{r_{ij}} \right)^{10} \\ &= -\frac{ds_{ij}}{dx_i}. \end{aligned}$$

Then,

$$\begin{aligned}\frac{df_{ij_x}}{dx_i} &= 48\left(\frac{ds_{ij}}{dx_i}\cdot x_{ij} + s_{ij}\cdot\frac{dx_{ij}}{dx_i}\right)\\ &= 48\left(-14\left(x_i-x_j\right)^2\left(\frac{1}{r_{ij}}\right)^{16} + 4\left(x_i-x_j\right)^2\left(\frac{1}{r_{ij}}\right)^{10} + s_{ij}\right), \qquad (3.9)\\ \frac{df_{ij_x}}{dx_j} &= 48\left(\frac{ds_{ij}}{dx_j}\cdot x_{ij} + s_{ij}\cdot\frac{dx_{ij}}{dx_j}\right)\\ &= 48\left(-\frac{ds_{ij}}{dx_i}\cdot x_{ij} - s_{ij}\right)\\ &= -\frac{df_{ij_x}}{dx_i}. \qquad (3.10)\end{aligned}$$

Furthermore,

$$\begin{aligned}\frac{df_{ij_x}}{dy_i} &= 48\left(\frac{ds_{ij}}{dy_i}\cdot x_{ij} + s_{ij}\cdot\frac{dx_{ij}}{dy_i}\right)\\ &= 48\left(-14\left(x_i-x_j\right)\left(y_i-y_j\right)\left(\frac{1}{r_{ij}}\right)^{16} + 4\left(x_i-x_j\right)\left(y_i-y_j\right)\left(\frac{1}{r_{ij}}\right)^{10}\right), \qquad (3.11)\\ \frac{df_{ij_x}}{dy_j} &= 48\left(\frac{ds_{ij}}{dy_j}\cdot x_{ij} + s_{ij}\cdot\frac{dx_{ij}}{dy_j}\right)\\ &= 48\left(-\frac{ds_{ij}}{dy_i}\cdot x_{ij}\right)\\ &= -\frac{df_{ij_x}}{dy_i}. \qquad (3.12)\end{aligned}$$

We extend these equations to all three components, and get from Equations 3.9 and 3.11 that

$$\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{p}_i} = \begin{pmatrix} s_{ij} & 0 & 0\\ 0 & s_{ij} & 0\\ 0 & 0 & s_{ij}\end{pmatrix} + 48\left(-14\left(\frac{1}{r_{ij}}\right)^{16} + 4\left(\frac{1}{r_{ij}}\right)^{10}\right)\mathbf{r}_{ij}\mathbf{r}_{ij}^T,$$

where $\mathbf{r}_{ij} = \begin{pmatrix} x_i-x_j\\ y_i-y_j\\ z_i-z_j\end{pmatrix}$. Lastly, we need the following relations

$$\begin{aligned}&\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{p}_j} = -\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{p}_i} && \cdots \text{ from Equations 3.10 and 3.12,}\\ &\frac{\partial \mathbf{f}_{ji}}{\partial \mathbf{p}_i} = -\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{p}_i} && \cdots \text{ as } \mathbf{f}_{ji} = -\mathbf{f}_{ji},\\ &\frac{\partial \mathbf{f}_{ji}}{\partial \mathbf{p}_j} = -\frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{p}_j} = \frac{\partial \mathbf{f}_{ij}}{\partial \mathbf{p}_i} && \cdots \text{ combining the two relations above.}\end{aligned}$$

Now, we can use the last four relations to populate Equation 3.8, and we can calculate the Jacobian.

4 Experiment

We now look at running the simulation, and analyzing the results. First, we look at some theoretical results pertaining to the four geometrical invariants under investigation - time reversibilty, total energy, total linear momentum, and total angular momentum. Then, we look at running the simulation for different values of the parameters β and γ. We will assume that the simulations have the following initial configuration (unless stated otherwise):

Parameter	Initial Value
No. of particles	864
Temperature	1 K^*
Density	38.744 $m^*{\sigma^*}^{-3}$
h	0.032
N_e	100
N_f	500
N_s	10
N_n	15

Table 4.1: Initial Configuration for molecular simulations

The last four elements in the table indicate that the simulation will initially be equilibrated for 100 iterations, and then run for 500 iterations where samples will be collected every 10 iterations (giving a total of 50 samples). The list of neighbours will be updated every 10 iterations in both the equilibration and final stages. These simulations wil be run on an Intel i3 processor with clockspeed 1.7 GHz and 8 GB of RAM.

4.1 Time Reversibility

The molecular dynamics system is time reversible - if we go from state s_1 to state s_2 in time δt, then we can return to state s_1 from state s_2 in time δt by reversing the signs on the velocities. This can be seen formally by applying the transformation $t \mapsto -t$ to Equation 3.2 - the position q does not change, but the sign on momentum p gets reversed.

$$q \mapsto q = \tilde{q}, \qquad p \mapsto -p = \tilde{p}.$$

Then

$$\begin{aligned}
\dot{\tilde{q}} = \frac{d\tilde{q}}{d\,(-t)} &= -\nabla_{\tilde{p}} H &&= \nabla_p H &&= \dot{q}, \\
\dot{\tilde{p}} = \frac{d\tilde{p}}{d\,(-t)} &= -\nabla_{\tilde{q}} H &&= -\nabla_q H &&= \dot{p}.
\end{aligned}$$

Thus, the Hamiltonian system is time reversible.

We say that a numerical one-step method $\Phi_h : (\mathbf{x}_i, \mathbf{v}_i) \mapsto (\mathbf{x}_{i+1}, \mathbf{v}_{i+1})$ is time reversible if $\Phi_h = \Phi_{-h}^{-1}$. Informally, this means that if we exchange $i \leftrightarrow i+1$ and replace h by $-h$ in our original method, then we should get the same method back. In order to ascertain which Newmark Beta methods are reversible, we cite [9]:

Theorem 4.1. *The maximal order of a reversible one-step method is always even.*

As Newmark Beta methods have maximal order 2 only if $\gamma = \frac{1}{2}$, the contrapositive of Theorem 4.1 states that any scheme with $\gamma \neq \frac{1}{2}$ is not reversible. Now, we claim

Claim 4.2. *Any Newmark Beta method with $\gamma = \frac{1}{2}$ is reversible.*

Proof. A Newmark Beta scheme with $\gamma = \frac{1}{2}$ has the form

$$\mathbf{v}_{i+1} = \mathbf{v}_i + h\left(\frac{1}{2}\mathbf{a}_i + \frac{1}{2}\mathbf{a}_{i+1}\right), \tag{4.1}$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\left[(1-2\beta)\mathbf{a}_i + 2\beta\mathbf{a}_{i+1}\right]. \tag{4.2}$$

Swapping $i \leftrightarrow i+1$ and replacing h with $-h$,

$$\mathbf{v}_i = \mathbf{v}_{i+1} - h\left(\frac{1}{2}\mathbf{a}_{i+1} + \frac{1}{2}\mathbf{a}_i\right), \tag{4.3}$$

$$\mathbf{x}_i = \mathbf{x}_{i+1} - h\mathbf{v}_{i+1} + \frac{h^2}{2}\left[(1-2\beta)\mathbf{a}_{i+1} + 2\beta\mathbf{a}_i\right]. \tag{4.4}$$

It is straight-forward to see that Equation 4.3 is the same as Equation 4.1. For Equation 4.4, we see

$$\begin{aligned}
\mathbf{x}_{i+1} &= \mathbf{x}_i + h\mathbf{v}_{i+1} - \frac{h^2}{2}\left[(1-2\beta)\mathbf{a}_{i+1} + 2\beta\mathbf{a}_i\right] \\
&= \mathbf{x}_i + h\left(\mathbf{v}_i + \frac{h}{2}\mathbf{a}_i + \frac{h}{2}\mathbf{a}_{i+1}\right) - \frac{h^2}{2}\left[(1-2\beta)\mathbf{a}_{i+1} + 2\beta\mathbf{a}_i\right] \\
&= \mathbf{x}_i + h\mathbf{v}_i + \frac{h^2}{2}\left[(1-2\beta)\,\mathbf{a}_i + 2\beta\mathbf{a}_{i+1}\right].
\end{aligned}$$

□

While the schemes may be time reversible in theory, the rounding-off errors in practice mean that we may not get back the initial state of the system after reversing the velocities. We test this for different values of β and γ over different timescales. In these experiments, we do not differentiate between equilibration and final iterations, and reverse the sign on the velocities after half of the total number of iterations have been completed.

