UCD School of Physics

PHYC30320 – Advanced Laboratory I

Langevin Dynamics

By Joseph Anderson

Start Date: 13/10/2022

Table of Contents

Abstract

Langevin dynamics extends molecular dynamics to allow for solvent effects like frictional jostling of molecules and occasional high velocity collisions of molecules. This experiment aimed to study the one-dimensional motion of a particle in the Lennard-Jones potential. Using the velocity Verlet algorithm the trajectory of a particle in this system will be found. I attempted to show that the motion of a particle in this system is periodic and thus obeys the law of the conservation of energy. However, due to a systematic error the trajectory was found to be divergent. The average velocity of the particle in this system was found to be (-3.73 ± 0.43) x$10^{-4} m/s$ with a calculated observed temperature of 7.45x10^{22}K. The potential of mean force of the system was also found and compared to the Lennard-Jones potential. Results from this paper seem to prompt for further research into the limitation of algorithms that can be used to solve the Langevin equations of motion.

Keywords: Langevin dynamics; velocity Verlet algorithm; Brownian motion; Lennard-Jones potential.

1 Introduction

Langevin dynamics is a method of describing the motion of Brownian particles [1]. Paul Langevin provided an alternative to Newtonian dynamics that could model Brownian particles in a fixed potential by introducing a stochastic force which accounted for omitted degrees of freedom [2]. A molecular system in the real world is unlikely to be present in a vacuum. Hence, there are other factors that influence the system. Jostling of solvent air molecules causes friction and the occasional high velocity collision will perturb the system [3]. Langevin dynamics therefore extends molecular dynamics to allow for these effects through the use of a stochastic time dependent force which can be used to simulate the time evolution of molecular systems [4]. The one-dimensional motion of these Brownian particles will be studied under the influence of the Lennard-Jones potential.

This experiment aims to model the Brownian particles in the Lennard-Jones potential using the velocity Verlet algorithm to show that the trajectory is periodic in order to obey the conservation law of energy. Finding the trajectory of the particle will allow the position and velocity distribution histograms to be studied. Through position histogram analysis from this simulation the potential of mean force can be found. This will then be compared to the Lennard-Jones potential function.

2 Theory

2.1 Background of Langevin Dynamics

Brownian motion was first discovered in 1827 by botanist Robert Brown. While studying pollen particles floating in water Brown observed particles in the pollen grains executing a jittery random motion [5]. However, the origin of this motion remained unknown. The explanation of this Brownian motion was first given quantitatively by Einstein and Smoluchowski in 1905 [6] [7]. Both Einstein and Smoluchowski explained the motion in detail by accurately predicting the irregular, random motions of the particles [8]. Later that year, Jean-Baptiste Perrin then verified experimentally Einstein's explanation of Brownian motion [9]. However, Einstein's theory despite being experimentally verified seemed far removed from the Newtonian dynamics of particles [10]. It was Langevin who in 1908 introduced the concept of the equation of motion of some random variable (in this case a Brownian particle) [11]. Langevin was able to bridge the gap from Newtonian dynamics to Einstein's theory by splitting the force felt by Brownian particles in two: one part being the Stoke's drag and the other part being a random time dependent force that represents the collisions of the atoms [12].

2.2 Langevin Equation

The theory proposed by Langevin lead to a stochastic differential equation of motion that mathematically describes the forces experienced by a Brownian particle [2]. In our case we assume the particle of mass m with position coordinate x=x(t) is acting under the influence of a fixed potential. The resulting Langevin equation is:

$$m\ddot{x} = -\frac{\partial U(x)}{\partial x} - \gamma m\dot{x} + \sqrt{2\gamma k_B T m}\ R(t) \qquad (eqn\ 1)\ [13]$$

U(x) is the particle interaction potential and so its negative derivative is the force calculated from the particle interaction potential. The dot is a time derivative so that $\dot{x}$ is the velocity and $\ddot{x}$ is the acceleration. T is the temperature, k_B is Boltzmann's constant and γ is the frictional damping constant. The function R(t) is a delta correlated stationary Gaussian process with zero mean, satisfying the following equations where δ is the Dirac delta [14]:

$$\langle R(t)\rangle = 0 \qquad (eqn\ 2)$$

$$\langle R(t)R(t')\rangle = \delta(t-t') \qquad (eqn\ 3)$$

2.3 Lennard-Jones Potential

We will observe Brownian particles in the Lennard-Jones potential. The Lennard-Jones potential was initially proposed by Sir John Edward Lennard-Jones in 1924 to describe the cohesive energy of crystals of argon [15]. The now conventional Lennard-Jones potential was later proposed in 1931 after he had derived that the dispersion interaction between atoms decays as r^{-6} [16]. Thus, the Lennard-Jones potential describes the potential energy of interaction between two non-bonding atoms or molecules based on their distance of separation [17]. The Lennard-Jones potential is given by the formula:

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \qquad (eqn\ 4)$$

where ε denotes the depth of the well and σ is the interparticle distance where the potential changes sign [18]. In our experiment the energy unit will be set equal to the characteristic thermal energy $k_B T = 1$, so that both ε and σ will be equal to 1.

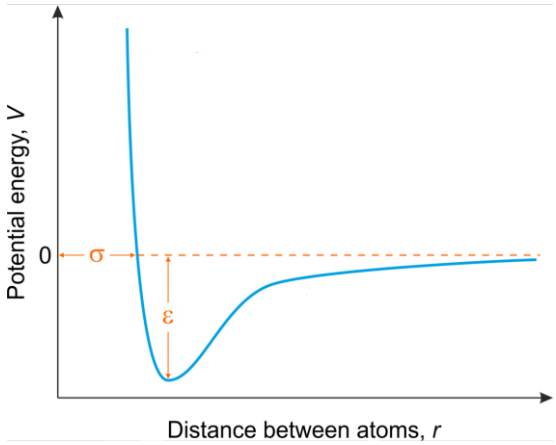


Figure 1 – Graphical representation of the Lennard-Jones Potential [19]. This figure shows the attraction between non-ionic particles denoted by the r^{-6} dependence and the repulsion between non-ionic particles denoted by the r^{-12} dependence. We can also see the effect of the parameters ε and σ.

2.4 The Equipartition Theorem

The equipartition theorem relates the temperature of a system to the average energies of the system. More specifically the equipartition theorem of energy states that molecules in thermal equilibrium contain the same level of average energy with each independent motional degree of freedom [20]. The equipartition theorem can be written as:

$$\langle \frac{1}{2} m v^2 \rangle = \frac{d}{2} k_B T \qquad (eqn\ 5)\ [21]$$

where the brackets $\langle - \rangle$ represents the average value. The constant k_B is the Boltzmann constant and m is the mass of the particle. T is the temperature of the system; v is the velocity of the particles and d represents the degrees of freedom of the system [21].

In our experiment the one-dimensional motion of a particle in the Lennard-Jones potential will be studied. Thus, 1 degree of freedom is assumed for this system.

2.5 The Potential of Mean Force

The free energy changes as a function of reaction coordinates [22]. In our experiment we study the distance between two particles as the reaction coordinate. When a molecular system is in a solvent, the potential of mean force incorporates solvent effects as well as the intrinsic interaction between the two particles dictated by the interaction potential [23].

Using the separation between two particles as the reaction coordinate the potential of mean force can be expressed as:

$$U_{PMF} = -k_B T\ ln(1 + h(r)\ ) \qquad (eqn\ 6)$$

Where h(r) is the position distribution function [24].

3 Computational Methods

The velocity Verlet algorithm will be used to model the particles motion whilst also calculating the velocity for each timestep. Langevin dynamics along with the Lennard-Jones potential will be used to find the acceleration term within our algorithm.

3.1 General Velocity Verlet Algorithm

Langevin's equation of motion can be solved numerically to produce trajectories of the system [25]. The velocity Verlet algorithm is one of the most popular algorithms to solve the system of first-order differential equations provided by Langevin dynamics due to its simplicity. This algorithm is written as:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2}(\Delta t^2)\ a_n \qquad (eqn\ 7)\ [26]$$

$$v_{n+1} = v_n + \frac{1}{2}(a_n + a_{n+1})\ \Delta t \qquad (eqn\ 8)\ [26]$$

where x is the particle's position coordinate, v is the particles velocity and a is the particles acceleration. The timestep of the algorithm is displayed by Δt.

3.2 Acceleration Term using Langevin Dynamics

The acceleration term in our velocity Verlet algorithm will be derived using Langevin dynamics outlined in section 2.2. The energy unit was set equal to the characteristic thermal energy. This implies that $k_B T = 1$ which means $\varepsilon = 1$. Therefore, $\sigma = 1$ and the mass of the particle is equal to 1. This particle is assumed to be under the influence of the Lennard-Jones Potential which means the partial derivative of equation 4 can be put into equation 1.

Hence, equation 1 can be rewritten as:

$$a_n = \frac{48}{{x_n}^{13}} - \frac{24}{{x_n}^{7}} - \gamma v_n + \sqrt{2\gamma}\, R_n(t) \qquad (eqn\ 9)\ [13]$$

where we assume the damping constant γ is at its optimal value of 1.8 [27]. The stochastic term R(t) is a constant that depends on the input time. It accounts for the random high velocity collisions that perturb the system. The "stochastic" package is used to plot a Gaussian Noise process [28]. The properties of this function outlined by equation 2 and 3 is verified in section 4.1.

```
import stochastic.processes.noise as sto

G = sto.GaussianNoise(1)
times = np.arange(0,100 ,0.001)
# Creates an array of the 100,000 steps with timestep 0.001

R_t = G.sample_at(times)
# Generates the corresponding value of the stochastic function of each time

plt.plot(times[1:], R_t)
# Plots the Noise Function
```

Figure 2 – Implementation of the stochastic package to find specific values of the stochastic term for different times. The times() method is utilised in this code as it is particularly useful for plotting the samples obtained by the noise process which is useful in understanding the values given by the noise process [28].

3.3 Implementation of Velocity Verlet Algorithm

The velocity Verlet algorithm can be easily extended to Langevin dynamics by using equation 9 in both equation 7 and 8. This results in:

$$x_{n+1} = x_n + v_n \Delta t + \frac{1}{2}\left(\frac{48}{{x_n}^{13}} - \frac{24}{{x_n}^{7}} - \gamma v_n + \sqrt{2\gamma}\, R_n(t)\right) \qquad (eqn\ 10)$$

$$v_{n+1} = v_n + \frac{1}{2}\left(a_n + \frac{48}{{x_{n+1}}^{13}} - \frac{24}{{x_{n+1}}^{7}} - \gamma v_{n+1} + \sqrt{2\gamma}\, R_{n+1}(t+dt)\right)\Delta t \qquad (eqn\ 11)$$

Equation 11 defines v_{n+1} implicitly. Linear dependency allows for a solution for v_{n+1} in closed explicit form [29]. Thus, equation 11 is rearranged to solve for v_{n+1} and put in the velocity Verlet algorithm in our code. The velocity Verlet algorithm can be best summarised by a flow chart of the steps taken during the algorithm. Thus, the velocity Verlet algorithm can be run in the following way:

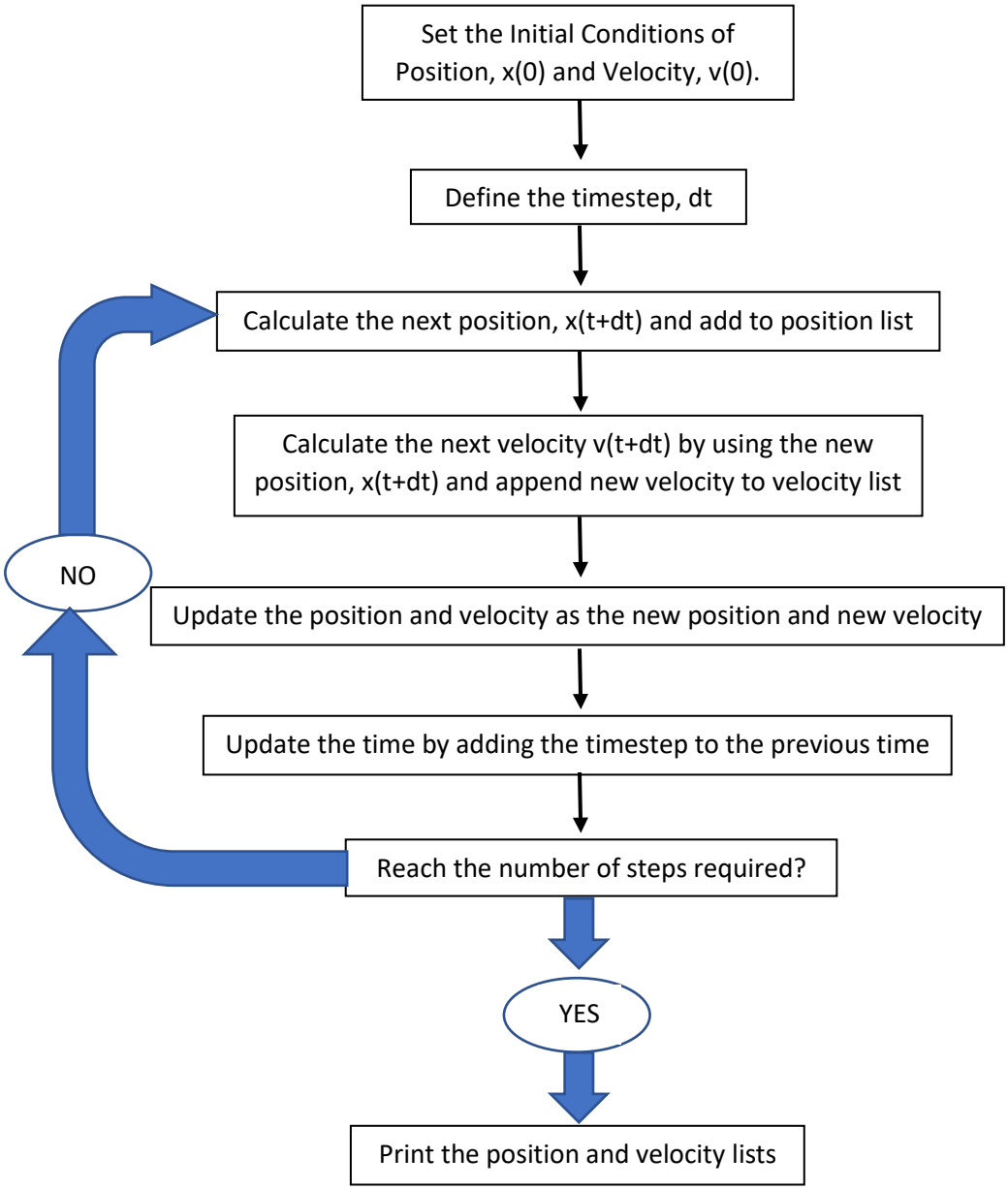


Figure 3 – Flow chart of the molecular system simulation using the velocity Verlet algorithm. Both the velocities and the positions are stored in lists. The loop is repeated for a desired number of steps that can be changed.

3.4 Validation of the Velocity Verlet Algorithm

Equation 9 will have velocity dependence and a random stochastic force however the algorithm can still be checked and validated using a simpler molecular system example. Therefore, to validate the velocity Verlet algorithm the dynamics of a classical one-dimensional simple harmonic oscillator can be simulated to show that the velocity Verlet algorithm produces the exact analytical solution of the system. The potential of a harmonic oscillator is given by:

$$V(x) = \frac{1}{2} k\, x^2 \qquad (eqn\ 12)\ [30]$$

where k is a force constant that measures the stiffness of the spring and x is the displacement of the particle attached to the spring from its equilibrium position [30]. Applying Newton's second law to the system results in an equation of motion of:

$$-\frac{dV(x)}{dx} = m\frac{d^2x}{dt^2} \qquad (eqn\ 13)$$

where m is the mass of the particle that the spring is attached to. This equation can be solved for the exact solution of the position of the particle using equation 12 as the potential of the harmonic oscillator. The velocity of the particle on the spring can be found by taking the derivative of the position solution. This results in a solution of:

$$x(t) = Acos(wt + \Phi) \qquad (eqn\ 14)\ [31]$$

$$v(t) = -wAsin(wt + \Phi) \qquad (eqn\ 15)\ [31]$$

where $w = \sqrt{\frac{k}{m}}$ is the angular frequency of the system [32]. Hence, both equation 14 and 15 are the exact analytical solutions of the equations of motions of the system of a simple spring harmonic oscillator system.

Thus, using the velocity Verlet algorithm that was outlined in section 3.3 the solutions can be shown. The acceleration term changes from the expression given by equation 9. The acceleration term for this harmonic oscillator system is given by:

$$a = -kx \qquad (eqn\ 16)$$

due to the assumption that the particle has a mass of m=1. This means we have an acceleration term with explicit dependence on position. Applying the velocity Verlet algorithm outlined in figure 3 we find the following.

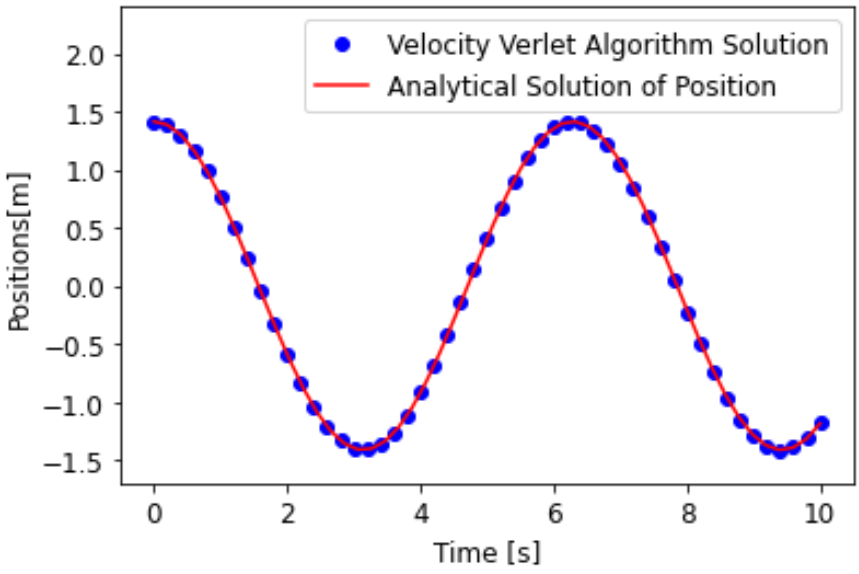


Figure 4 – Shows the position solution of the one-dimensional harmonic oscillator using the velocity Verlet algorithm (blue) in comparison to the analytical solution (red) defined by equation 14. This example had an initial position of $\sqrt{2}$ with a timestep of 0.2 seconds. The particles motion is modelled for 50 steps.

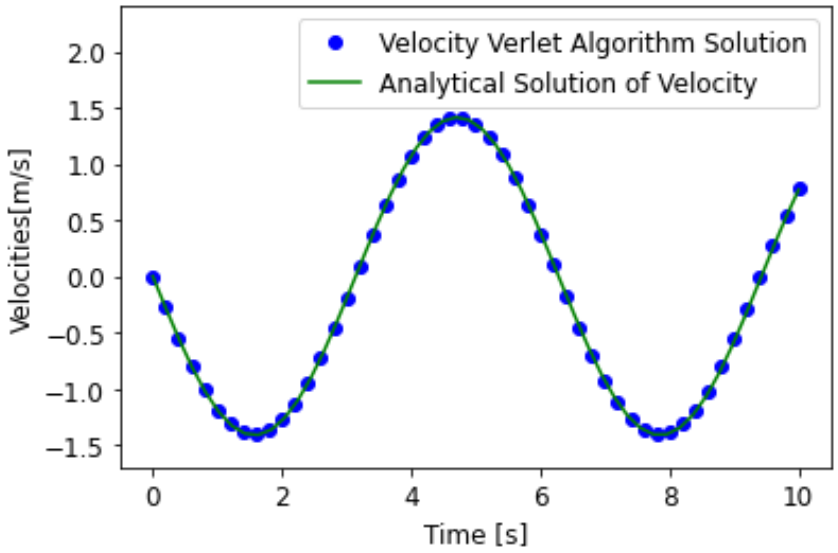


Figure 5 – Shows the velocity solution of the one-dimensional harmonic oscillator using the velocity Verlet algorithm (blue) in comparison to the analytical solution (green) defined by equation 15. This example had an initial position of $\sqrt{2}$ with a timestep of 0.2 seconds. The particles motion is modelled for 50 steps.

From visual inspection of figure 4 and figure 5 it is evident that the velocity Verlet algorithm matches the exact analytical solution of the harmonic oscillator example. This velocity Verlet algorithm had 50 steps of a timestep of 0.2 seconds which is a smaller number of steps and timestep than we will do in our study of particles in the Lennard-Jones potential.

3.5 Accuracy of Velocity Verlet Algorithm

The error analysis of the velocity Verlet algorithm can be derived mathematically using the Taylor expansion of each new position and velocity [33]. This analysis is divided into 2 types of errors - a truncation error which is the error caused by one iteration and a global error which is the cumulative error caused by many iterations of the algorithm [34].

3.5.1 Truncation Error

The truncation error of the position can be found with the direct use of the Taylor expansion of $x(t+\Delta t)$ at $x(t)$ which gives the following equation [35]:

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2}a(t)(\Delta t)^2 + O(\Delta t^3) \qquad (eqn\ 17)\ [35]$$

This is similar to equation 7 however this is explicitly time dependent and also includes the truncation error of the position which is $O(\Delta t^3)$.

The truncation error analysis uses the Taylor expansion of $a(t+\frac{\Delta t}{2})$ and $a(t+\Delta t)$ in the expansion of $v(t)$ and $v(t+\Delta t)$. This results in the expression:

$$v(t+\Delta t) = v(t) + \frac{\Delta t}{2}(\,a(t) + a(t+dt)\,) + O(\Delta t^3) \qquad (eqn\ 18)\ [36]$$

It can be seen that the truncation error of the velocity is $O(\Delta t^3)$.

3.5.2 Global Error

The global error will give the cumulative error on both the position and velocity calculated by the velocity Verlet algorithm [37]. This will give us the error caused by many iterations of the algorithm. Considering equations 17 and 18 the following equations can be derived for a general expression of the global error on n steps:

$$error[x(t + n\Delta t)] = nO(\Delta t^3) \qquad (eqn\ 19)\ [33]$$

$$error[v(t + n\Delta t)] = nO(\Delta t^3) \qquad (eqn\ 20)\ [33]$$

The global error in position between $x(t)$ and $x(t + T)$ where T is the total time we run the algorithm over ($T = n\Delta t$) is:

$$error(x(t + T)) = \frac{T}{\Delta t} O(\Delta t^3) = O(\Delta t^2) \qquad (eqn\ 21)\ [33]$$

A similar calculation is done the produce the global error on the velocity which results also in a value of $O(\Delta t^2)$.

Hence, the errors in the velocity algorithm is concluded by the following table:

Error	Position [x]	Velocity [v]
Truncation Error	$O(\Delta t^3)$	$O(\Delta t^3)$
Global Error	$O(\Delta t^2)$	$O(\Delta t^2)$

Table 1 – Error terms of the velocity Verlet algorithm. The truncation error represents the error on each iteration of the algorithm whereas the global error gives the cumulative error on many iterations of the algorithm. The truncation error is the same for both position and velocity. The global error is also the same for both positions and velocity.

3.6 Advantages and Disadvantages of the Velocity Verlet Algorithm

The velocity Verlet algorithm was first introduced in 1982 as an extension to the general Verlet algorithm [38]. There are many different algorithms that integrate the equations of motion of a system.

3.6.1 Advantages of Velocity Verlet Algorithm

The velocity Verlet algorithm is mathematically equivalent to the general Verlet scheme however it explicitly incorporates velocity [39]. This solves the first time-step problem of the general Verlet algorithm. Velocity Verlet is also not necessarily more memory consuming because it's not necessary to keep track of the velocity at every time step during the simulation [40]. It also has a similar global error as the general Verlet algorithm despite being quicker to initiate [33]. This algorithm is also self-starting using an initial position and an initial velocity which allows easier calculations. The velocity Verlet algorithm is time-reversible which is a desirable numerical property that reflects the physical reality of the simulation problem [41]. The velocity Verlet algorithm is symplectic which is also a good numerical property that will help reflect the physical reality of the system. Symplectic means that the method is guaranteed to conserve the total energy in a conservative simulation[42].

3.6.2 Limitations of Velocity Verlet Algorithm

The velocity Verlet algorithm is easily applied to conservative forces like the classical harmonic oscillator studied in section 3.4. However, unlike Runge-Kutta methods, it is more difficult to introduce velocity dependent forces like damping [43]. The simplicity of the velocity Verlet algorithm

comes at the expense of accuracy and stability in comparison with some other second-order methods. The Runge-Kutta (RK2) method is a comparable second order method that is significantly more accurate and the backward differential formula (BDF2) is a second order method that is much more stable [44].

However, for our use of the algorithm the velocity Verlet algorithm is most suitable as the RK2 and BDF2 can only be used for certain physical processes. Due to the Runge-Kutta method not being a symplectic integrator it cannot be used as an algorithm to integrate the equation of motion in a molecular dynamics simulation [45].

3.7 Choice of Parameters

The particle is studied starting from the position x(0) = 3. This is a logical position to a particle to start within the Lennard-Jones potential as the Lennard-Jones potential goes to infinity as the position gets closer to 0 [46]. This means that the position of 3 will have a defined potential value.

A timestep of Δt = 0.001 seconds will be used for at least 100,000 steps. Similar timesteps are used to solve Newton's equations of motions to plot a trajectory of a Brownian particle in a Lennard-Jones potential [47]. The large nature of the number of steps we perform should give us a better insight into the true particle's trajectory.

4 Results

4.1 Mean of Stochastic Process Used

In order to calculate the acceleration of the particle (during the velocity Verlet algorithm) at specific times using the Langevin equation a stochastic process was required. Physically, this term ensured that the random force of collisions of the particles was accounted for. Hence, as outlined in section 3.2, the stochastic package was used to plot our delta-correlated stationary Gaussian process.

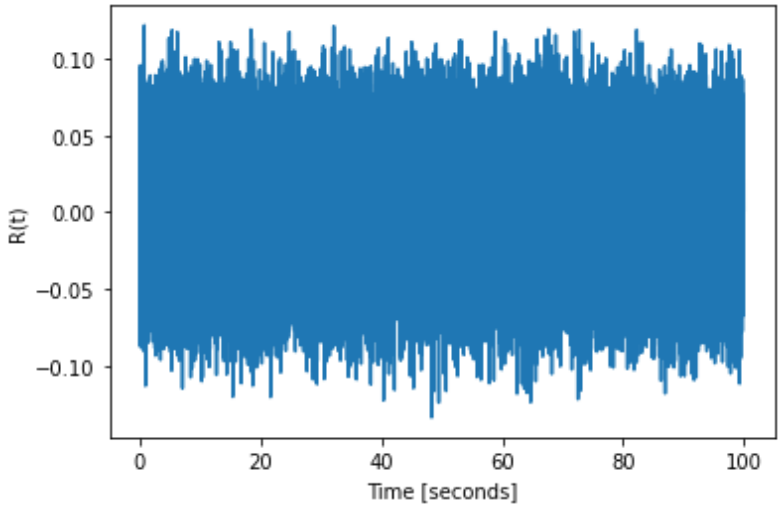


Figure 6 – The delta-correlated stationary gaussian process found using the stochastic package. Physically representing the random force of collisions that perturbs the system's acceleration and so has to be taken into account using Langevin dynamics.

As part of the Langevin equation it was assumed that this function had an expectation value of 0. This can be verified by fitting a Gaussian distribution to the histogram of the stochastic process

displayed in figure 6. Thus, enabling us to calculate the mean value of the function which should be close to being 0.

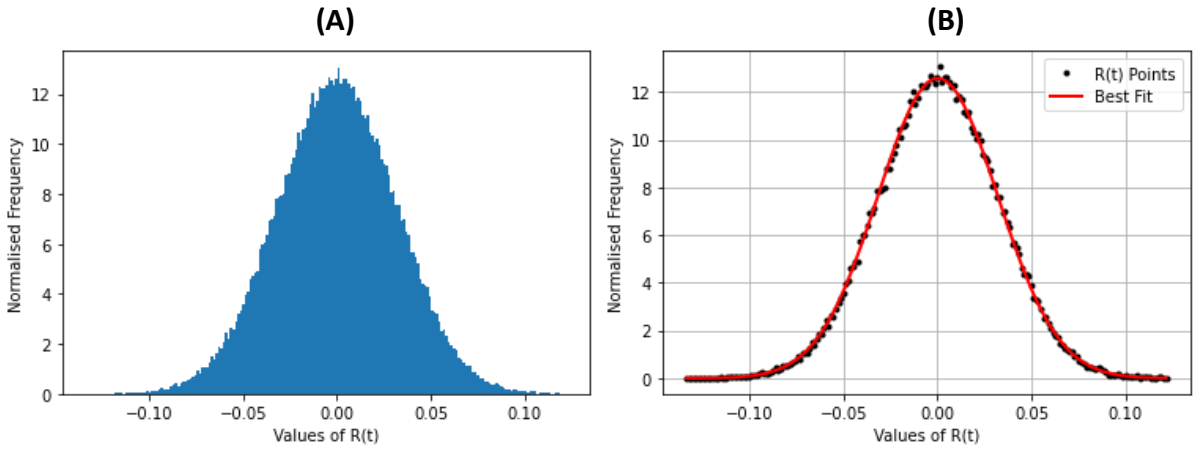


Figure 7 – (A) Shows the normalised histogram of the stochastic function R(t) with 200 bins plotted. This can be visually inspected to infer that it follows a Gaussian distribution. (B) Shows the normalised histogram of R(t) with a Gaussian curve of best fit fitted to it.