We can see in Figure 4.1 that errors are introduced while reversing 500 iterations of the Velocity Verlet algorithm. While the particles are almost at their initial positions, the velocities tend to be higher than those in the initial state. This is worsened when we reverse 500 steps of the Newmark Beta method with $\beta = 0$ and $\gamma = 0.75$. As noted earlier, this is not a reversible method and we can see that in Figure 4.2, where the velocity histogram is much narrower that it should be. Figure 4.3 is interesting - the method clearly does not conserve the total energy in the system if it is reversed after 500 iterations (samples have been taken every 5 iterations). However, if the reversal happens after a short period of time (25 iterations) then we get that the total energy of the system is reversed as well. Figure 4.4 shows the errors in time reversibility for an implicit method wtih $\beta = 0.25$ and $\gamma = 0.5$ (which is time reversible). The histograms seem to agree very well, but if we take the absolute error in velocities of each point, we get that the mean absolute error is 0.233 with a standard deviation of 0.174 - which is considerably large given that velocities range over $[-0.6, 0.6]$. In comparison, the Velocity Verlet method has a mean absolute error of 0.158 with a standard deviation of 0.118.

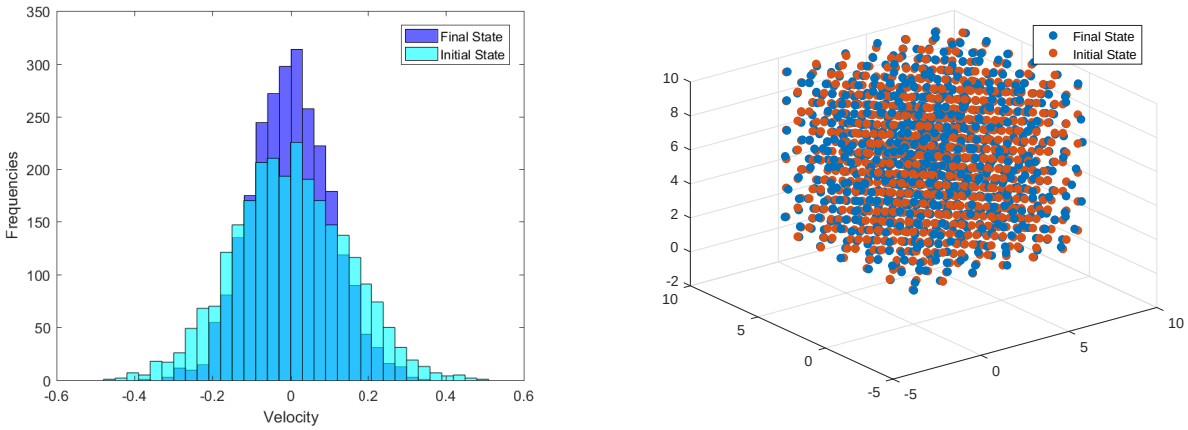

Figure 4.1: Changes in position and velocities for $\beta = 0.00$ and $\gamma = 0.50$ after reversing 500 steps

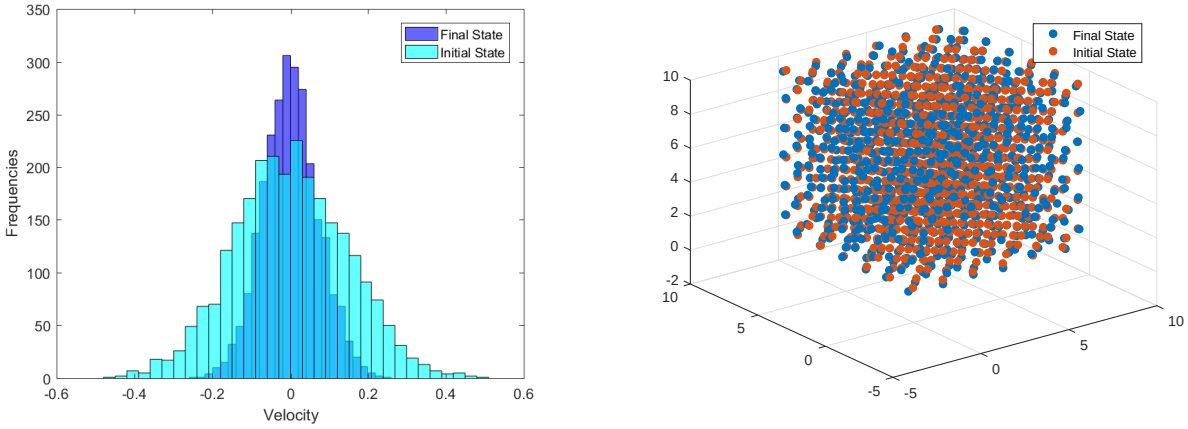

Figure 4.2: Changes in position and velocities for $\beta = 0.00$ and $\gamma = 0.75$ after reversing 500 steps

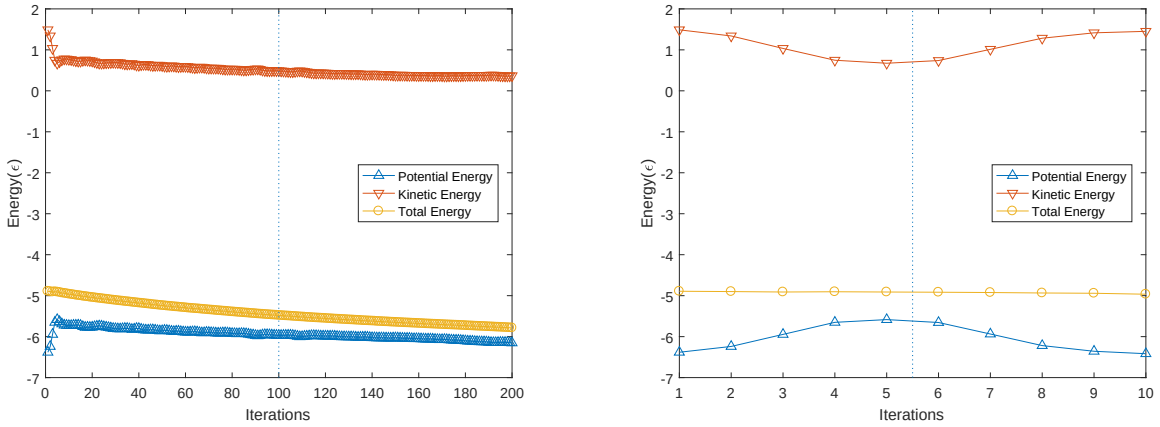

Figure 4.3: Difference in behaviour of total energy for $\beta = 0.00$ and $\gamma = 0.75$ over different timesteps

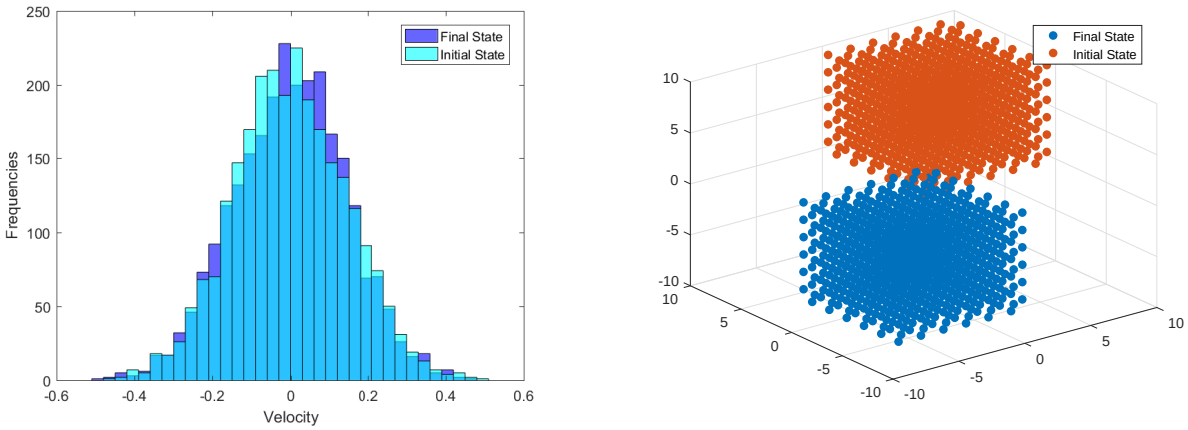

Figure 4.4: Changes in position and velocities for $\beta = 0.25$ and $\gamma = 0.50$ after reversing 125 steps

The coordinates in Figure 4.4do not seem to be conserved, but rather translated. A possible reason for this is that in the Newton-Raphson Iteration, we introduce a small noise to $\mathbf{x}_i$ before calculating $\mathbf{x}_{i+1}$ to prevent the Jacobian from being sparse. This seems to be a plausible reason, as all the coordinates have a pointwise absolute error of 1.600 (accurate to three decimal places).