The average value of the range each bin covers is taken to be the R(t) value. This is a valid assumption due to the large number of bins used in the plotted histogram [48]. This can then be plotted using the exact values of each amplitude. Thus, a Gaussian function can be curve fitted to match our data using the Gaussian distribution formula:

$$G(x) = Ae^{-\frac{(x-x_0)^2}{2\sigma^2}} \qquad (eqn\ 22)$$

where A is a scaling factor, x_0 is the mean of the distribution and σ is the standard deviation [49]. Using the parameters found by the curve fitting a value for the mean is found to be 0.00019. The uncertainty on the best fit parameters are found using the covariance matrix. Hence, the uncertainty on the mean is found by taking the square root of the [1,1] element. This gives us a value of $4x10^{-5}$ for the uncertainty on the mean.

Thus, the value for the mean of the stochastic process R(t) is found to be $(19 \pm 4)\ x10^{-5}$. This is dimensionless. This is a very small value that can be taken to be 0. Hence, meaning the stochastic process shown in figure 7 is verified to be Gaussian with an expectation value of 0.

4.2 Trajectory of the Particle

The velocity Verlet algorithm can then be used in tandem with Langevin dynamics to observe the time evolution of a particle in the Lennard-Jones potential subjected to solvent effects.

Assumptions:

The magnitude of the damping coefficient γ determines the relative strength of the inertial forces with respect to the random external forces [29]. The stochastic Langevin forces mimics the viscous aspect of solvents which physically corresponds to collisions between the solvent molecules and the solute. Using phase space preobservables on trajectory data it can be shown that the optimal value of the damping coefficient is γ=1.8 [27]. This is close to the heuristic choice for a damping coefficient of γ=1. The particle is modelled assuming that the damping coefficient is γ=1.8. The particle is also assumed to start at rest. This is a valid assumption as the driving force and potential force can be presumed to start influencing the particle as soon as the particle starts its motion [50].

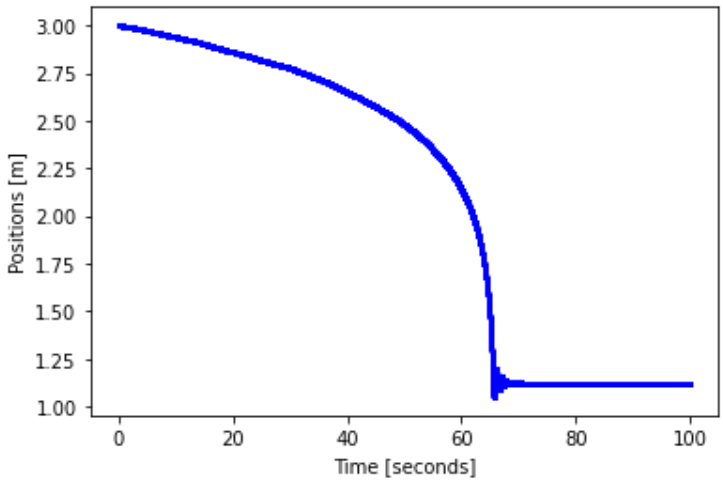


Figure 8 – Time evolution performed by the velocity Verlet algorithm for a single particle in a one-dimensional Lennard-Jones potential from time t=0 to t=100 seconds with a timestep of Δt = 0.001 seconds. This was performed with the particle starting at initial position x(t)=0. The particle is assumed to start at rest. The damping coefficient is assumed to be γ = 1.8.

The particle's motion seems to diverge to a position of approximately 1.12 after a time of around 70 seconds. However, when the particle gets sufficiently close to this position it seems to briefly move periodically before diverging and staying at the same level. The significance of the position the particle ends up at is due to the Lennard-Jones potential. The equilibrium position of the Lennard-Jones potential can be calculated by setting the derivative of the potential equal to 0 and solving for the value of r which is found to be $r_0 = 2^{\frac{1}{6}} = 1.122$. This value is the distance that the particles in a Lennard-Jones potential will experience the minimum force between them [51]. Thus, the particle diverging to this value is logical. However, as discussed in section 5.2 the divergent trajectory is not expected from theoretical predictions.

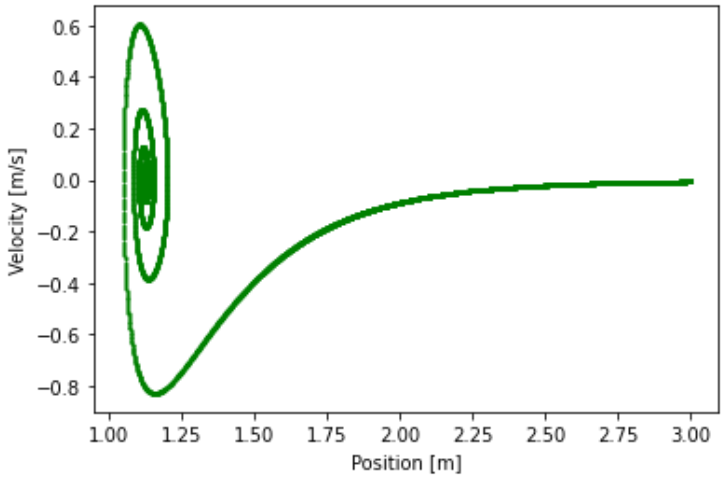


Figure 9 – Displays the phase diagram of the particle in the damped Lennard-Jones potential. From this figure it is evident that the velocity of the particle is equal to 0 at the start of the motion at x=3 and at the position the particle diverges to.

4.3 Position and Velocity Distribution Histograms

The position and velocity distribution histograms can be calculated using the positions and velocities calculated by the velocity Verlet algorithm. These positions and velocities can be normalised

resulting in the area covered by the bins being equal to 1. Each histogram was calculated using 40 bins with 100,000 steps done by the velocity Verlet algorithm. Then in order to get smoother histograms these parameters were increased.

Bin size and step number were increased for the velocity histogram which led to the distribution histogram getting smoother. 300 bins were plotted with 1,000,000 velocity Verlet steps taken lead to a sufficiently smooth distribution histogram.

Bin size and step number were increased for the position histogram. However, this did not result in the histogram getting smoother. Instead, the value between 1.10 and 1.15 blew up causing the other values to become incredibly small and difficult to see. Hence, the distribution histogram of the positions cannot be made smoother by increasing these parameters so the distribution is left with a step number of 100,000 with 40 bins.

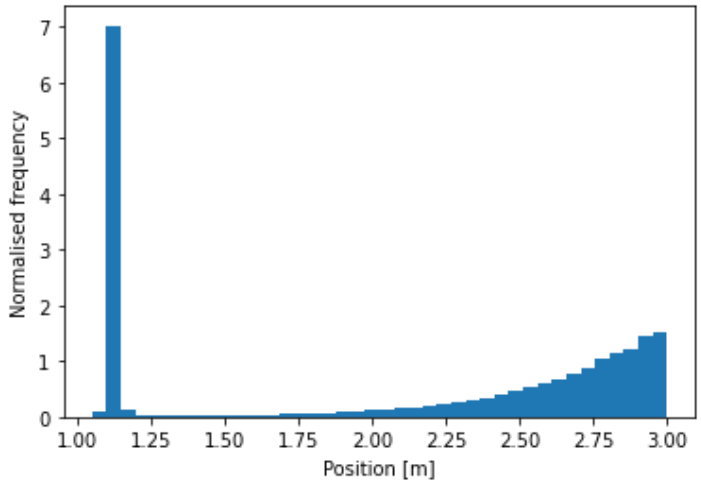


Figure 10 – Shows the normalised position distribution histogram with 40 bins plotted. There is a clear tendency for the position to be located in the bin that represents a position between 1.10 and 1.15. This is due to the equilibrium position of the Lennard-Jones potential outlined in section 4.2.

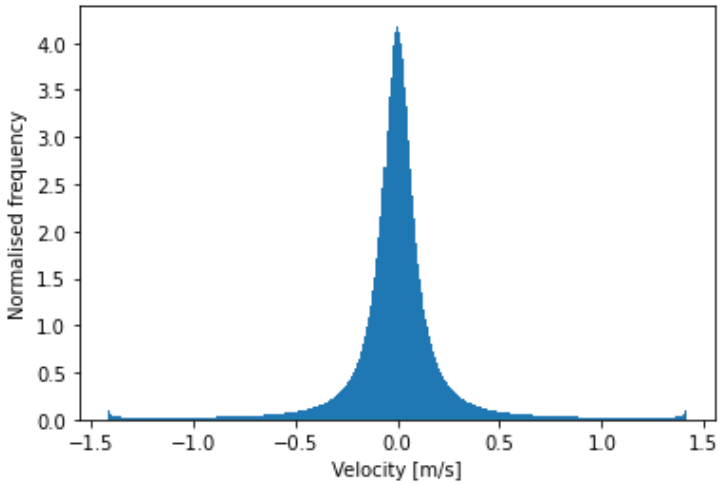


Figure 11 – Shows the normalized velocity distribution histogram with 300 bins. The number of steps ran by the algorithm is increased to 1,000,000 steps in order to make the histogram smooth. It is evident that this is similar shape to a probability curve with a mean at a value around 0.

4.4 Average Velocity and Temperature

Average Velocity:

Distribution functions are fitted to the velocity distribution histogram in order to calculate the average velocity. This will give us a more accurate calculation of the average velocity than using the mean of the velocity array as the distribution fitting will minimise the effect outliers have on the calculation. Hence, looking at the histogram in figure 11 we can hypothesis that this histogram would be well fitted by a symmetric distribution like a Gaussian distribution or a Lorentzian distribution. The Gaussian distribution has already been defined in equation 22. The Lorentzian distribution can be defined by:

$$L(x) = \frac{1}{\pi}\frac{a}{((x-x_0)^2 + a^2)} \qquad (eqn\ 23)\ [52]$$

where a is the amplitude of the distribution and x_0 is the mean value of the distribution.

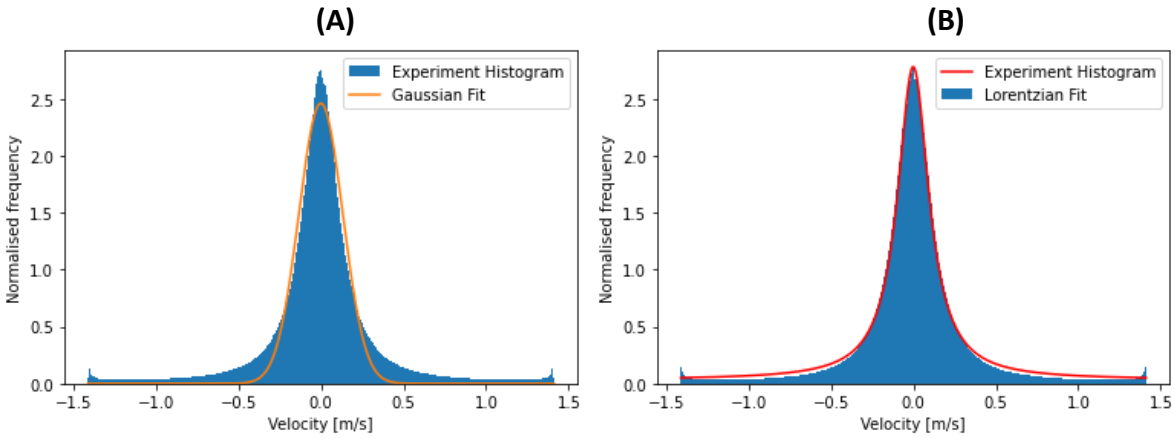


Figure 12 – Shows the two distribution curves fitted to the velocity distribution histograms. (A) Shows the histogram fitted with a Gaussian curve. (B) Shows the histogram fitted with a Lorentzian curve. Clearly, the Lorentzian curve (B) is a more suitable fit for this data so the average value parameter x_0 from the Lorentzian fit will be used to find the average value of the velocity.

Using our code, we can find the covariance matrix to obtain the uncertainty on our calculation of the average value of the velocity using the Lorentzian fit. The uncertainty on the average value of the velocity is the square root of the [1,1] element in the covariance matrix which is calculated to be 4.3 x10^{-5}m/s.

Hence, the average value of the velocity is calculated to be (-3.73 ± 0.43) x$10^{-4} m/s$.

Input Temperature:

Input temperature is calculated using the characteristic thermal energy which we presumed in order to simplify the Langevin equation. The equation $k_B T = 1$ can be solved to find T when the Boltzmann constant, k_B, is taken as 1.38064…x10^{-23}J/K [53]. Hence, the input temperature is calculated to be 7.24x10^{22}K to 3 significant figures.

Observed Temperature:

The equipartition theorem in equation 5 is rearranged to solve for temperature. This can be written as

$$T = \frac{m{v_{RMS}}^2}{k_B} \qquad (eqn\ 24)$$

Where v_{RMS} can be written as:

$$v_{RMS} = \sqrt{\frac{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}{n}} \qquad (eqn\ 25)\ [56]$$

Hence, using the velocities used to plot the velocity distribution histogram in figure 12 the root mean square speed can be found. It is found to be 1.014…m/s. This can be subbed into equation 24 to find the observed temperature assuming m=1. Hence, the observed temperature is calculated to be 7.45×10^{22}K.

Uncertainties:

From section 3.5 the global error on the velocity from the velocity Verlet algorithm is $O(\Delta t^2)$ which can be approximated to be 1×10^{-6} as the timestep is 0.001. Hence, each velocity used to calculate the root mean square speed has an uncertainty associated with it. This can be propagated through equation 25 to find the uncertainty on the value of v_{RMS} which will be used to calculate the uncertainty on the temperature. This leads to the uncertainty on the temperature to be 2.9×10^{11}K shown in section 8.1. This is a very small uncertainty when compared to the observed temperature calculated. This could be due to the fact that only the uncertainty from the velocity Verlet algorithm was included.

4.5 Potential of Mean Force

In order to plot the potential of mean force the position distribution histogram will have to be fitted with a curve. From visual inspection of figure 10 it is evident that a standard function like a linear, exponential or even Gaussian curve is not a suitable function to fit this data with. Therefore, we decide to interpolate the curve which constructs a curve through every data point. This makes the implicit assumption that the data points are accurate and distinct [57]. This is a valid assumption to make due to the relatively low global errors on the velocity Verlet algorithm discussed in section 3.5. However, this does not account for systematic errors incurred throughout the measurements.

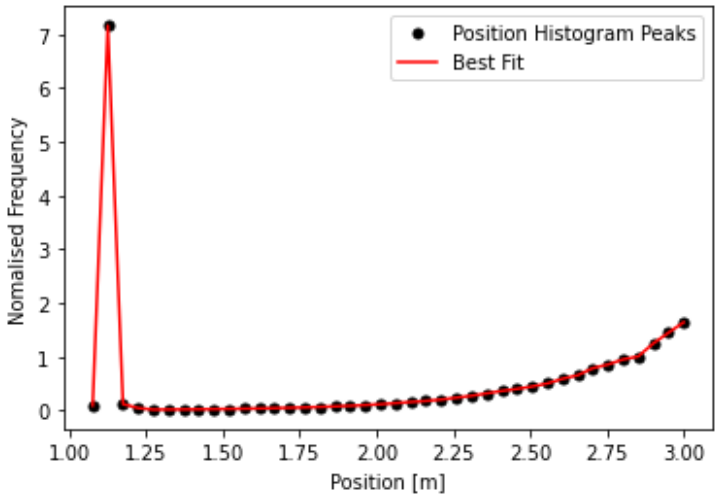


Figure 13 – Shows the interpolation fit with respect to the middle of each position histogram bin. The bin number of 40 is kept in order to show the accurate fit of the interpolation function to the data. The bin number will be increased when solving for the potential of mean force.

Using the interpolation data, a function for h(r) the position distribution can be found at specific values of r. This can be used in equation 6 along with the observed temperature found in section 4.4 to find a function for the potential of mean force, U_{PMF}. This can then be plotted and compared to the Lennard-Jones potential.

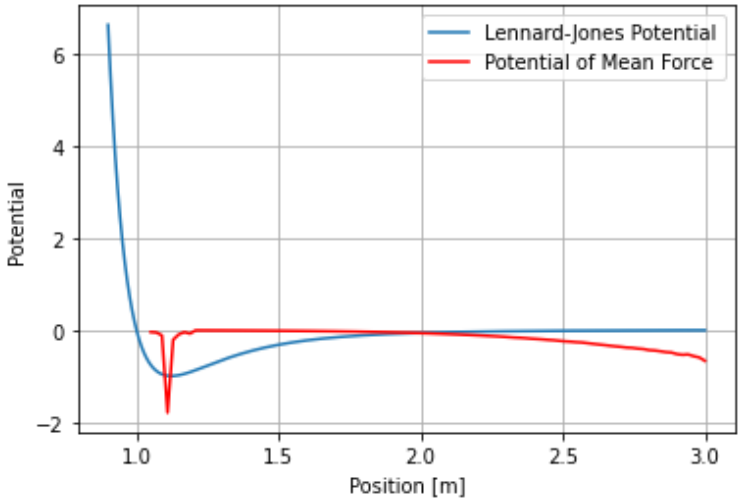


Figure 14 – Shows the comparison between the potential of mean force (red), U_{PMF}, and the Lennard-Jones potential (blue). They should not be exactly the same as the potential of mean force function accounts for solvent effects as well as the interaction potential between the particles. Whereas the Lennard-Jones potential only incorporates the intrinsic interaction between the two particles caused by itself.

4.6 Motion of Particle in an Undamped System

The undamped motion of Brownian particles can be studied in order to give us a better understanding of potential errors in our experiment. We observe a particle in the Lennard-Jones potential in an undamped system. This means that γ=0. The value for the damping coefficient determines the relative strength of the inertial forces with respect to the random external forces [29]. Hence, when it is equal to 0 it means that there is no damping nor random force acting on the particles in the Lennard-Jones potential. Hence, the only force that exits is the force applied by the external potential.

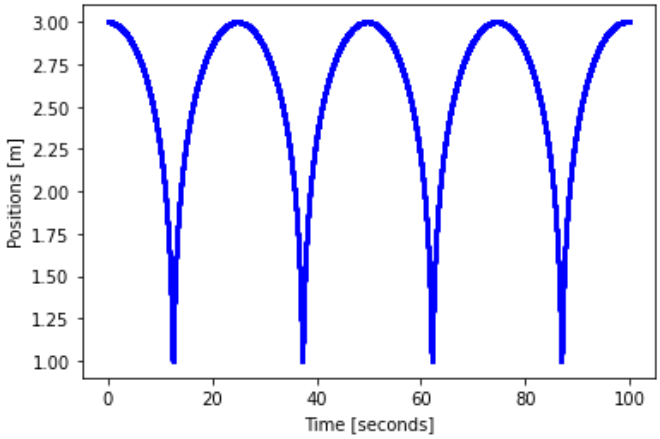


Figure 15 – Shows the particle's motion in the undamped system where the initial position is 3 and the number of steps the velocity Verlet algorithm takes is 100,000 with a timestep of 0.001 seconds. It is also assumed that the particle starts with an initial velocity of 0.

The periodic nature of the particle's trajectory is figure 15 is clear. The particle oscillates between the initial position x=3 and the lower bound position x=1. The significance of x=1 is clear from looking at the diagram of the Lennard-Jones potential. Below x=1 the Lennard-Jones potential is repulsive meaning that the particle is pushed away and so the distance begins to increase again. This is shown by figure 15.

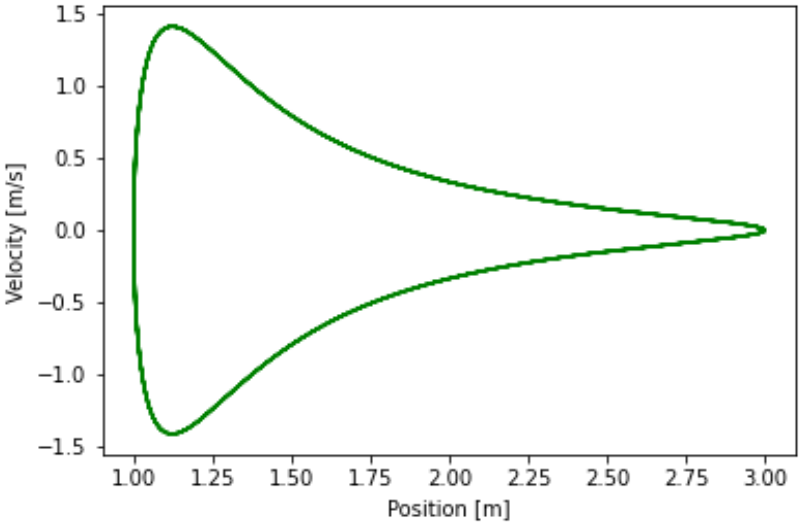


Figure 16 – Displays the closed phase diagram of the particle in the undamped Lennard-Jones potential. Using the same initial conditions and parameters as for figure 15. From this figure it is evident that the velocity of the particle is equal to 0 at the start of the motion at x=3 and at the position x=1.

Again, figure 16 can be physically interpreted to show the significance of the various parameters. The velocity only becomes 0 at two positions: when the particle is at x=3 the initial position and when the particle is at x=1 which is when the particle stops after being repulsed by the interactive potential.

Hence, during when the system is undamped the trajectory is periodic rather than divergent like we found for the damped system.

5 Discussion

5.1 Comparison of Results with Theory

In our experiment a stochastic process R(t) was modelled using the stochastic python package. A mean value of $(19 \pm 4)\ x10^{-5}$ was found by fitting a Gaussian curve to the histogram distribution of this function. This is a very small value which can be approximated to 0 which means we can verify that the stochastic function that we fitted was a stationary Gaussian process with zero-mean. This signifies that this function can be used in the Langevin equation (equation 1).

We observed that the motion of a particle in a damped Lennard-Jones potential was divergent. The particle diverged to a position after around 70 seconds. We found the average position of the particle between 70 and 100 seconds was 1.122… which agrees with the equilibrium position of the Lennard-Jones potential to 5 significant figures [51]. This makes some physical sense as after a certain amount of time the particle ends up in the equilibrium position of the interactive potential. However, generally the trajectory of a Brownian particle should be periodic so that energy is conserved [58] [59]. Some periodic motion was observed in our graph however shortly after this

periodic motion the particle diverged to the equilibrium position. The phase diagram of the undamped particle (figure 9) clearly shows the particle's trajectory is divergent meaning the periodic motion we observe is not representative of the overall motion.

The distribution function of the position of the particle and the velocity of the particle were also found. The position distribution histogram had a large value between a position of 1.10 and 1.15. This was due to the particle's motion diverging to the equilibrium position between these position values. Hence, our position distribution function g(r) had a peak around the equilibrium position. This property is matched in the analytical solution for the radial distribution function for the Lennard-Jones potential at both high and low temperatures [60]. A large peak around the equilibrium position of the Lennard-Jones potential is also observed in the radial distribution function in the Lennard-Jones fluid using a general equation with limiting conditions of zero density and infinite distance imposed by statistical thermodynamics [61]. However, both these theoretical calculated radial functions display a periodic nature directly after the peak [60] [61]. This is something that is not evident in the radial distribution function found in our experiment. This is a direct result of the lack of periodic motion we found in the trajectory of the particle. The distribution of the velocity histogram was fitted with a Lorentzian function with a mean of around 0. This matches the histograms found by studying the Lennard-Jones potential with the Berendsen thermostat [62].

The average velocity of the particle under the influence of the Lennard-Jones potential was calculated to be (-3.73 ± 0.43) x$10^{-4} m/s$. This agrees with the study of Argon particles in the Lennard-Jones potential in which the average velocity becomes 0m/s at a critical temperature [63]. The value for average velocity we calculated is very close to 0 so it agrees with theoretical predictions.

The observed temperature in the system was found to be 7.45x10^{22}K. The uncertainty on this value was found to be very small as only the global error on the velocity Verlet algorithm was taken into account. It did not include any potential uncertainty in the velocity distribution histogram. This was of the same order as the input temperature which was calculated to be 7.24x10^{22}K using the characteristic thermal energy. Hence, the input temperature and the observed temperature can be approximated to be around the same value. This justifies the assumption made at the start of the experiment in section 3.2 that the energy unit is equal to the characteristic thermal energy.

The potential of mean force of the system was also found and compared to the Lennard-Jones potential. The potential of mean force considers solvent effects as well as the intrinsic potential which means that it will not exactly match the Lennard-Jones potential curve. However, the overall shape should be similar due to the Lennard-Jones potential being the main contributor to the force experience. Our potential of mean force matches the position of the minimum potential around 1.12m which is a direct result of our observed trajectory diverging to this value. However, our potential of mean force does not match the potential of mean force found by observing a system of 64 Lennard-Jones particles evolving via both inertial dynamics and overdamped dynamics [64]. A similar radial distribution function can be seen by studying sodium and chloride ions in supercritical water however the potential of mean force is very different to our calculation [65]. This perhaps means that our assumption that the position histogram points were accurate was invalid which then means that the interpolation method used is not accurate.

The simplified case of the system being undamped was studied in section 4.6 in order to try to verify that the trajectory found in section 4.2 should be periodic. For the undamped case a periodic trajectory was found. A closed loop was observed in the phase diagram. This adheres to the law of

conservation of energy [66]. Both the position trajectory (figure 15) and the phase diagram (figure 16) distinctly match predictions calculated using recurrent neutral networks with large timesteps [67]. This verifies our velocity Verlet algorithm when the force is only dependent on the interaction potential. This gives us a better understanding of potential errors in calculating the trajectory of the particle in the damped case.

5.2 Reasoning Behind Discrepancies with Theoretical Predictions

The main discrepancy of the results found during our experiment and theoretical predictions was that we found the particle's trajectory to be divergent. Theoretical predictions indicate that this should be periodic. This seeming error then lead to an inaccurate position histogram which in turn lead to the potential of mean force being unlike other calculations of the mean force.

The interpolation method used to calculate the potential of mean force is valid when the data points from the position distribution function are accurate and distinct. However, due to the divergent nature of our particle's trajectory we hypothesised that the position distribution function is not as accurate as required. Nonetheless, had the values from the position distribution been more accurate this method would be valid in order to find the potential of mean force.

The undamped case is verified by theoretical predictions. This highlights that our velocity Verlet algorithm is correct when the force is only dependent on the interactive Lennard-Jones potential. Therefore, the mistake that led to an incorrect divergent trajectory is due to a systematic error in the way the damping and random forces were defined.

6 Conclusion

This experiment studied the one-dimensional motion of a particle in the Lennard-Jones potential using the velocity Verlet algorithm. Through the use of Langevin dynamics and the Lennard-Jones potential the contribution of external solvent forces to the acceleration of the particle is calculated.

In summary, the trajectory of the particle was found to be divergent which does not match theoretical predictions. These theoretical predictions state that the trajectory should be periodic in order to satisfy the conservation of energy [58] [59]. By studying the motion of the undamped case, it is hypothesised that a systematic error has been made related to the damping and random force defined within the velocity Verlet algorithm. This could be improved with an in-depth study of the effect these forces have on the overall trajectory of the particle using different initial conditions. It could also be improved by comparing results with different algorithms like a velocity Verlet half-step method.

The average velocity of the particle under the influence of the Lennard-Jones potential was found to be (-3.73 ± 0.43) x$10^{-4} m/s$ which agreed with theoretical predictions [63]. The observed temperature in the system was found to be 7.45x10^{22}K which agreed with the input temperature calculated using the characteristic thermal energy.

The potential of mean force of the system was also found and compared to the Lennard-Jones potential. It was found that the potential of mean force calculated was different to other experimental values found using different methods [64] [65]. The cause of the error on this function is due to the potential of mean force being directly dependent on the position distribution function which is derived from the trajectory of the particle.

These results seem to naturally lead to a need for further research into the algorithms that can be used to solve Langevin's equations of motion. The limitations of each algorithm could be studied.

7 References

[1] A. Satoh (2003) “Brownian Dynamics Methods,” in *Introduction to Molecular-Microsimulation of Colloidal Dispersions*. 1st edn, pp. 127–135. Available at: https://reader.elsevier.com/reader/sd/pii/S1383730303800376?token=70B39C36E07DC454DA20CE82903754A84FF49A286E600B6A059B813678434FB61CC09506324385268C19745FD2512D62&originRegion=eu-west-1&originCreation=20221030123203.

[2] D. Lemons, and A. Gythiel (1997) “Paul Langevin’s 1908 paper ‘on the theory of Brownian motion’ [‘sur la théorie du mouvement brownien,’ C. R. Acad. sci. (Paris) 146, 530–533 (1908)],” *American Journal of Physics*, 65(11), pp. 1079–1081. Available at: https://doi.org/https://www.physik.uni-augsburg.de/theo1/hanggi/History/Langevin1908.pdf.

[3] M. Yuewen (2013) “Introduction to Langevin dynamics and its use in VASP,” *ScienceNet*. Science Network . Available at: https://wap.sciencenet.cn/blog-588243-746675.html?mobile=1 (Accessed: October 30, 2022).

[4] L. Stella, C. Lorenz and L. Kantorovich (2014) “Generalized langevin equation: An efficient approach to nonequilibrium molecular dynamics of open systems,” *Physical Review B*, 89(13), pp. 1–5. Available at: https://doi.org/10.1103/physrevb.89.134303.

[5] R. Brown (1828) “XXVII. *a brief account of microscopical observations made in the months of June, July and August* 1827, *on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies*,” *The Philosophical Magazine*, 4(21), pp. 161–173. Available at: https://doi.org/https://www.tandfonline.com/doi/abs/10.1080/14786442808674769.

[6] A. Einstein (1905) “Investigations on the Theory of the Brownian Movement .” Translated by A. Cowper, pp. 14–23. Available at: https://scholar.google.com/scholar?q=einstein+brownian+motion&hl=en&as_sdt=0%2C5&as_ylo=1905&as_yhi=1905.

[7] M. Smoluchowski (1906) “On the Kinetic Theory of the Brownian Molecular Motion and of Suspensions,” *Annalen der Physik*. Translated by R. Schmitz and R. Jones, 21, pp. 756–780. Available at: https://doi.org/https://cf2-taniaksiazka.statiki.pl/images/files/F39/@DL_E06F56DCEB_DL-EBWM,71595.pdf.