Aside: First Integrals

The next three invariants - total energy, linear momentum, and angular momentum - follow readily from Newton's Laws of Motion as there is no external force on the system. In order to simplify checking which methods preserve these invariants the best, we introduce the concept of first integrals:
A non-constant function $I(y)$ is a *first integral* (or an invariant) of the differential equation $\dot{y} = F(y)$ if $I(y(t))$ is constant along every solution, or equivalently, if

$$\nabla I(y)F(y) = 0 \qquad \forall y. \quad [2] \tag{4.5}$$

Total energy, total linear momentum, and total angular momentum are all first integrals [2]. We look at proving this in the next subsections, adapting the proofs from [2].

4.2 Total Linear Momentum

Theorem 4.3. *Total linear momentum $P = \sum_{i=1}^{N} p_i$ is a first integral.*

Proof.

$$\begin{aligned}
\nabla P = \frac{dP}{dt} &= \sum_{i=1}^{N} \frac{dp_i}{dt} \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} \nu_{ij}\left(q_i - q_j\right) \\
&= 0 \qquad\qquad \cdots \text{ as } \nu_{ij} = \nu_{ji} \forall i, j
\end{aligned}$$

□

Total linear momentum - as the name suggests - is a linear first integral. Most numerical methods preserve linear first integrals - in fact, it is a property shared by all Runge-Kutta methods. Indeed, this holds true for the entire family of Newmark Beta methods as well.

Claim 4.4. *All Newmark Beta methods preserve linear first integrals*

Proof. Let $I(\mathbf{x}, \mathbf{v}) = b^T\mathbf{x} + c^T\mathbf{v}$ be a linear first integral, where b and c are some constant vectors. By the defintion of first integrals, we want that $I'(\mathbf{x}, \mathbf{v}) = 0$ for all $\mathbf{x}, \mathbf{v}$. But $I'(\mathbf{x}, \mathbf{v}) = b^T\mathbf{v} + c^T\mathbf{a}(\mathbf{x}) = 0$, implying that $b^T = 0$ and $c^T\mathbf{a}(\mathbf{x}) = 0$ for all $\mathbf{x}$. So, $I(\mathbf{x}, \mathbf{v}) = I(\mathbf{v}) = c^T\mathbf{v}$ (This makes sense in a real-world scenario as total linear momentum depends only on the velocity of the body, and not its position).
Now, we multiply c^T to the velocity portion of the Newmark Beta method and get that

$$\begin{aligned} I(\mathbf{v}_{i+1}) &= c^T\mathbf{v}_{i+1} \\ &= c^T\mathbf{v}_i + h\left[(1-\gamma)\, c^T\mathbf{a}(\mathbf{x}_i) + \gamma c^T\mathbf{a}(\mathbf{x}_{i+1})\right] \\ &= c^T\mathbf{v}_i = I(\mathbf{v}_i) \end{aligned}$$

□

As we calibrated the system to have an initial total linear momentum of zero, the total linear momentum will be zero for all of the trials. As this holds true for all members of the Newmar Beta family, we do not present experimental results for this property, but rather state that in all trials, the total linear momentum was almost zero - it cannot be exactly zero due to inaccuracies introduced by floating-point arithmetic and rounding-off, but these errors were in the range of 10^{-10}.

4.3 Total Angular Momentum

Theorem 4.5. *Total angular momentum $L = \sum_{i=1}^{N} q_i \times p_i$ is a first integral.*

Proof.

$$\begin{aligned} \nabla L &= \sum_{i=1}^{N} \frac{d}{dt}\left(q_i \times p_i\right) \\ &= \sum_{i=1}^{N} \left(\dot{q}_i \times p_i + q_i \times \dot{p}_i\right) \\ &= \sum_{i=1}^{N} \left(\frac{1}{m_i} p_i \times p_i + q_i \times \sum_{j=1}^{N} \nu_{ij}\left(q_i - q_j\right)\right) \\ &= \sum_{i=1}^{N} \left(\frac{1}{m_i} p_i \times p_i\right) + \sum_{i=1}^{N}\sum_{j=1}^{N} \left(q_i \times \nu_{ij}\left(q_i - q_j\right)\right) \\ &= 0 \qquad\qquad \cdots \text{ as } \nu_{ij} = \nu_{ji} \text{ and } p_i \times p_i = 0\, \forall i, j \end{aligned}$$

□

Total angular momentum is a quadratic first integral. By Noether's Theorem, it has the form $I(\mathbf{x}, \mathbf{v}) = \mathbf{v}^T\left(C\mathbf{x} + d\right)$, where C is a constant square matrix and d is a constant vector [2]. We have the Velocity Verlet method preserves quadratic integrals of this form,

and hence, total angular momentum. To see this, we need to express it as the composition of two half-step methods:

$$\begin{aligned} \mathbf{v}_{i+\frac{1}{2}} &= \mathbf{v}_i + \frac{h}{2}\mathbf{a}(\mathbf{x}_i) \\ \mathbf{x}_{i+\frac{1}{2}} &= \mathbf{x}_i + \frac{h}{2}\mathbf{v}_{i+\frac{1}{2}} \end{aligned} \tag{4.6}$$

and

$$\begin{aligned} \mathbf{x}_{i+1} &= \mathbf{x}_{i+\frac{1}{2}} + \frac{h}{2}\mathbf{v}_{i+\frac{1}{2}} \\ \mathbf{v}_{i+1} &= \mathbf{v}_{i+\frac{1}{2}} + \frac{h}{2}\mathbf{a}(\mathbf{x}_{i+1}) \end{aligned} \tag{4.7}$$

Theorem 4.6. *The Velocity Verlet algorithm preserves quadratic first integrals of the form* $I(\mathbf{x}, \mathbf{v}) = \mathbf{v}^T (C\mathbf{x} + d)$

Proof. We have that $I'(\mathbf{x}, \mathbf{v}) = \mathbf{a}(\mathbf{x})^T (C\mathbf{x} + d) + \mathbf{v}^T C\mathbf{v} = 0$ for all $\mathbf{x}, \mathbf{v}$. Now,

$$\begin{aligned} \mathbf{v}_{i+\frac{1}{2}}^T \left(C\mathbf{x}_{i+\frac{1}{2}} + d\right) &= \mathbf{v}_{i+\frac{1}{2}}^T \left(C\mathbf{x}_i + \frac{h}{2}C\mathbf{v}_{i+\frac{1}{2}} + d\right) \\ &= \mathbf{v}_{i+\frac{1}{2}}^T (C\mathbf{x}_i + d) + \frac{h}{2}\mathbf{v}_{i+\frac{1}{2}}^T C\mathbf{v}_{i+\frac{1}{2}} \\ &= \left(\mathbf{v}_i + \frac{h}{2}\mathbf{a}(\mathbf{x}_i)\right)^T (C\mathbf{x}_i + d) + \frac{h}{2}\mathbf{v}_{i+\frac{1}{2}}^T C\mathbf{v}_{i+\frac{1}{2}} \\ &= \mathbf{v}_i^T (C\mathbf{x}_i + d) + \frac{h}{2}\left(\mathbf{a}(\mathbf{x}_i)^T (C\mathbf{x}_i + d) + \mathbf{v}_{i+\frac{1}{2}}^T C\mathbf{v}_{i+\frac{1}{2}}\right) \\ &= \mathbf{v}_i^T (C\mathbf{x}_i + d) + \frac{h}{2} I'(\mathbf{x}_i, \mathbf{v}_{i+\frac{1}{2}}) \\ &= \mathbf{v}_i^T (C\mathbf{x}_i + d) \end{aligned}$$