[8] A. Chodos (2005) “Einstein and Brownian Motion,” *American Physical Society - This Month in Physics History*, 14(2). Available at: https://doi.org/https://www.aps.org/publications/apsnews/200502/history.cfm#:~:text=In%201827%2C%20the%20English%20botanist,and%20cause%20them%20to%20jiggle.

[9] The Editors of Encyclopedia Britanica (2022) *Jean Perrin Biography*, *Encyclopædia Britannica*. Britannica. Available at: https://www.britannica.com/biography/Jean-Perrin (Accessed: October 30, 2022).

[10] E. Nelson (2001) “Dynamical Theories of Brownian Motion,” *Department of Mathematics Princeton University*, 2, pp. 13–17. Available at: https://doi.org/https://web.math.princeton.edu/~nelson/books/bmotion.pdf.

[11] W. Coffey and Y. Kalmykov (2017) “Historical background and introductory concepts,” *The Langevin Equation*, 4, pp. 11–15. Available at: https://doi.org/https://www.worldscientific.com/doi/pdf/10.1142/9789813222007_0001.

[12] Y. Pomeau and J. Piasecki (2017) “The Langevin equation,” *Comptes Rendus Physique*, 18(9-10), pp. 570–582. Available at: https://doi.org/https://www.researchgate.net/publication/320491913_The_Langevin_equation.

[13] A. Tokmakoff (2021) *Computing Dynamics*, *Chemistry LibreTexts*. University of Chicago. Available at:https://chem.libretexts.org/Bookshelves/Biological_Chemistry/Concepts_in_Biophysical_Chemistry_(Tokmakoff)/06%3A_Dynamics_and_Kinetics/22%3A_Biophysical_Reaction_Dynamics/22.02%3A_Computing_Dynamics (Accessed: October 30, 2022).

[14] Y. Demirel and V. Gerbaud (2019) “Probabilistic approach in thermodynamics,” *Nonequilibrium Thermodynamics*, pp. 721–724. Available at: https://doi.org/https://www.sciencedirect.com/science/article/pii/B9780444641120000150.

[15] J. Lennard-Jones (1924) “On the determination of molecular fields.—i. from the variation of the viscosity of a gas with temperature,” *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 106(738), pp. 441–462. Available at: https://doi.org/https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1924.0081.

[16] J. Lennard-Jones (1931) “Cohesion,” *Proceedings of the Physical Society*, 43(5), pp. 461–482. Available at: https://doi.org/10.1088/0959-5309/43/5/301.

[17] R. Naeem (2022) *Lennard-Jones potential*, *Chemistry LibreTexts*. UC Davis. Available at: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Physical_Properties_of_Matter/Atomic_and_Molecular_Properties/Intermolecular_Forces/Specific_Interactions/Lennard-Jones_Potential (Accessed: October 30, 2022).

[18] X. Wang, S. Ramírez-Hinestrosa et al. (2020) “The Lennard-Jones Potential: When (not) to use it,” *Physical Chemistry Chemical Physics*, 22(19), pp. 10624–10633. Available at: https://doi.org/https://pubs.rsc.org/en/content/articlepdf/2020/cp/c9cp05445f.

[19] E. Generalic (2022) *Chemistry glossary*, *Lennard-Jones+potential @ Chemistry Dictionary & Glossary*. KTF-Split. Available at: https://glossary.periodni.com/glossary.php?en=Lennard-Jones%2Bpotential (Accessed: October 30, 2022).

[20] R. Nave (2016) *Equipartition of Energy*, *Equipartition of energy*. Georgia State University. Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/Kinetic/eqpar.html (Accessed: November 4, 2022).

[21] A. Medved, R. Davis and P. Vasquez (2020) “Understanding fluid dynamics from Langevin and Fokker–Planck equations,” *Fluids*, 5(1), pp. 40–45. Available at: https://doi.org/https://www.mdpi.com/2311-5521/5/1/40.

[22] D. Trzesniak, AP. Kunz et al. (2007) “A comparison of methods to compute the potential of mean force,” *ChemPhysChem*, 8(1), pp. 162–169. Available at: https://doi.org/https://chemistry-europe.onlinelibrary.wiley.com/doi/epdf/10.1002/cphc.200600527?saml_referrer.

[23] A. Leach (2001) “Empirical Force Field Models: Molecular Mechanics,” in *Molecular modelling: Principles and applications*. Harlow, England: Prentice Hall, pp. 165–212. Available at: https://www.scribd.com/document/449103078/Andrew-R-Leach-Molecular-modelling-principles-and-applications-Prentice-Hall-2001-pdf

[24] J. Li (2007) *The Potential of Mean Force*, *The potential of mean force*. University of Berkeley. Available at: https://w3.iams.sinica.edu.tw/lab/jlli/thesis_andy/node15.html (Accessed: October 30, 2022).

[25] N. Grønbech-Jensen and O. Farago (2013) “A simple and effective verlet-type algorithm for simulating Langevin Dynamics,” *Molecular Physics*, 111(8), pp. 983–991. Available at: https://doi.org/https://www.tandfonline.com/doi/full/10.1080/00268976.2012.760055.

[26] Q. Spreiter and M. Walter (1999) “Classical molecular dynamics simulation with the velocity Verlet algorithm at strong external magnetic fields,” *Journal of Computational Physics*, 152(1), pp. 102–119. Available at: https://doi.org/https://www.sciencedirect.com/science/article/pii/S002199919996237X.

[27] R. Skeel and C. Hartmann (2021) “Choice of damping coefficient in Langevin Dynamics,” *The European Physical Journal B*, 94(9), pp. 178–190. Available at: https://link.springer.com/content/pdf/10.1140/epjb/s10051-021-00182-z.pdf.

[28] C. Flynn (2018) *Noise processes*, *Noise Processes - stochastic 0.7.0 documentation*. MIT. Available at: https://stochastic.readthedocs.io/en/stable/noise.html#stochastic.processes.noise.GaussianNoise (Accessed: November 5, 2022).

[29] T. Schlick (2010) "14.4 Langevin Dynamics," in *Molecular modeling and simulation: An interdisciplinary guide*. 2nd edn. New York: Springer (Interdisciplinary Applied Mathematics), pp. 479–484. Available at: https://link.springer.com/content/pdf/10.1007/978-1-4419-6351-2.pdf

[30] S. Blinder (2022) *Chapter 5: Harmonic Oscillator*, *Chemistry LibreTexts*. University of Michigan. Available at: https://chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Quantum_Mechanics/06._One_Dimensional_Harmonic_Oscillator/Chapter_5%3A_Harmonic_Oscillator (Accessed: November 5, 2022).

[31] S. Ling, J. Sanny and W. Moebs (2016) "15.1 Simple Harmonic Motion," in *University physics. volume 2*. Houston , Texas: OpenStax College Rice University, pp. 724–733. Available at: https://assets.openstax.org/oscms-prodcms/media/documents/UniversityPhysicsVol1-WEB.pdf

[32] H. Martin (2020) *13.1: The Motion of a spring-mass system*, *Physics LibreTexts*. University of Wisconsin-Madison. Available at: https://phys.libretexts.org/Bookshelves/University_Physics/Book%3A_Introductory_Physics_-_Building_Models_to_Describe_Our_World_(Martin_Neary_Rinaldo_and_Woodman)/13%3A_Simple_Harmonic_Motion/13.01%3A_The_motion_of_a_spring-mass_system (Accessed: November 5, 2022).

[33] B. Li (2014) "Error Propagation of Verlet Algorithm," *Civil and Environmental Engineering*, pp. 1–4. Available at: https://www.researchgate.net/publication/265908915_Error_Propagation_of_Verlet_Algorithm.

[34] J. Feldman, A Rechnitzer and E. Yeager (2021) "Local Truncation Error for Euler's Method," in *CLP-2 Integral Calculus*. Vancouver: UBC, pp. 599–603. Available at: https://personal.math.ubc.ca/~CLP/CLP2/clp_2_ic/ap_local_trunc_Euler.html

[35] H. Jónsson (2000) "Classical dynamics," *Lecture notes University of Washington*, 1, pp. 16–24. Available at: https://doi.org/http://www.physics.drexel.edu/~valliere/PHYS305/Diff_Eq_Integrators/Verlet_Methods/Diffrntleqn3.pdf.

[36] D. Frenkel and B. Smit (2002) "Molecular dynamics simulations," *Understanding Molecular Simulation*, 2, pp. 63–107. Available at: https://doi.org/https://www.sciencedirect.com/science/article/pii/B9780122673511500067.

[37] L. Shampine (1986) "Global error estimation with one-step methods," *Computers & Mathematics with Applications*, 12(7), pp. 885–894. Available at: https://doi.org/https://www.sciencedirect.com/science/article/pii/0898122186900325.

[38] W. Swope, H. Andersen et al. (1982) "A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters," *The Journal of Chemical Physics*, 76(1), pp. 637–649. Available at: https://doi.org/https://aip.scitation.org/doi/10.1063/1.442716.

[39] C. Holm (2012) "Molecular Dynamics integrators," *Simulation Methods in Physics 1*, pp. 12–16. Available at: https://doi.org/https://www2.icp.uni-stuttgart.de/~icp/mediawiki/images/5/54/Skript_sim_methods_I.pdf.

[40] University of Helsinki (2020) "Solving the equations of motion," *Lecture 3* , pp. 56–67. Available at: https://doi.org/http://www.courses.physics.helsinki.fi/fys/moldyn/lectures/L3.pdf.

[41] M. Bader (2013) "Molecular Dynamics," *Scientific Computing II*, pp. 1–5. Available at: https://doi.org/https://www5.in.tum.de/lehre/vorlesungen/sci_compII/ss13/uebungen/blatt9solution.pdf

[42] E. Hairer (2010) "Basic symplectic integration schemes," *Lecture 2: Symplectic integrators*, pp. 2–4. Available at: https://doi.org/https://www.unige.ch/~hairer/poly_geoint/week2.pdf.

[43] A. Chambliss and J. Franklin (2020) “A magnetic velocity verlet method,” *American Journal of Physics*, 88(12), pp. 1075–1082. Available at: https://doi.org/https://arxiv.org/pdf/2008.11810.pdf.

[44] R. Zhang (2016) *Comparison of velocity Verlet and leapfrog algorithms*, *Computational Science Stack Exchange*. Available at: https://scicomp.stackexchange.com/questions/24232/comparison-of-velocity-verlet-and-leapfrog-algorithms (Accessed: November 6, 2022).

[45] L. Tiburzi (2012) *Runge-Kutta versus Velocity-Verlet Solutions for the Classical Harmonic Oscillator*, *Wolfram Demonstrations Project*. Available at: https://demonstrations.wolfram.com/RungeKuttaVersusVelocityVerletSolutionsForTheClassicalHarmon/ (Accessed: November 6, 2022).

[46] B. Dreyfus and J. Redish (2011) *The Lennard-Jones potential*, *The Lennard-Jones Potential - Nexus Wiki*. Available at: https://www.compadre.org/nexusph/course/The_Lennard-Jones_Potential (Accessed: November 8, 2022).

[47] J. Kadupitiya, G. Fox and V. Jadhao (2022) “Solving Newton’s equations of motion with large timesteps using recurrent neural networks based operators,” *Machine Learning: Science and Technology*, 3(2), p. 025002. Available at: https://doi.org/https://iopscience.iop.org/article/10.1088/2632-2153/ac5f60/pdf.

[48] W. Syam (2022) *Tutorial: Python for fitting gaussian distribution on data*, *Wasy Research*. Applied Programming Wasy. Available at: https://www.wasyresearch.com/tutorial-python-for-fitting-gaussian-distribution-on-data/ (Accessed: November 9, 2022).

[49] R. Nave (2016) *Gaussian distribution function*, *Hyperhysics Gaussian Distribution*. Georgia State University. Available at: http://hyperphysics.phy-astr.gsu.edu/hbase/Math/gaufcn.html (Accessed: November 9, 2022).

[50] T. Thiery, P. Le Doussal and K. Wiese (2015) “Spatial shape of avalanches in the Brownian force model,” *Journal of Statistical Mechanics: Theory and Experiment*, 2015(8). Available at: https://doi.org/https://iopscience.iop.org/article/10.1088/1742-5468/2015/08/P08019/meta.

[51] N. Yu and A. Polycarpou (2004) “Adhesive contact based on the Lennard–Jones Potential: A correction to the value of the equilibrium distance as used in the potential,” *Journal of Colloid and Interface Science*, 278(2), pp. 428–435. Available at: https://doi.org/https://www.sciencedirect.com/science/article/pii/S0021979704005454?via%3Dihub.

[52] E. Weisstein (2002) *Lorentzian function*, *from Wolfram MathWorld*. Available at: https://mathworld.wolfram.com/LorentzianFunction.html (Accessed: November 11, 2022).

[53] The Editors of Encyclopædia Britannica (2022) *Boltzmann constant*, *Britannica*. Encyclopædia Britannica, inc. Available at: https://www.britannica.com/science/Boltzmann-constant (Accessed: November 11, 2022).

[54] A. Mangs (2022) *Fit a curve to a histogram in Python*, *DevPress*. Available at: https://devpress.csdn.net/python/630463e97e6682346619af03.html (Accessed: November 11, 2022).

[55] B. Prudholm (2014) *Fit points to a Lorentzian curve and find center and half maximum bandwidth in Python.* [online] Stack Overflow. Available at: <https://stackoverflow.com/questions/24437070/fit-points-to-a-lorentzian-curve-and-find-center-and-half-maximum-bandwidth-in-p> [Accessed 3 October 2022].

[56] P. Flowers (2022) *9.5 the Kinetic-molecular theory*, *IU PressBooks*. Rice University . Available at: https://iu.pressbooks.pub/openstaxchemistry/chapter/9-5-the-kinetic-molecular-theory/ (Accessed: November 12, 2022).

[57] J. Kiusalaas (2005) *Numerical methods in engineering with python*, *Chapter 3: Interpolation and Curve Fitting | Engineering360*. Available at: https://www.globalspec.com/reference/72167/203279/chapter-3-interpolation-and-curve-fitting#:~:text=There%20is%20a%20distinction%20between,usually%20due%20to%20measurement%20errors. (Accessed: November 12, 2022).

[58] A. Khurshudyan (2018) “Exact and approximate controllability conditions for the micro-swimmers deflection governed by electric field on a plane: The Green’s function approach,” *Archives of Control Sciences*, 28(3), pp. 335–347. Available at: https://doi.org/https://www.researchgate.net/publication/328306265_Exact_and_approximate_controllability_conditions_for_the_micro-swimmers_deflection_governed_by_electric_field_on_a_plane_The_Green's_function_approach.

[59] L. Borland (1998) “Microscopic dynamics of the nonlinear Fokker-Planck equation: A phenomenological model,” *Physical Review E*, 57(6), pp. 6634–6642. Available at: https://doi.org/10.1103/physreve.57.6634.

[60] M. Khanpour, G. Parsafar and B. Najafi (2004) “Analytic solution to integral equations of liquid state theories for potentials with a hard core at low densities,” *Journal of the Physical Society of Japan*, 73(5), pp. 1197–1204. Available at: https://doi.org/10.1143/jpsj.73.1197.

[61] A. Morsali, E. Goharshadi et al. (2005) “An accurate expression for radial distribution function of the Lennard-Jones fluid,” *Chemical Physics*, 310(1-3), pp. 11–15. Available at: https://doi.org/10.1016/j.chemphys.2004.09.027.

[62] D. Kleinerman, C. Czaplewski, et al. (2008) “Implementations of nosé–hoover and Nosé–Poincaré thermostats in mesoscopic dynamic simulations with the United-residue model of a polypeptide chain,” *The Journal of Chemical Physics*, 128(24), pp. 245103–1-245103–16. Available at: https://doi.org/10.1063/1.2943146.

[63] H. Tabe *et al.* (2021) “Molecular dynamics study on characteristics of reflection and condensation molecules at vapor–liquid equilibrium state,” *PLOS ONE*, 16(3), pp. 1–19. Available at: https://doi.org/10.1371/journal.pone.0248660.

[64] I. Jenkins, J. Crocker and T. Sinno (2015) “Interaction potentials from arbitrary multi-particle trajectory data,” *Soft Matter*, 11(35), pp. 6948–6956. Available at: https://doi.org/10.1039/c5sm01233c.

[65] P. Cummings and A. Chialvo (1996) “Molecular simulation of Supercritical Water and Aqueous Solutions,” *Journal of Physics: Condensed Matter*, 8(47), pp. 9281–9287. Available at: https://doi.org/10.1088/0953-8984/8/47/016.

[66] J. Brennan (2017) *How Is Energy Conserved Within a Closed System?*, *Sciencing*. Available at: https://sciencing.com/mechanical-energy-kids-8520550.html (Accessed: November 13, 2022).

[67] J. Kadupitiya, G. Fox and V. Jadhao (2022) “Solving Newton’s equations of motion with large timesteps using recurrent neural networks based operators,” *Machine Learning: Science and Technology*, 3(2), p. 025002. Available at: https://doi.org/10.1088/2632-2153/ac5f60.

8 Appendix

8.1 Derivation of the Uncertainty on Temperature

By equation 25,

$$v_{RMS} = \sqrt{\frac{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}{n}}$$

therefore, using propagation of error formula with v_{RMS} as the general function we find:

$$\Delta v_{RMS} = \sqrt{\frac{1}{n}\left(\left(\frac{\partial v_{RMS}}{\partial v_1}\Delta v_1\right)^2 + \left(\frac{\partial v_{RMS}}{\partial v_2}\Delta v_2\right)^2 + \ldots + \left(\frac{\partial v_{RMS}}{\partial v_n}\Delta v_n\right)^2\right)}$$

$$\Delta v_1 = \Delta v_2 = \ldots = \Delta v_n = 1\text{x}10^{-6}$$

$$\Delta v_{RMS} = \sqrt{\frac{1\text{x}10^{-6}}{n}\left(\left(\frac{\partial v_{RMS}}{\partial v_1}\right)^2 + \left(\frac{\partial v_{RMS}}{\partial v_2}\right)^2 + \ldots + \left(\frac{\partial v_{RMS}}{\partial v_n}\right)^2\right)}$$

Calculating the partial derivatives, we find:

$$\frac{\partial v_{RMS}}{\partial v_1} = \frac{v_1}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}}$$

Other partial derivatives follow a similar style hence,

$$\Delta v_{RMS} = \sqrt{\frac{1\text{x}10^{-6}}{n}\left(\left(\frac{v_1}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}}\right)^2 + \left(\frac{v_2}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}}\right)^2 + \ldots + \left(\frac{v_n}{\sqrt{{v_1}^2 + {v_2}^2 + \ldots + {v_n}^2}}\right)^2\right)}$$

Which can be simplified to become:

$$\Delta v_{RMS} = \sqrt{\frac{1\text{x}10^{-6}}{n\,({v_1}^2 + {v_2}^2 + \ldots + {v_n}^2)}\,\left((v_1)^2 + (v_2)^2 + \ldots + (v_n)^2\right)}$$

The factor of $\left((v_1)^2 + (v_2)^2 + \ldots + (v_n)^2\right)$ cancels to leave:

$$\Delta v_{RMS} = \sqrt{\frac{1\text{x}10^{-6}}{n}}$$

Where n is the number of steps ran by the velocity Verlet algorithm which is 1,000,000 as we increased the velocity Verlet steps to smooth the velocity distribution histogram. This gives a value of $\Delta v_{RMS} = 1\text{x}10^{-6}$. This can then be propagated through equation 24 to find the uncertainty on the observed temperature:

$$\left(\frac{\Delta T}{T}\right)^2 = \left(2\frac{\Delta v_{RMS}}{v_{RMS}}\right)^2$$

This can be solved for ΔT to be 2.9x10^{11}K.

8.2 Experimental Code

In [1]:
```
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
import numpy.random as rand
import stochastic.processes.noise as sto

from scipy import interpolate
```

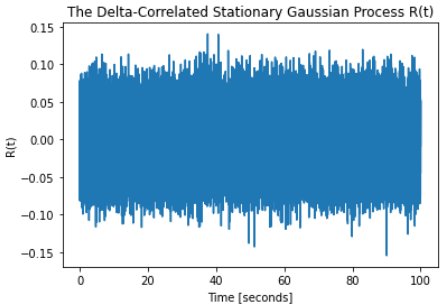
In [2]:
```
plt.rcParams['figure.figsize'] = (6,4)
plt.rcParams['font.size'] = 12
plt.rcParams['savefig.bbox'] = 'tight'
```

In [3]:
```
def mean(x):
    return sum(x)/len(x)
# Function that gives the average value of an array
```

Calculating the values of the stochastic function R(t)

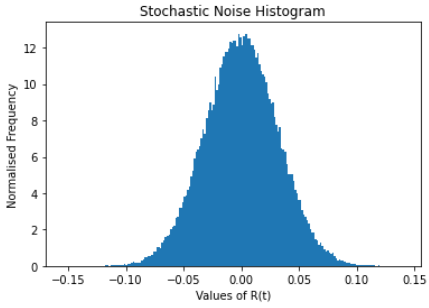
In [4]:
```
import stochastic.processes.noise as sto

G = sto.GaussianNoise(1)
times = np.arange(0,100 ,100/100002)
# Creates an array of the 100,000 steps with timestep 0.001

R_t = G.sample_at(times)
# Generates the corresponding value of the stochastic function of each time

plt.plot(times[1:], R_t)
# Plots the Noise Function

plt.xlabel("Time [seconds]")
plt.ylabel("R(t)")
plt.title("The Delta-Correlated Stationary Gaussian Process R(t)")

# reference - pyhton stochastic package
# [28] C. Flynn (2018) Noise processes, Noise Processes - stochastic 0.7.0 documentation.
# MIT. Available at:
# https://stochastic.readthedocs.io/en/stable/noise.html#stochastic.processes.noise.GaussianNoise (Accessed: November 5, 2022).
```

Out[4]:
```
Text(0.5, 1.0, 'The Delta-Correlated Stationary Gaussian Process R(t)')
```



In [5]:
```
amplitudes_Rt, amplitudesbins_Rt, npatches_Rt = plt.hist(R_t, bins=200, density=True)
# Plots the normalised histogram of the stochastic function R(t) with 200 bins

plt.xlabel("Values of R(t)")
plt.ylabel("Normalised Frequency")
plt.title("Stochastic Noise Histogram")
```

Out[5]:
```
Text(0.5, 1.0, 'Stochastic Noise Histogram')
```

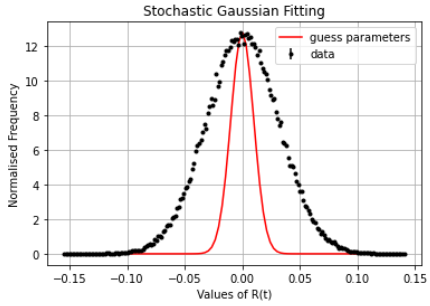


A Gaussian Curve can be fitted to this histogram in order to show the Gaussian distribution. This also will give us an accurate value of the mean of the stochastic function R(t).

In [6]:
```
range_avg_Rt = np.linspace(min(R_t), max(R_t), 200)
# Gives the average value of R(t) within each bin to help us fit a Gaussian Function to the data
```

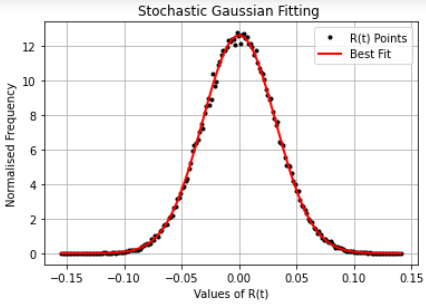
In [7]:
```
def gaus(x,a,x0,sigma):
    return a*np.exp(-(x-x0)**2/(2*sigma**2))
# Defines the gaussian function with parameterters a, x0 and sigma
```

In [8]:
```
# plot the raw data
plt.errorbar(range_avg_Rt, amplitudes_Rt, yerr=0, fmt=".", color = 'black', label = 'data')
plt.grid(True)
plt.xlabel('Values of R(t)')
plt.ylabel('Normalised Frequency')


# initial guess
pars0 = (max(amplitudes_Rt), 0,0.01)

# overlay gaussian
x = np.linspace(min(range_avg_Rt), max(range_avg_Rt), 100)

plt.plot(x, gaus(x,*pars0), color = 'red', label = 'guess parameters')
plt.legend();
plt.title("Stochastic Gaussian Fitting");
```



In [9]:
```
# perform fitting
popt, pcov = curve_fit(gaus, range_avg_Rt, amplitudes_Rt,  absolute_sigma=True, p0=pars0)

print("The covariance matrix is: \n", pcov, "\n")

# extract best-fit parameter and its error
a_opt = popt[0]
x0_opt = popt[1]
sigma_opt = popt[2]
a_opt_err = np.sqrt(pcov[0, 0])
x0_opt_err = np.sqrt(pcov[1, 1])
sigma_opt_err = np.sqrt(pcov[2, 2])

print(f" a vlaue (best-fit) = {a_opt:.2g} ± {a_opt_err:.2g}")
print(f" x0 value (best-fit) = {x0_opt:.2g} ± {x0_opt_err:.2g}")
print(f" sigma value (best-fit) = {sigma_opt:.2g} ± {sigma_opt_err:.2g}")

# plot data
plt.plot(range_avg_Rt, amplitudes_Rt, 'ko', markersize=3, label = 'R(t) Points')

# creating x interval to include in y fit
x = np.linspace(min(range_avg_Rt), max(range_avg_Rt), 100)
y_fit = gaus(x, *popt)
plt.plot(x, y_fit, color="red", label="Best Fit", linewidth=2)


plt.grid(True)
plt.xlabel("Values of R(t)")
plt.ylabel("Normalised Frequency")
plt.title("Stochastic Gaussian Fitting ")
plt.legend();
```

```
The covariance matrix is:
 [[ 3.95633619e-02 -8.47415511e-11 -6.62751070e-05]
 [-8.47415511e-11  3.33065194e-07  1.41511701e-13]
 [-6.62751070e-05  1.41511701e-13  3.33064934e-07]]

 a vlaue (best-fit) = 13 ± 0.2
 x0 value (best-fit) = 4.6e-05 ± 7.7e-05
 sigma value (best-fit) = 0.032 ± 0.00058
```



Velocity Verlet Algorithm

In [10]:
```
γ = 1.8
# Optimal value for damping coefficient
# Reference: [27]
```

In [11]:
```
def verlet_algorithm(x, dt):
    v = 0
    # Initial velocity of particle is 0

    t = 0
    # Time starts at 0

    i = 0
    # Indexing starts at 0

    xlist =[]
    vlist = []
    tlist = []
    # Empty lists the positions, velocities and times will be entered into respectively

    xlist.append(x)
    vlist.append(v)
    tlist.append(t)
    # Appends initial values of position(x), velocity(v) and time(t) into their respective lists

    for i in range (0, 100000):
        # When i is between 0 and 100,000

        x_new = x + v * dt + (dt**2) * 0.5 * ( 48/x**13 - 24/x**7 - γ*v + np.sqrt(2*γ)*R_t[i] )
        # Defines the new position values using equation 10

        v_new = (v + 0.5*dt*( 48/(x**13) - 24/(x**7) - γ*v + np.sqrt(2*γ)*(R_t[i] + R_t[i+1]) + 48/(x_new)**13 -24/(x_new)**7) )
                / (1 + 0.5*dt*γ)
        # Defines the new velocity value explicitly by rearranging equation 11

        x = x_new
        v = v_new
        # Redefines x as the new position and v as the new velocity

        t = t + dt
        # Adds the timestep to the time value

        xlist.append(x)
        vlist.append(v)
        tlist.append(t)
        # Adds new position, velocity and times into their respective lists
        # Then code iterates to new next index


    return xlist, vlist, tlist
    # Returns the position, velocity and time lists
```

Validation of the Velocity Verlet Algorithm (Section 3.4)

To validate the velocity Verlet algorithm a simple example is considered - the one-dimensional classical harmonic oscillator. This system has an equation of motion of $-\frac{dV(x)}{dx} = m\frac{d^2x}{dt^2}$

This has exact analytical solutions of $x(t) = Acos(\omega t + \Phi)$ and $v(t) = -\omega Asin(\omega t + \Phi)$

The Velocity Verlet algorithm will be used to solve this equation of motion. The solution found from the velocity Verlet algorithm should match the exact analytical solutions found above.

In [12]:
```
def x(t):
    return A*np.cos(w*t+φ)
# Solution of Position

def v(t):
    return -A*w*np.sin(w*t+φ)
# Solution of Velocity
```

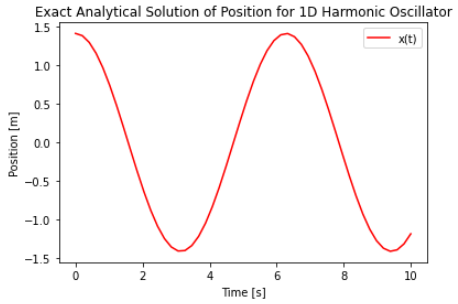
The following constant values are assumed as this is an example to help validate the velocity verlet algorithm.