Similarly, we get from Equation 4.7 that

$$\begin{aligned} I(\mathbf{x}_{i+1}, \mathbf{v}_{i+1}) &= \mathbf{v}_{i+1}^T (C\mathbf{x}_{i+1} + d) \\ &= \mathbf{v}_{i+\frac{1}{2}}^T \left(C\mathbf{x}_{i+\frac{1}{2}} + d\right) \\ &= \mathbf{v}_i^T (C\mathbf{x}_i + d) \\ &= I(\mathbf{x}_i, \mathbf{v}_i) \end{aligned}$$

□

There are no other Newmark Beta methods that conserve total angular momentum [8]. To see this, we calculate the difference between the angular momentum at time t_{i+1}, L_{i+1} and time t_i, L_i [13]. We use the notation $(\mathbf{b}_i)_k$ to denote the k^{th} entry of vector $\mathbf{b}$ at time t_i.

$$L_{i+1} - L_i = \sum_{k=1}^{N} (q_{i+1})_k \times (p_{i+1})_k - \sum_{k=1}^{N} (q_i)_k \times (p_i)_k \,.$$

Substituting in the Newmark Beta expressions for position and momentum, we get that

$$L_{i+1} - L_i = h^2 \left(\gamma - \frac{1}{2}\right) \sum_{k=1}^{N} (\mathbf{a}_i)_k \times (\mathbf{p}_i)_k + h^2 \beta \sum_{k=1}^{N} (\mathbf{a}_{i+1})_k \times (\mathbf{p}_{i+1})_k - (\mathbf{a}_i)_k \times (\mathbf{p}_i)_k . [13] \tag{4.8}$$

Thus, we get that Equation 4.8 is 0 only if $\beta = 0$ and $\gamma = \frac{1}{2}$. We can test this result in our experimental setup, varying the parameters β and γ to see which methods come close to preserving total angular momentum. Surprsingly, we get that the angular momentum is not experimentally preserved by the Velocity Verlet algorithm! Table 4.2 shows the first five samples of total angular momentum taken during the simulation, depicting that the lack of clear trend in change of angular momentum.

Iteration	x-component	y-component	z-component
10	493.28	-402.32	1407.30
20	-4.80	-10.18	914.24
30	320.20	-147.35	980.93
40	717.70	-181.21	704.27
50	594.66	152.47	507.19

Table 4.2: Total angular momentum of the system over the first 50 iterations

The reason behind this incoherent behaviour is the application of periodic boundary conditions. Angular momentum is defined relative to the origin as it is the cross product of position and momentum. When a particle moves out of the box, it re-enters the box from the opposite face due to periodic boundary conditions - this creates a discontinuity in the position of the particle, and hence, a discontinuity in the angular momentum of the particle. Note however, that the velocity of the particle does not change, and hence, linear momentum is preserved. We say that periodic boundary conditions preserve translational symmetry, but not rotational symmetry. As a result, we have that in the simulations, no method will preserve total angular momentum. However, this will not affect the simulation as no thermodynamic property depends upon the angular momenta of the particles.

4.4 Total Energy

Theorem 4.7. *The total energy, or the Hamiltonian, is a first integral.*

Proof. From definitions,

$$\nabla H = \left(\nabla_p H^T, \nabla_q H^T\right)$$

and

$$\frac{dH}{dt} = \begin{pmatrix} -\nabla_q H \\ \nabla_p H \end{pmatrix}$$

Then,

$$\nabla H(p,q) \cdot \frac{dH}{dt} = \nabla_p H^T \cdot -\nabla_q H + \nabla_q H^T \cdot \nabla_p H = 0$$

□

The case of total energy is more complicated than those of the other geometric invariants. As we will see later, it is not the Hamiltonian is not preserved by some methods, but rather it is the 'shadow' variant of the Hamiltonian.

We start by quoting Simo, Tarnow, and Wong from [13]: "*What may seem surprising is that all of the implicit members of the Newmark family, perhaps the most widely used time-stepping algorithms in nonlinear structural dynamics, are not designed to conserve energy and also fail to conserve momentum. Among the explicit members, only the central difference method preserves momentum.*"

Here, the "central difference method" refers to the Velocity Verlet Scheme, i.e., $\beta = 0$ and $\gamma = \frac{1}{2}$. But what is so significant about $\gamma = \frac{1}{2}$ that it appears everywhere? The answer lies in the interpretation of γ in the Newmark Beta equations: γ controls the damping in system by a factor of $\gamma - \frac{1}{2}$. So, if $\gamma > \frac{1}{2}$, then the energy in the system is damped, and if $\gamma < \frac{1}{2}$, the energy in the system grows. We see this numerically in Figure 4.5.

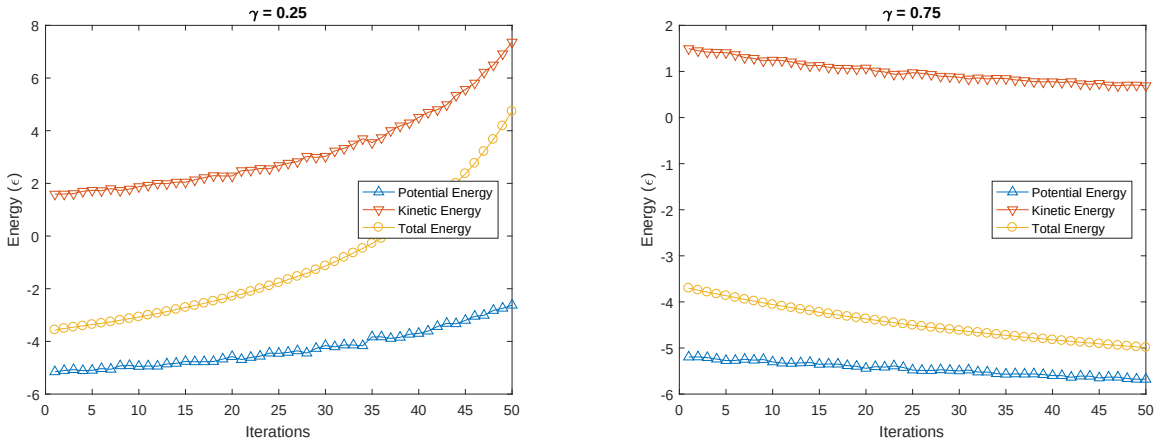

Figure 4.5: Energy time series for two explicit Newmark Beta methods

Thus, we will restrict our numerical investigation to the cases where $\gamma = \frac{1}{2}$ and $\beta \in \left[0, \frac{1}{2}\right]$. For the explicit case ($\beta = 0$), we will use the initial conditions as defined in Table 4.1. For the implicit cases, however, we set $N_e = 50$ and $N_f = 200$ (N_n and N_s remain unchanged). Understandably, this could hamper the simulation; however, an implicit Newmark Beta method takes an average of 1100 s to perform 250 overall iterations, as compared to an explicit Newmark Beta method that takes an average of 170 s to perform 600 iterations. This slow speed of execution makes it impossible to run longer simulations for different parameters multiple times.

The results of the investigation are visible in Figures 4.6 and 4.7. We can see that the Velocity Verlet algorithm - ever dependable, as always - shows minimal change in the energy of the system over 500 iterations. The reason for this behaviour is that the Velocity Verlet shows a property known as symplecticity. A discussion on what is symplecticity, and what it entails is beyond the scope of this project. However, the keen reader is can find a detailed, yet approachable, discussion in [12]. As the solutions of a Hamiltonian system are also symplectic, we have that the Velocity Verlet performs well as a numerical approximation scheme due to its symplecticity. Nonetheless, we can see that the total energy is not constant throughout the run - fluctuations in energy around the mean are shown in Figures 4.6. This gives us that the Velocity Verlet algorithm does not preserve

the energy of the system (energy is not constant throughout the simulation), but rather conserves. We quote [**?**]

Theorem 4.8. *For a Hamiltonian $H(p,q)$, the total energy of a numerical solution (p_n, q_n) of the Velocity Verlet method satisfies $|H(p_n,q_n) - H(p_0,q_0)| \leq Ch^2$, where C is a constant independent of h.*

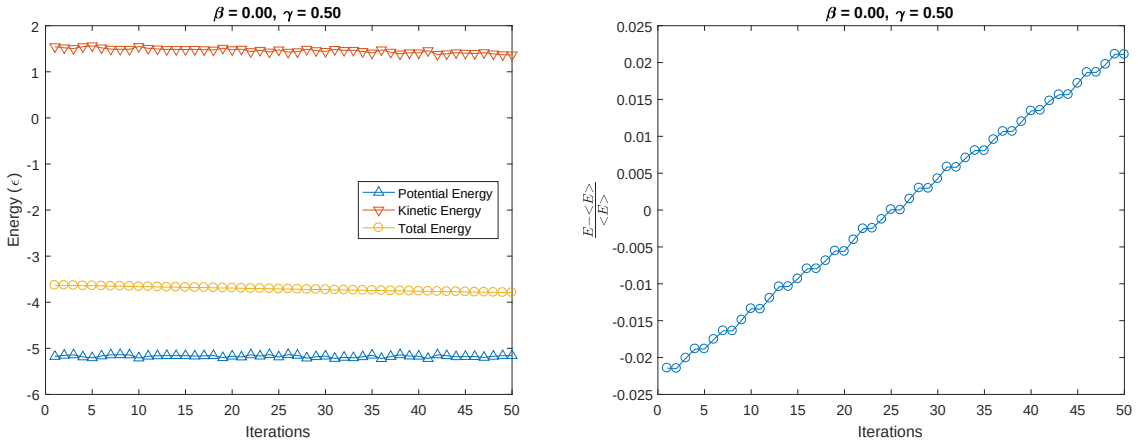

Figure 4.6: Energy time series for Velocity Verlet method

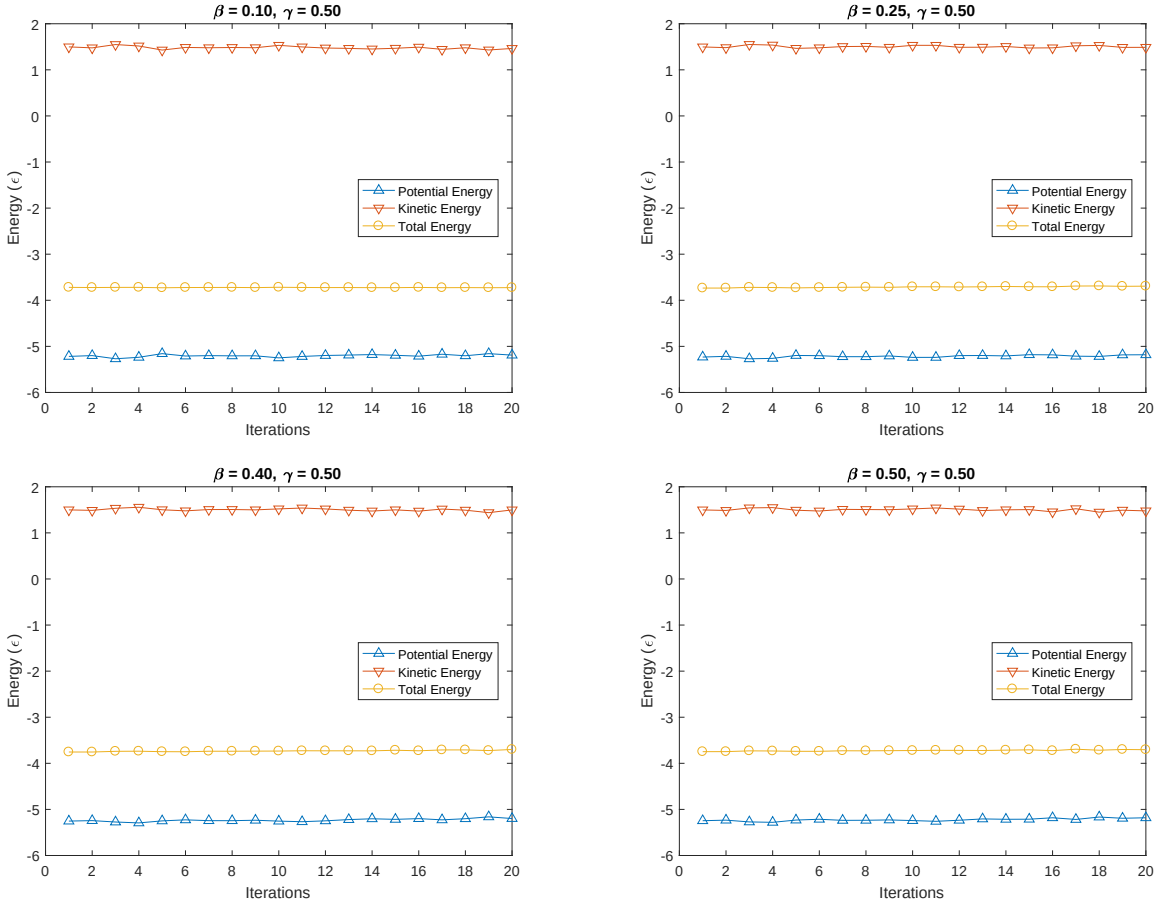

Figure 4.7: Energy time series for implicit Newmark Beta methods with $\gamma = 0.50$

Figure 4.7, however, brings bad news. We see that the graphs depict energy being conserved for every value of β tested. This contradicts the quote above, which states that the implicit Newmark Beta methods are not designed to preserve energy. This highlights that the simplification in our initial configuration could have actually caused problems such that the system may not have been properly equilibrated before it was sampled. It is untenable to run simulations with the initial configurations defined in Table 4.1 for all permissible values of β. Thus, we need to consider a way to reduce the possible candidates that we review. Looking at literature, the trapezoidal rule $\left(\beta = 0, \gamma = \frac{1}{2}\right)$ is mentioned quite often - with [1] and [5] showing that the trapezoidal equation convserves energy balance. Figure 4.8 shows the results of running the simulation with unmodified initial configurations with the Trapezoidal Rule, and it seems to corroborate the observation that the Trapezoidal Rule will conserve total energy. How does this observation then not contradict the results of [13]? This is because the authors of [13] were referring to case of an arbitrary nonlinear Hamiltonian system - that "*For an arbitrary nonlinear Hamiltonian system, the central difference method is the only symplectic scheme within the family of Newmark algorithms.*"

Before concluding that the Trapezoidal Rule conserves energy for the Hamiltonian in question, we need to confirm that the observations are not similar to those in Figure 4.7. As the last trial took almost 2200 seconds to compute, we cannot use for this technique. Instead, we first equilibrate the system using the Velocity Verlet algorithm, and provide the equilibrated state as the initial state to the methods being investigated. As the Velocity Verlet algorithm conserves total energy, the total energy of the state being provided will not be too different from the total energy of the initial state. So, we are using the unmodified initial configurations, except that $N_e = 0$ for the implicit Newmark Beta method. Figures 4.9 and 4.10 show the results of the simulation on two different values of β - 0.10 and 0.40. Both these figures depict total energy being conserved over the long run. In fact, the behaviour of both these systems is extremely similar to the behaviour of the system with $\beta = 0.25$, which raises the question - does β have no effect on the solutions of the system? This problem can be related to our handling of β - in the Newton-Raphson Iterations, we multiplied β by h^2 and $\mathbf{J}_{\mathbf{a}_{i+1}}$ - both very small in magnitude. This would have masked the contributions of β to the solutions of the system.

Hence, we cannot state with certainty whether the Trapezium Rule - or any other implicit Newmark Beta rules - conserve total energy. We can, however, state that the Velocity Verlet algorithm is symplectic, and hence, conserves the total energy in the system.