In [13]:
```
k = 1
E = 1

A = np.sqrt(2*E/k)
w = np.sqrt(1/1)

φ = 0
```

In [14]:
```
t = np.linspace(0,10,50)
```

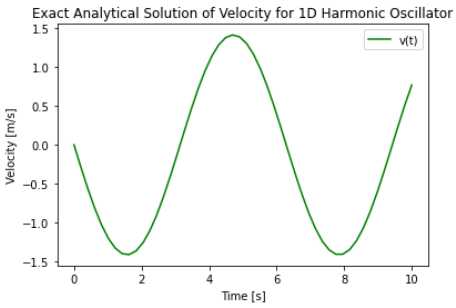
In [15]:
```
plt.plot(t, x(t), color = "red")

plt.legend(["x(t)"])

plt.xlabel("Time [s]")
plt.ylabel("Position [m]")
plt.title("Exact Analytical Solution of Position for 1D Harmonic Oscillator")
```

Out[15]:
```
Text(0.5, 1.0, 'Exact Analytical Solution of Position for 1D Harmonic Oscillator')
```



In [16]:
```
plt.plot(t, v(t), color = "green")
plt.legend(["v(t)"])

plt.xlabel("Time [s]")
plt.ylabel("Velocity [m/s]")
plt.title("Exact Analytical Solution of Velocity for 1D Harmonic Oscillator")
```

Out[16]:
```
Text(0.5, 1.0, 'Exact Analytical Solution of Velocity for 1D Harmonic Oscillator')
```



From equation 13 we see that the force of this system is $F = -\frac{dV(x)}{dx}$

Hence, we can take the mass of the particles to be 1 meaning that we can use Newton's second law to find an expression for acceleration of the particle which we define as: $a = -\frac{dV(x)}{dx}$

Using equation 12 as the potential energy we define the acceleration by $a = -kx$

In [17]:
```
def acceleration(x):
    return -k*(x)
# Function of accleration using the derivative of the potential shown in equation 12
```

In [18]:
```
initial_position = A
initial_velocity = 0
# Assume particle starts at rest
```

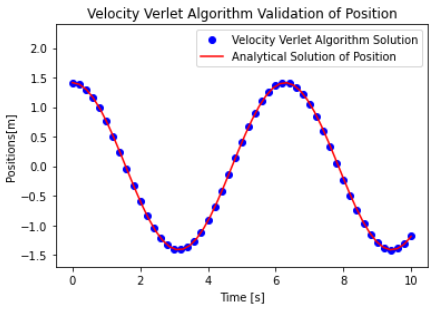
In [19]:
```
def verlet_algorithm(x, dt):
    v = 0
    # Initial velocity of particle is 0

    t = 0
    # Time starts at 0

    i = 0
    # Indexing starts at 0

    xlist =[]
    vlist = []
    tlist = []
    # Empty lists the positions, velocities and times will be entered into respectively

    xlist.append(x)
    vlist.append(v)
    tlist.append(t)
    # Appends initial values of position(x), velocity(v) and time(t) into their respective lists

    for i in range (0, 50):
        # When i is between 0 and 50

        x_new = x + v * dt + (dt**2) * 0.5 * acceleration(x)
        # Defines the new position values using equation 16

        v_new = v + 0.5*dt*(acceleration(x_new) + acceleration(x))
        # Defines the new velocity value explicitly by rearranging equation 16

        x = x_new
        v = v_new
        # Redefines x as the new position and v as the new velocity

        t = t + dt
        # Adds the timestep to the time value

        xlist.append(x)
        vlist.append(v)
        tlist.append(t)
        # Adds new position, velocity and times into their respective lists
        # Then code iterates to new next index


    return xlist, vlist, tlist
    # Returns the position, velocity and time lists
```

In [20]:
```
dt = 0.2
# Time step is 0.2 as we study 50 steps between 0 and 10 seconds
```

In [21]:
```
positions_SHM = verlet_algorithm(initial_position, dt)[0]
velocities_SHM = verlet_algorithm(initial_position, dt)[1]
times_SHM = verlet_algorithm(initial_position, dt)[2]
```

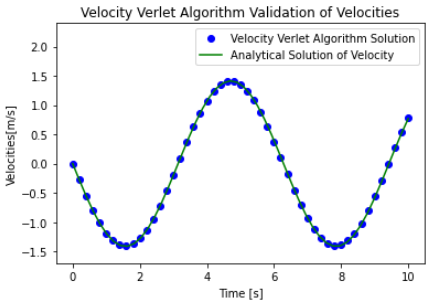
In [22]:
```
plt.plot(times_SHM, positions_SHM, 'bo', markersize =6)

plt.plot(t, x(t), color = "red")
plt.ylim(-1.7,2.4)

plt.ylabel("Positions[m]")
plt.xlabel("Time [s]")
plt.title("Velocity Verlet Algorithm Validation of Position")

plt.legend(["Velocity Verlet Algorithm Solution","Analytical Solution of Position"], loc="upper right")
```

Out[22]:
```
<matplotlib.legend.Legend at 0x15be9d59d30>
```



In [23]:
```
plt.plot(times_SHM, velocities_SHM, 'bo', markersize =6)

plt.plot(t, v(t), color = "green")
plt.ylim(-1.7,2.4)

plt.ylabel("Velocities[m/s]")
plt.xlabel("Time [s]")
plt.title("Velocity Verlet Algorithm Validation of Velocities")

plt.legend(["Velocity Verlet Algorithm Solution","Analytical Solution of Velocity"], loc="upper right")
```

Out[23]: <matplotlib.legend.Legend at 0x15be9db7e50>



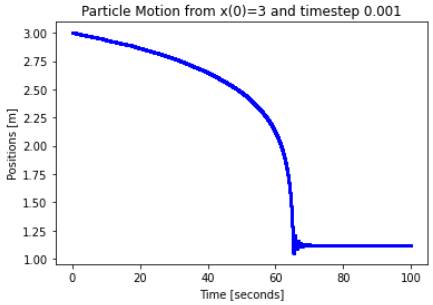
(a) Particle's Motion

In [24]:
```
def verlet_algorithm(x, dt, steps):
    v = 0
    # Initial velocity of particle is 0

    t = 0
    # Time starts at 0

    i = 0
    # Indexing starts at 0

    xlist =[]
    vlist = []
    tlist = []
    # Empty lists the positions, velocities and times will be entered into respectively

    xlist.append(x)
    vlist.append(v)
    tlist.append(t)
    # Appends initial values of position(x), velocity(v) and time(t) into their respective lists

    for i in range (0, steps):
        # When i is between 0 and 100,000

        x_new = x + v * dt + (dt**2) * 0.5 * ( 48/x**13 - 24/x**7 - γ*v + np.sqrt(2*γ)*R_t[i] )
        # Defines the new position values using equation 10

        v_new = (v + 0.5*dt*( 48/(x**13) - 24/(x**7) - γ*v + np.sqrt(2*γ)*(R_t[i] + R_t[i+1]) + 48/(x_new)**13 -24/(x_new)**7) )
                / (1 + 0.5*dt*γ)
        # Defines the new velocity value explicitly by rearranging equation 11

        x = x_new
        v = v_new
        # Redefines x as the new position and v as the new velocity

        t = t + dt
        # Adds the timestep to the time value

        xlist.append(x)
        vlist.append(v)
        tlist.append(t)
        # Adds new position, velocity and times into their respective lists
        # Then code iterates to new next index


    return xlist, vlist, tlist
    # Returns the position, velocity and time lists
```

In [25]:
```
positions = verlet_algorithm(3, 0.001, 100000)[0]
# Gives the positions of the particles array when the particle starts at x=3 with timestep 0.001

velocities = verlet_algorithm(3, 0.001, 100000)[1]
# Gives the Velocities of the particles array when the particle starts at x=3 with timestep 0.001

time = verlet_algorithm(3, 0.001, 100000)[2]
# Gives the Times of the particles array when the particle starts at x=3 with timestep 0.001
```

In [26]:
```
plt.plot(time ,positions,'bo', markersize='1')
plt.xlabel("Time [seconds]")
plt.ylabel("Positions [m]")
# plt.grid(True)

plt.title("Particle Motion from x(0)=3 and timestep 0.001")
```

Out[26]:
```
Text(0.5, 1.0, 'Particle Motion from x(0)=3 and timestep 0.001')
```



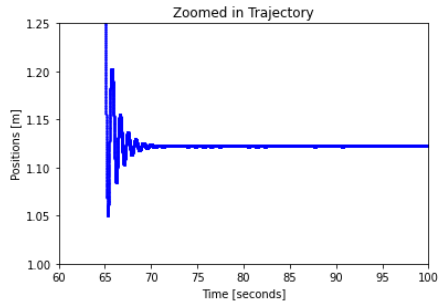
Where does this diverge

To study this behaviour more in depth we look at the behaviour of the particle as it diverges

Limiting the x-axis to being between 60 and 100 gives us an inidcation of the periodic behaviour as the partcile's position gets closer and closer to the r value of the equilbirium point of the Lennard-Jones potential which is $r_0 = 2^{\frac{1}{6}} = 1.122462048$.

In [27]:
```
plt.plot(time ,positions,'bo', markersize='1')
plt.xlabel("Time [seconds]")
plt.ylabel("Positions [m]")
# plt.grid(True)

plt.title("Zoomed in Trajectory")

plt.xlim(60,100)
plt.ylim(1,1.25)
```

Out[27]:
```
(1.0, 1.25)
```



It is evident that the breif periodic behaviour of the partcile stops at around 70 seconds. Hence, an average value of the position can be taken after this time to give an inidication of the position the partcile finishes at.

In [28]:
```
positions_after_70s = positions[70000:]
mean(positions_after_70s)
# Gives the mean value of the positions after 70 seconds
```

Out[28]:
```
1.1224596599034407
```

In [29]:
```
print(f"The mean value of the position after 70 seconds is {mean(positions_after_70s):.6g}")
print("\nThe equilibrium position of the Lennard-Jones potential is 1.122462048")
print("\nHence, the mean value of the position after 70 seconds matches the equilibirum position to 5 significant figures")
```

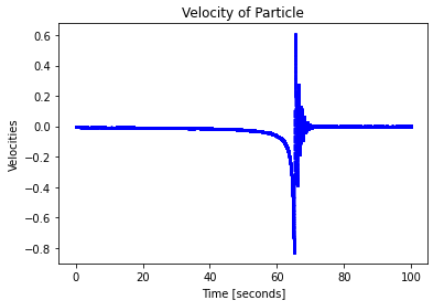
```
The mean value of the position after 70 seconds is 1.12246

The equilibrium position of the Lennard-Jones potential is 1.122462048

Hence, the mean value of the position after 70 seconds matches the equilibirum position to 5 significant figures
```

In [30]:
```
#calculate standard error of the mean
np.std(positions_after_70s, ddof=1) / np.sqrt(np.size(positions_after_70s))
```

Out[30]:
```
1.0821050343572205e-06
```

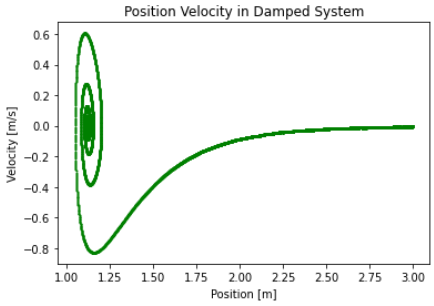
In [31]:
```
plt.plot(time , velocities,'bo', markersize='1')
plt.xlabel("Time [seconds]")
plt.ylabel("Velocities")
plt.title("Velocity of Particle")
```

Out[31]:
```
Text(0.5, 1.0, 'Velocity of Particle')
```



In [32]:
```
plt.plot(positions, velocities, 'go', markersize = 1)

plt.xlabel("Position [m]")
plt.ylabel("Velocity [m/s]")
plt.title("Position Velocity in Damped System")
```

Out[32]:
```
Text(0.5, 1.0, 'Position Velocity in Damped System')
```



In [33]:
```
positions_10th_step = positions[0::10]
# List of the positions at every 10th Step
```

In [33]:
```
positions_10th_step = positions[0::10]
# List of the positions at every 10th Step
```

In [34]:
```
print("Positions at every 10th step is")
positions_10th_step
```

```
Positions at every 10th step is
```

Out[34]:
```
[3,
 3.0000010858381616,
 3.0000042545984464,
 3.000009086208006,
 3.0000147171890053,
 3.0000178514739417,
 3.0000207730684805,
 3.000023350135384,
 3.0000219430805655,
 3.0000197696548505,
 3.0000142763317745,
 3.0000057896580903,
 2.9999941510421535,
 2.9999834897053437,
 2.9999732584406194,
 2.999961614669764,
 2.9999500747650694,
 2.999937235202397
```

In [35]:
```
velocities_10th_step = velocities[0::10]
# List of Velocities at every 10th Step
```

In [35]:
```
velocities_10th_step = velocities[0::10]
# List of Velocities at every 10th Step
```

In [36]:
```
print("Velocities at every 10th step is")
velocities_10th_step
```

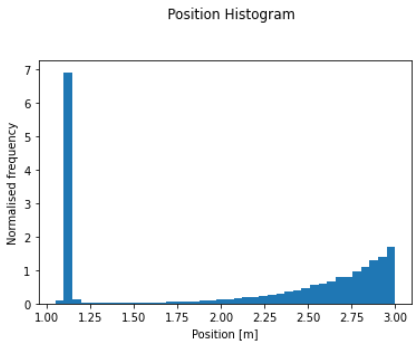
```
Velocities at every 10th step is
```

Out[36]:
```
[0,
 0.00019665788136078173,
 0.00036534546135705607,
 0.0006359690461034481,
 0.0003972848087419899 6,
 0.0002776078617644757,
 0.00031593067687938665,
 -4.435056562696643e-05,
 -0.00019753714836798583,
 -0.0003152355314548799,
 -0.0003764538042609 09,
 -0.0002655923643147697,
 -0.0001133430439462 5691,
 -0.000985555035224285,
 -0.0010786244921010405,
 -0.0011480841480937534,
 -0.0013238540377427277,
 -0.0011741796942426757
```

Position Histogram

In [37]:
```
amplitudes_x, amplitudesbins_x, npatches_x = plt.hist(positions, bins=40, density = True)

plt.title("Position Histogram")
plt.xlabel("Position [m]")
plt.ylabel("Normalised frequency")
```

Out[37]:
```
Text(0, 0.5, 'Normalised frequency')
```

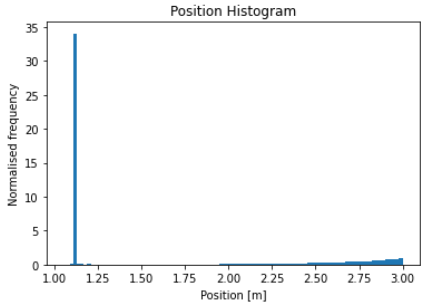


To try to make the histogram distribution smoother the number of velocity Verlet algorithm steps is increased and the number of bins is increased.

In [38]:
```
G = sto.GaussianNoise(1)
times = np.arange(0,100 ,100/200002)
# Creates an array of the 100,000 steps with timestep 0.001

R_t = G.sample_at(times)
```

In [39]:
```
positionsch = verlet_algorithm(3, 0.001, 200000)[0]
```

In [40]:
```
amplitudes_x, amplitudesbins_x, npatches_x = plt.hist(positionsch, bins=100, density = True)

plt.title("Position Histogram")
plt.xlabel("Position [m]")
plt.ylabel("Normalised frequency")
```



We can see that increasing these paramters will just cause the value between 1.10 and 1.15 to blow up and hence, the other values would be impossible to see. Hence, the orginal graph is taken as the the distribution histogram for the positions.