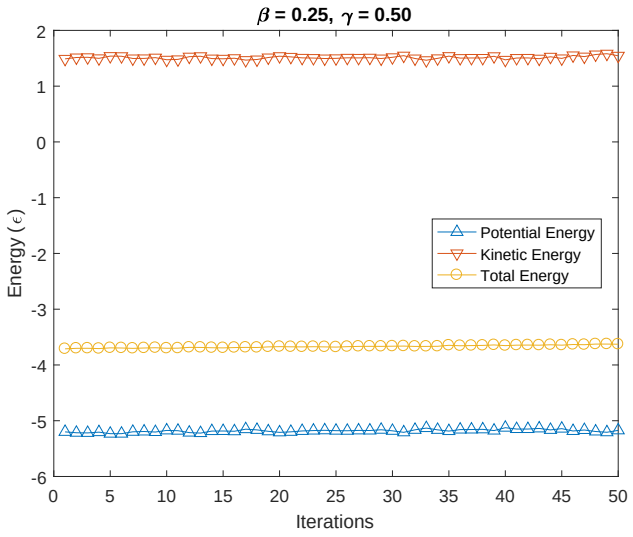

Figure 4.8: Energy time series for Newmark Beta methods with $\beta = 0.25$ and $\gamma = 0.50$ under unmodified initial configurations

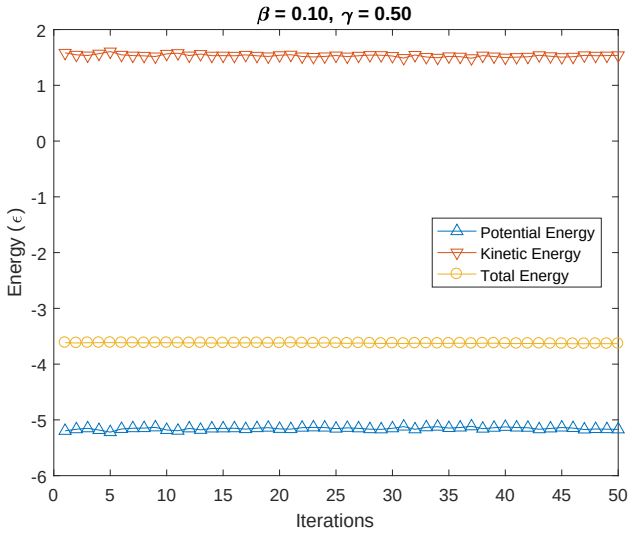

Figure 4.9: Energy time series for Newmark Beta methods with $\beta = 0.10$ and $\gamma = 0.50$ under unmodified initial configurations

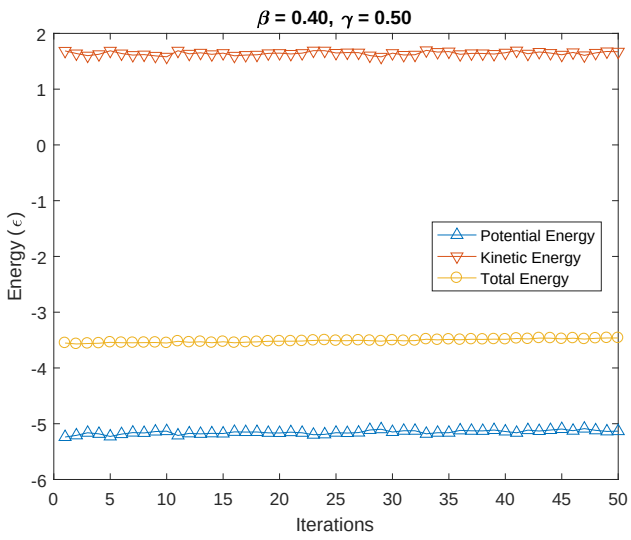

Figure 4.10: Energy time series for Newmark Beta methods with $\beta = 0.40$ and $\gamma = 0.50$ under unmodified initial configurations

5 Conclusion

The Velocity Verlet algorithm clearly stands out from the rest of the algorithms. It is the only method that conserves all four properties of time reversibility, total linear momentum, total angular momentum, and total energy. Any other explicit Newmar Beta scheme conserves only linear momentum, a property common to the entire family of Newmark Beta methods. As $\gamma \neq \frac{1}{2}$ introduces energy drift in the total energy of the system (proportional to $\gamma - \frac{1}{2}$), we concentrate only on the family of Newmark Beta schemes with $\gamma = \frac{1}{2}$ and $\beta \in \left[0, \frac{1}{2}\right]$ - we will call this family the reversible Newmark Beta methods, from Claim 4.2. However, we saw in practice that the schemes did not behave as expected on time reversibilty. While we cannot look at the mean pointwise absolute error in positions and velocities as a good metric (geometric numerical integration does not preserve individual trajectories - the velocity of a particle fluctuates throughout the simulation), we cannot ignore that the floating point arithmetic tends to include errors. Furthermore, the errors compound as the value of a force on a particle derived at the start of the simulation influences the behaviour of the particle for the rest of the simulation. Further focus on the time reversibility problem and ways to reduce this error are looked at in [6] - an interesting suggestion is to not scale length by the value of σ, but rather by the inter-particle distance in the initial cube lattice strucutre. This would then make each particle have an integer displacement, rather than a floating point one.

Only the explicit case of the reversible Newmark Beta methods preserves angular momentum - or quadratic first integrals of the form defined in Theorem 4.6. However, this does not hold in practice, due to presence of periodic boundary conditions.

The case of total energy conservation is peculiar. While we can definitely say the Velocity Verlet algorithm conservers energy (to the order of $\mathcal{O}(h^2)$), we cannot make or deny this claim for any other reversible method due to a couple of reasons. Firstly, the computationally intensive nature of Newton-Raphson iteration prevents us from running longer trials to test the hypothesis. In the three cases that we did run for longer time frames, no distincitve difference in graphs or data was visible. We suspect that the Newton-Raphson iteration is not the best tool for computing the Newmark Beta algorithms. It is computation and memory exhaustive - the 2592×2592 Jacobian takes 53.7 MB of memory to be stored - and could be the reason behind the translated reversed positions of particles under the implicit reversible Newmark Beta methods.

Further steps from here would be to focus mainly on improving the runtime of the implicit Newmark Beta methods. As forces do not deviate by much between trials, we can generate a table of most common forces and use that to lookup the inter-particle force instead of computing them each time. We can do the same for the Jacobian, reusing it over a few trials to make computations faster. However, this could lead to added errors in the system. One way to make the program run faster would be to avoid using Newton-Raphson iterations completely, but no study has been done by us to compare how the predictor-corrector algorithms fare in comparison on the grounds of accuracy and speed of execution.

An interesting study would be to find the optimal initial configuration. We chose a lot of the parameters such as the cut-off distance and the update interval of neighbour lists on the basis of empirical evidence on a small sample, rather than an extensive exploration backed with mathematical fact. Understanding how varying the parameters influence the various results of the simulation would help identify the source of errors in the simulation.

A Appendix

A.1 Lotka-Volterra Solutions Code

```
T = 20;
N = 1000;
h = T/N;

% Initializing the arrays to store solutions over time.
Z_exp = zeros(2,N); Z_imp = zeros(2,N); Z_symp = zeros(2,N);

% Lotka-Volterra Equations
f = @(x) [x(1)-x(1)*x(2); x(1)*x(2)-2*x(2)];
df = @(x, y) [1-x(2); x(1) - 2];

% Newton-Raphson iteration for implicit methods
F_IE = @(x, y0, h) x - y0 - h*f(x);
DF_IE = @(x, h) eye(2) - h*df(x);
N_IE = @(y0, h) y0+h - (DF_IE(y0+h, h))^-1*F_IE(y0+h,y0,h);

% Initial value (2,4).
ysolv_EE = [2;4]; ysolv_IE = [2;4]; ysolv_symp = [2;4];
Z_exp(:,1) = ysolv_EE; Z_imp(:,1) = ysolv_IE; Z_symp(:,1) = ysolv_symp;


for i = 2: N
   ysolv_EE = ysolv_EE + h*f(ysolv_EE);
   ysolv_IE = N_IE(ysolv_IE, h);

   % Symplectic Euler variant: explicit in u, implicit in v
   Z_symp(1,i) = Z_symp(1,i-1)/(1-h*(1-Z_symp(2,i-1)));
   Z_symp(2,i) = Z_symp(2,i-1)*(1+h*(Z_symp(1,i)-2));

   Z_exp(:,i) = ysolv_EE;
   Z_imp(:,i) = ysolv_IE;

end

% Phase planes of Lotka-Volterra equations
x = 0.2:0.2:max(Z_exp(1,:));
y = 0.2:0.2:max(Z_exp(2,:));
[X,Y] = meshgrid(x,y);
Z = X - 2*real(log(X)) + Y - real(log(Y));

% Plotting the numerical solutions beside each other
ax1 = subplot(1,3,1);
plot(ax1, Z_exp(1,:), Z_exp(2,:),'b')
hold on;
plot(ax1, [2], [4], 'o');
hold on;
contour(ax1, X,Y,Z,'k:');
title(ax1,'Explicit Euler');
xlabel(ax1, 'u');
ylabel(ax1, 'v');

ax2 = subplot(1,3,2);
```

```
53 plot(ax2, Z_imp(1,:), Z_imp(2,:),'b')
54 hold on;
55 plot(ax2, [2], [4], 'o');
56 hold on;
57 contour(ax2, X,Y,Z,'k:');
58 title(ax2,'Implicit Euler');
59 xlabel(ax2, 'u');
60 ylabel(ax2, 'v');
61
62 ax3 = subplot(1,3,3);
63 plot(ax3, Z_symp(1,:), Z_symp(2,:),'b')
64 hold on;
65 plot(ax3, [2], [4], 'o');
66 hold on;
67 contour(ax3, X,Y,Z,'k:');
68 title(ax3,'Symplectic Euler');
69 xlabel(ax3, 'u');
70 ylabel(ax3, 'v');
```

A.2 Convergence Tests on Pendulum Problem Code

```
% Code adapted from A. Paganini, Numerical Solutions of DEs I.

% Pendulum problem initial configuration
l = 5; g = 10; T = 5;
theta = 0;
v_theta = 5;
y0 = [theta; v_theta];

f = @(y) [y(2); -g*(1/l)*sin(y(1))];

% Reference solution with MATLAB function ode45
opts = odeset('AbsTol', 1e-12, 'RelTol', 1e-12);
[~, yref] = ode45(@(t,y) f(y), [0, T], y0, opts);
yref_final = yref(end,:).';

N = 2.^(4:10); hsave = zeros(size(N));
err = zeros(size(N));
figure_num = 0;
beta = 1.00; % we vary this value as preferred

% For each value of gamma, repeat the pendulum problem for different step sizes h
for gamma =0:0.25:1
    for i = 1:length(N)
        h = T/N(i); hsave(i) = h;
        x = theta; v = v_theta;

        for jj = 1:N(i)
            [x, v] = newmark_beta(beta, gamma, x, v, h, ...
                                  @update_acceleration, @d_acceleration);
         end

        err(i) = norm([x; v] - yref_final);
    end

    % Plot the graph of convergence
    loglog(hsave, err, '*-', 'linewidth', 2)
    ylim([10^-5 10^1]);
    xlabel('log(h)');
    ylabel('log(err)');
    title('\beta = 1.00');
    hold on;
    % Find (asymptotic) slope of the graph plotted
    fit = polyfit(log(hsave(end-3:end)), log(err(end-3:end)), 1);
    fprintf('The convergence order of the Newmark-Beta method with \beta: %f ...
             and \gamma: %f is %1.2f\n', beta, gamma, fit(1));
end
hold off;
legend('0.00','0.25','0.50','0.75','1.00');
title(legend,'\gamma values');
legend('Location','northwest');
```

```
% Define the acceleration function and its derivate (for Newton-Raphson iteration)
function new_acceleration = update_acceleration(x)
    l = 5; g = 10;
    new_acceleration = -g*(1/l)*sin(x);
end

function derivative_acceleration = d_acceleration(x)
   l = 5; g = 10;
   derivative_acceleration = -g*(1/l)*cos(x);
end
```

```
function [x_new, v_new] = newmark_beta(beta, gamma, x_i, v_i, h, ...
                                       update_acceleration, d_acceleration)
    a_i = update_acceleration(x_i);
    y_i = [v_i; x_i];

    % Newton-Raphson iteration
    f = @(z, v_i, a_i, h) [(1-gamma)*a_i + gamma*update_acceleration(z); ...
                   v_i + 0.5*h*((1-2*beta)*a_i + 2*beta*update_acceleration(z))];
    Df = @(z, h) [zeros(size(z)) h*gamma*z; zeros(size(z)) 0.5*beta*h*h*z];
    F = @(y_j, y_i, h) y_j - y_i - h*f(y_j(2), v_i, a_i, h);
    DF = @(y_j, h) eye(size(y_j, 1)) - Df(d_acceleration(y_j(2)), h);
    N = @(y_i, h) y_i+h - (DF(y_i+h, h))^-1*F(y_i+h,y_i,h);

    % New solutions
    y_j = N(y_i, h);
    x_new = y_j(2);
    v_new = y_j(1);

end
```

A.3 Lennard-Jones Potential Graph Code

```
% Plots the Lennard-Jones potential curve

r = 0:0.01:5;
inv_r6 = r.^-6;
inv_r12 = inv_r6.^2;
V = 4*(inv_r12-inv_r6);
plot(r(1,:),V(1,:),'b');
xlim([0 3]);
ylim([-1.5 2]);
hold on;
plot(r,zeros(size(r,2)),'k');
set(gca,'xtick',[],'ytick',[0]);
ylabel('V');
xlabel('r','fontweight','bold');
```

A.4 Molecular Dynamics: Implicit Methods Code

```
% Adapted from http://www.cchem.berkeley.edu/chem195/index.html and ...
%              https://github.com/brucefan1983/simple-md-matlab
tic;
[num_particles, epsilon, sigma, r_cutoff, mass, density, kB, temperature, ...
                      h, N_e, N_f, N_s, N_n, beta, gamma] = initialize_params();

length_cube = find_cube_length(num_particles, density);
% Adding space to the top given how the lattice is created in initialize_cube.m
diff = length_cube/((2*num_particles)^(1/3));
coordinates = initialize_cube(num_particles, length_cube-diff);
velocities = initialize_velocities(num_particles, mass, kB, temperature);

% Calculate initial neighbours_list
[neighbours_list, num_neighbours_list] = find_neighbours(num_particles, ...
                                         coordinates, length_cube, r_cutoff);
[forces, ~, jacobian_matrix] = find_forces(num_particles, epsilon, sigma, ...
                                         coordinates, length_cube,
                                         neighbours_list, num_neighbours_list);
energy = zeros(N_f/N_s, 3);
for i = 1:(N_e+N_f)
    % Update coordinates and velocities
    acceleration = forces/mass;
    seed_coordinates = coordinates+h;
    F = seed_coordinates - coordinates - h*velocities - ...
        0.5*h^2*((1-2*beta)*acceleration);
    [forces, ~, jacobian_matrix] = find_forces(num_particles, epsilon, sigma,...
                                          seed_coordinates, length_cube,
                                          neighbours_list, num_neighbours_list);
    acceleration = forces/mass;
    F = F - 0.5*h^2*(2*beta*acceleration);
    DF = eye(3*num_particles) - beta*h^2*jacobian_matrix;
    delta_coordinates = DF\(-F);
    coordinates = coordinates + delta_coordinates;

    velocities = velocities + h*(1-gamma)*acceleration;
    [forces, potential_energy, jacobian_matrix] = ...
                                find_forces(num_particles, epsilon, sigma, ...
                                        coordinates, length_cube, ...
                                        neighbours_list, num_neighbours_list);
    acceleration = forces/mass;
    velocities = velocities + h*gamma*acceleration;

    if (i <= N_e)
       % Equilibration stage
       scaling_factor = (3*kB*num_particles*temperature)/ ...
                        (mass*sum(sum(velocities.^2,2)));
       velocities = velocities * sqrt(scaling_factor) ;
    elseif (mod(i, N_s) == 0)
       % Take sample
       energy_index = (i-N_e)/N_s;
       energy(energy_index, 1) = potential_energy/num_particles;
       energy(energy_index, 2) = 0.5*(mass*sum(sum(velocities.^2,2)))...
                                 /num_particles;
    end
```

```
55
56      if (mod(i, N_n) == 0)
57          % Refresh neighbours list
58          [neighbours_list, num_neighbours_list] = ...
59                find_neighbours(num_particles, coordinates, length_cube, r_cutoff);
60      end
61  end
62
63  % Adjust potential energy
64  r_cutoff_6 = r_cutoff^6;
65  sigma_6 = sigma^6;
66  shifting_potential_term = 4*(sigma_6/r_cutoff_6)*((sigma_6/r_cutoff_6) - 1);
67  energy(:,1) = energy(:,1) - shifting_potential_term;
68  energy(:,3) = energy(:,1) + energy(:,2);
69
70  steps = linspace(1,N_f/N_s,N_f/N_s);
71  figure;
72  plot(steps, energy(:,1), '-^', steps, energy(:,2), '-v', steps, energy(:,3), '-o');
73  xlabel('Iterations'); ylabel('Energy (\epsilon)');
74  legend('Potential Energy', 'Kinetic Energy', 'Total Energy','Location','east');
75  toc;
```

```
1   function [num_particles, epsilon, sigma, r_cutoff, mass, density, kB, ...
2             temperature, h, N_e, N_f, N_s, N_n, beta, gamma] = initialize_params()
3
4       num_particles = 864;
5       epsilon = 1;
6       sigma = 1;
7       r_cutoff = 2.5;
8       mass = 48;
9       density = 38.74457642;
10      kB = 1;
11      temperature = 1;
12      h = 0.032;
13      N_e = 100;
14      N_f = 500;
15      N_s = 10;
16      N_n = 15;
17      beta = 0.00;  % [0,0.5]
18      gamma = 0.50; % [0,1]
19  end
```

```
1   % Creates an FCC crystal
2   function coordinates = initialize_cube(num_particles, length_cube) % FCC crystal
3       one_side = (2*num_particles)^(1/3);
4       coordinates = zeros(3*num_particles,1);
5       i = 1;
6       particle_spacing = length_cube / (one_side - 1);
7       for x = 0:particle_spacing:length_cube
8           for y = 0:particle_spacing:length_cube
9               if (mod(x+y,2*particle_spacing) == 0)
10                  for z = 0:2*particle_spacing:length_cube
11                      coordinates(i:i+2) = [x,y,z]';
12                      i = i + 3;
13                  end
14              else
```

```
                for z = particle_spacing:2*particle_spacing:length_cube
                  coordinates(i:i+2) = [x,y,z]';
                  i = i + 3;
                end
            end
        end
    end
end
```

```
function length_cube = find_cube_length(num_particles, density)
    length_cube = (48 * num_particles / density)^(1/3);
end
```

```
function velocities = initialize_velocities(num_particles, mass, kB, temperature)
    % Seeding to maintian reproducibility
    seed = 7;
    rng(seed);

    % Generate velocities using Maxwell distribution, and remove linear momentum
    velocities_3d = normrnd(0, (kB*temperature/mass)^0.5, num_particles, 3);
    velocities_3d = velocities_3d - sum(velocities_3d)/num_particles;
    %Arrange as a column vector
    velocities = [velocities_3d(:,1)'; velocities_3d(:,2')'; velocities_3d(:,3)'];
    velocities = velocities(:);
end
```

```
function [neighbours_list,num_neighbours_list] = ...
          find_neighbours(num_particles, coordinates, length_cube, r_cutoff)
    neighbours_list = zeros(num_particles, num_particles -1);
    num_neighbours_list = zeros(num_particles, 1);
    r_cutoff_2 = r_cutoff*r_cutoff;
    for i = 1:3:3*num_particles
        num_neighbours_i = 0;
        i_index = (i+2)/3;
        for j = i+3:3:3*num_particles
            j_index = (j+2)/3;
            diff_r = coordinates(i:i+2) - coordinates(j:j+2);
            diff_r = diff_r - length_cube*round(diff_r/length_cube);
            diff_r_2 = sum(diff_r.^2);
            if diff_r_2 < r_cutoff_2
                num_neighbours_i = num_neighbours_i + 1;
                neighbours_list(i_index, num_neighbours_i) = j;
                num_neighbours_j = num_neighbours_list(j_index);
                num_neighbours_j = num_neighbours_j + 1;
                neighbours_list(j_index, num_neighbours_j) = i;
                num_neighbours_list(j_index) = num_neighbours_j;
            end
        end
        num_neighbours_list(i_index) = num_neighbours_i;
    end
    % Re-adjust size of neighbours_list to save memory
    neighbours_list = neighbours_list(:,1:max(num_neighbours_list));
end
```

```
function [forces, potential_energy, jacobian_matrix] = ...
          find_forces(num_particles, epsilon, sigma, ...
          coordinates, length_cube, neighbours_list, num_neighbours_list)
    potential_energy = 0;
    sigma_6 = sigma^6;
    forces = zeros(3*num_particles, 1);
    jacobian_matrix = zeros(3*num_particles);

    for i = 1:3:3*num_particles
        i_index = (i+2)/3;
        for neighbour = 1:num_neighbours_list(i_index)
            j = neighbours_list(i_index, neighbour);
            if (j > i)
                diff_r = coordinates(i:i+2) - coordinates(j:j+2);
                diff_r = diff_r - length_cube*round(diff_r/length_cube);
                dist_r_2 = sum(diff_r.^2);
                dist_r_6 = dist_r_2^3; dist_r_8 = dist_r_6 * dist_r_2;
                dist_r_10 = dist_r_8 * dist_r_2;

                force_factor = (sigma_6/dist_r_8)*(sigma_6/dist_r_6 - 0.5);
                forces(i:i+2) = forces(i:i+2) + force_factor*diff_r;
                forces(j:j+2) = forces(j:j+2) - force_factor*diff_r;

                jacobian_factor = (sigma_6/dist_r_10)* ...
                                  ((-14*sigma_6/dist_r_6) + 4);
                a = diff_r * diff_r';
                jacobian_force_block = force_factor*eye(3) + ...
                                       jacobian_factor*(diff_r * diff_r');

                jacobian_matrix(i:i+2,i:i+2) = ...
                            jacobian_matrix(i:i+2,i:i+2) + jacobian_force_block;
                jacobian_matrix(j:j+2,j:j+2) = ...
                            jacobian_matrix(j:j+2,j:j+2) + jacobian_force_block;
                jacobian_matrix(i:i+2,j:j+2) = ...
                            jacobian_matrix(i:i+2,j:j+2) - jacobian_force_block;
                jacobian_matrix(j:j+2,i:i+2) = ...
                            jacobian_matrix(j:j+2,i:i+2) - jacobian_force_block;

                potential_energy = potential_energy + (sigma_6/dist_r_6)*...
                                                      ((sigma_6/dist_r_6) - 1);
            end
        end
    end
    forces = 48*epsilon*forces;
    jacobian_matrix = 48*epsilon*jacobian_matrix;
    potential_energy = 4*epsilon*potential_energy;
end
```

References

[1] D. Doyen, A. Ern, and S. Piperno. Time-integration schemes for the finite element dynamics signorini problem. *SIAM Journal on Scientific Computing*, 33(1):223–249, 2011.

[2] E.Harier, C. Lubich, and G. Wanner. Geometric numerical integration illustrated by the stormer-verlet method. *Acta Numerica*, pages 399–450, 2003.

[3] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. Academic Press, 2001.

[4] Ernst Hairer, Gerhard Wanner, and Christian Lubich. *Examples and Numerical Experiments*, pages 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[5] S. Krenk. Energy conservation in newmark based time integration algorithms. *Computational Methods in Applied Mechanics and Engineering*, 195:6110–6124, 2005.

[6] D. Levesque and L. Verlet. Molecular dynamics and time reversibility. *Journal of Statistical Physics*, 72(3):519–537, Aug 1993.

[7] Nathan M. Newmark. Computation of dynamic structural response in the range approaching failure. In *Proceedings of the Symposium on Earthquake and Blast Effects on Structures*, Los Angeles, 1952. Earthquake Engineering Research Institute.

[8] Nathan M. Newmark. A method of computation for structural dynamics. *American Society of Civil Engineers*, EM 3, 1959.

[9] A. Paganini. Numerical solution of differential equations i. Lecture Notes, nov 2017.

[10] Aneesur Rahman. Correlations in the motion of atoms in liquid argon. *Physical Review*, 136(2A):405–411, 1964.

[11] D. C. Rapaport and Dennis Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, 2004.

[12] J. M. Sanz-Serna. Symplectic integrators for hamiltonian problems: an overview. *Acta Numerica*, 1:243–286, 1992.

[13] J.C. Simo, N. Tarnow, and K.K. Wong. Exact energy-momentum conserving algorithms and symplectic schemes for nonlinear dynamics. *Computer Methods in Applied Mechanics and Engineering*, 100:63–116, oct 1992.

[14] Loup Verlet. Computer experiments on classical fluids. i. thermodynamical properties of lennard -jones molecules. *Physical Review*, 159(1):98–103, 1967.