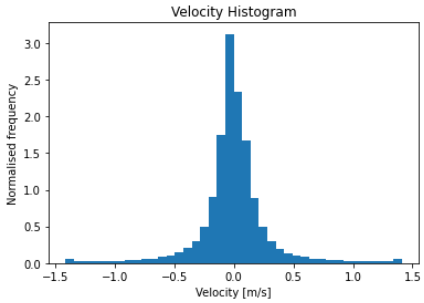
Velocity Histogram

In [41]:
```
v=0
```

In [44]:
```
velocties = verlet_algorithm(3, 0.001, 100000)[1]
```

In [45]:
```
amplitudes_v, amplitudesbins_v, npatches_v = plt.hist(velocties, bins=40, density = True)

plt.title("Velocity Histogram")
plt.xlabel("Velocity [m/s]")
plt.ylabel("Normalised frequency")
```

Out[45]:
```
Text(0, 0.5, 'Normalised frequency')
```



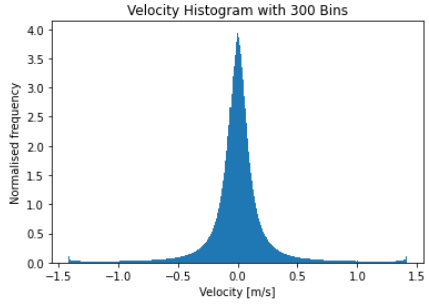
To try to make the histogram distribution smoother the number of velocity Verlet algorithm steps is increased and the number of bins is increased.

In [46]:
```
G = sto.GaussianNoise(1)
times = np.arange(0,100 ,100/1000002)
# Creates an array of the 1,000,000 steps with timestep 0.001

R_t = G.sample_at(times)
# Generates the corresponding value of the stochastic function of each time
```

In [47]:
```
velocities_increased_steps = np.array(verlet_algorithm(3,0.001, 1000000))[1]
# Gives an array of all the velocities
```

In [48]:
```
amplitudes_v, amplitudesbins_v, npatches_v = plt.hist(velocities_increased_steps, bins=300, density = True)

plt.title("Velocity Histogram with 300 Bins")
plt.xlabel("Velocity [m/s]")
plt.ylabel("Normalised frequency")
```

Out[48]:
```
Text(0, 0.5, 'Normalised frequency')
```



(c) Average Velocity and Temperature

Average Velocity

In order to find the correct average velocity based off the velocity distribution histogram we fit both a Gaussian curve and a Lorentzian curve in order to see which fits better with the data. Then the when the suitable distribution is fitted the average value will be calculated more accurately than a simple mean of the velocities.

Gaussian Function $G(x) = ae^{-\frac{(x-x_0)^2}{2\sigma^2}}$ [49]

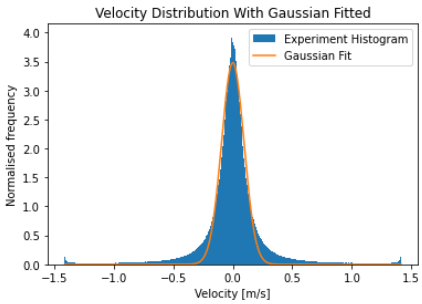
Lorentizan Function $L(x) = \frac{1}{\pi}\frac{a}{((x-x_0)^2+a^2)}$ [52]

where a is the amplitude of the distribution, x_0 is the mean value and σ is the standard deviation

Gaussian Fitting

In [49]:
```
def gaussian(x, mean, amplitude, standard_deviation):
    return amplitude * np.exp( - (x - mean)**2 / (2*standard_deviation ** 2))
```

In [50]:
```
bin_heights, bin_borders, _ = plt.hist(velocities_increased_steps, bins='auto', label='Experiment Histogram', density =True)

bin_centers = bin_borders[:-1] + np.diff(bin_borders) / 2
# Calculates the centre of the bins using the paramters found from the histograms

popt, _ = curve_fit(gaussian, bin_centers, bin_heights, p0=[1., 0., 1.])
# Fits the data

x_interval_for_fit = np.linspace(bin_borders[0], bin_borders[-1], 10000)
plt.plot(x_interval_for_fit, gaussian(x_interval_for_fit, *popt), label='Gaussian Fit')
plt.legend()

plt.xlabel("Velocity [m/s]")
plt.ylabel("Normalised frequency")
plt.title("Velocity Distribution With Gaussian Fitted")

# Reference - [54] A. Mangs (2022) Fit a curve to a histogram in Python, DevPress.
# Available at: https://devpress.csdn.net/python/630463e97e6682346619af03.html (Accessed: November 11, 2022).
```



Clearly the amlitude of the histogram is higher than the fitted gaussian function. This leads us to believe that a Lorenztian fitting would be much more suitable for this set of data.

Lorentzian Fitting

In [51]:
```
range_avg_v = np.linspace(min(velocities_increased_steps), max(velocities_increased_steps), 300)
# Gives the average value of the velocity within each bin to help us fit a Lorentzian Function to the data
```

In [52]:
```
# Lorentzian parameter fitting
def lorentzian(x, a, x0):
    return (1/ np.pi) * a / ((x-x0)**2 + a**2)

pguess = [1 / (np.pi * max(amplitudes_v)), sum(range_avg_v * amplitudes_v) /
          sum(amplitudes_v)]

# Fit the data
popt, pcov = curve_fit(lorentzian, range_avg_v, amplitudes_v, p0 = pguess)

# Results
bandwidth, avg_value = popt[0], popt[1]
# Gives the velocity at maximum amplitude and also gives the full
# bandwidth at half maximum amplitude

# Reference - [55] B. Prudholm (2014) Fit points to a Lorentzian curve and find center and half
# maximum bandwidth in Python. [online] Stack Overflow.
# Available at: <https://stackoverflow.com/questions/24437070/fit-points-to-a-lorentzian-
# curve-and-find-center-and-half-maximum-bandwidth-in-p> [Accessed 3 October 2022].
```

In [53]:
```
pcov
# Covariance Matrix
```

Out[53]:
```
array([[1.18999983e-07, 3.95646239e-15],
       [3.95646239e-15, 1.18980840e-07]])
```

In [54]:
```
print(f"The Uncertainty on the Average Velocity of the particle is = {np.sqrt(pcov[1,1]):.2g} m/s to 2 significant figures")
```

```
The Uncertainty on the Average Velocity of the particle is 0.000043 m/s to 2 significant figures
```

In [55]:
```
bandwidth
```

Out[55]:
```
0.08195052701936639
```

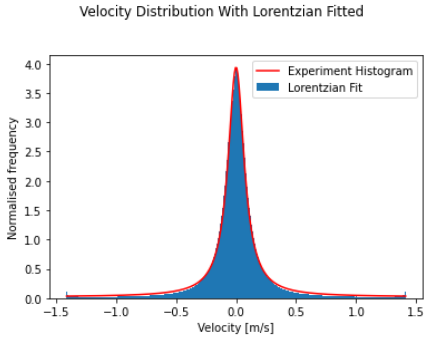
In [56]:
```
avg_value
```

Out[56]:
```
-0.0003729317319291526
```

In [57]:
```
y = ((max(amplitudes_v)-min(amplitudes_v) )
                / ( 1+ ((range_avg_v - avg_value)/(bandwidth))**2) + min(amplitudes_v) )
# Lorentzian function with correct paramters found using the parameter fitting above
```

In [58]:
```
#plt.plot(range_avg_v, amplitudes_v, 'o', color = 'black', markersize =5)
plt.plot(range_avg_v , y, color = 'red')

amplitudes_v, amplitudesbins_v, npatches_v = plt.hist(velocities_increased_steps, bins=300, density = True)

plt.xlabel("Velocity [m/s]")
plt.ylabel("Normalised frequency")

plt.legend(['Experiment Histogram', 'Lorentzian Fit'])
```

Out[58]:
```
<matplotlib.legend.Legend at 0x15beed5e4f0>
```



In [59]:
```
print(f"The Average Velocity of the particle is = {avg_value:.3g} m/s to 2 significant figures")
```

```
The Average Velocity of the particle is = -0.000373 m/s to 2 significant figures
```

In [60]:
```
velocities_increased_steps[2]
```

Out[60]:
```
-1.503251890784593e-05
```

Temperature

Input Temperature

The input temperature is given by $k_B T = 1$ and we know k_B to be $1.380649x10^{-23}$

Therefore temperature satisfies the equation $T = \frac{1}{1.380649x10^{-23}}$

In [61]:
```
input_T = 1/(1.380649*10**-23)
input_T
```

Out[61]:
```
7.24297051603992e+22
```

In [62]:
```
print(f"Thus the input temperature was = {input_T:.3g}")
```

```
Thus the input temperature was = 7.24e+22
```

Observed Temperature

Using the Equipartition Theorem we can write the root mean squared speed of a particle to be $v_{RMS} = \sqrt{\frac{k_B T}{m}}$

where k_B is the Boltzmann Constant, T is the temperature of the particle and m is the mass of the particle.

This equation can be rewritten to solve for temperature as $T = \frac{m(v_{RMS})^2}{k_B}$

Where v_{RMS} can be written as $v_{RMS} = \sqrt{\frac{v_1^2+v_2^2+...+v_n^2}{n}}$ [56]

In [63]:
```
v_rms = np.sqrt(sum(velocities_increased_steps**2)/len(velocities add _increased_steps))
```

In [64]:
```
v_rms
```

Out[64]:
```
1.01427625724
```

In [65]:
```
def T(v):
    return (m*(v**2))/(1*k)
```

In [66]:
```
m = 1
k = 1.380649 * 10 **-23
```

In [67]:
```
T(v_rms)
```

Out[67]:
```
7.452577312950122e+22
```

In [68]:
```
print(f"Observed temperature is = {(T(v_rms)):.3g} K")
```

```
Observed temperature is = 7.45e+22 K
```

In [69]:
```
v_uncert = (0.001)**2

# Uncertainty on each value of the velocity due to the Global error on the velocity verlet algorithm with timsetep = 0.001
```

In [70]:
```
Vrms_Uncert = np.sqrt(v_uncert/len(velocities_increased_steps))

# Formula can be derived using propogation of errors showns in section 8.1
```

In [71]:
```
Vrms_Uncert
# Uncertainty on the root mean square speed
```

Out[71]:
```
1e-06
```

In [72]:
```
T_uncert = T(v_rms)*((2*Vrms_Uncert/v_rms)**2)
```

In [73]:
```
print(f"Uncertainty on Observed Temperature = {T_uncert:.3g} ")
```

```
Uncertainty on Observed Temperature = 2.9e+11
```

The observed temperature is lower than the input temperature. Meaning the kinetic energy of the observed particles is lower than the input characteritsic thermal energy.

Hence, the particles aren't going as fast as when the energy unit is equal to the characteristic thermal energy.

Solvent effects causes the kinetic energy of partciles in the lennard-jones potential to decrease.

(d) Potential of Mean Force

The potential of mean force can be expressed as $U_{PMF} = -k_B T\, ln(1 + h(r))$

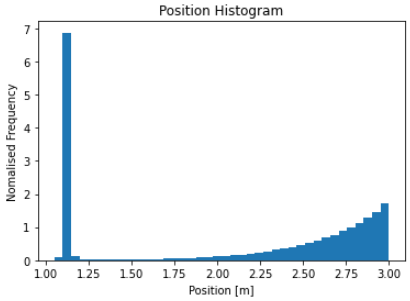
where $h(r)$ is the position distribution function. The position distribution function can be found by interploation to the position distribution histogram found in part (b). Thus, the potential of mean force can be compared to the Lennard-Jones potential

In order to show that interplotating is an accurate measure of the position distribution we first use 40 bins for the position histogram. Then we will show that the interpolation function matches the data points.

In [74]:
```
positions = verlet_algorithm(3, 0.001, 100000)[0]
```

In [75]:
```
amplitudes_x_40, amplitudesbins_x_40, npatches_x_40 = plt.hist(positions, bins=40, density=True)
# Increase the bin size of the position distribution histogram

plt.title("Position Histogram")
plt.xlabel("Position [m]")
plt.ylabel("Nomalised Frequency")
```

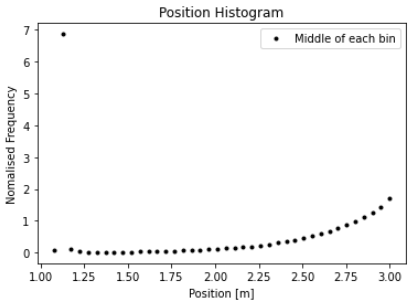
Out[75]:
```
Text(0, 0.5, 'Nomalised Frequency')
```



In [76]:
```
range_avg = np.linspace(0.025+min(positions), max(positions), 40)
# Gives the average value of the position within each bin to help us fit a curve to the data
```

In [77]:
```
plt.plot(range_avg, amplitudes_x_40, 'o', color = 'black', markersize =3)

plt.legend(['Middle of each bin'])

plt.title("Position Histogram")
plt.xlabel("Position [m]")
plt.ylabel("Nomalised Frequency")
```

Out[77]:
```
Text(0, 0.5, 'Nomalised Frequency')
```



Using interpolate to Find h(r)

In [78]:
```
len(range_avg)
```

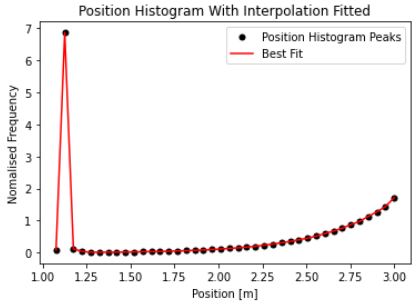
Out[78]:
```
40
```

In [79]:
```
len(amplitudes_x)
```

Out[79]:
```
100
```

In [80]:
```
f1 = interpolate.interp1d(range_avg, amplitudes_x_40,kind = 'nearest')
# Using the interpld1d function this interpolates between the middle of the bins and the amplitudes of the bins
# to find the nearest fit
```

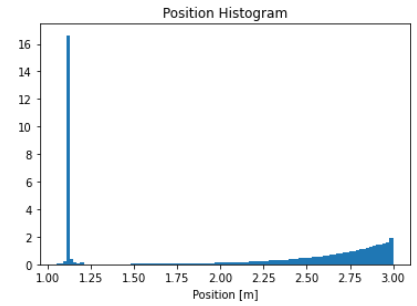
In [81]:
```
xnew = np.linspace(min(range_avg), max(range_avg),40)

plt.plot(range_avg, amplitudes_x_40, 'ko', xnew, f1(xnew), 'r-', markersize=5)
# Plots

plt.legend(['Position Histogram Peaks', 'Best Fit'], loc = 'best')
plt.title("Position Histogram With Interpolation Fitted")
plt.xlabel("Position [m]")
plt.ylabel("Nomalised Frequency")

plt.show()
```



Now the bin size is increased to 300 in order to improve the fit of the interpolation.

In [82]:
```
amplitudes_x, amplitudesbins_x, npatches_x = plt.hist(positions, bins=100, density=True)
# Increase the bin size of the position distribution histogram

plt.title("Position Histogram")
plt.xlabel("Position [m]")
```

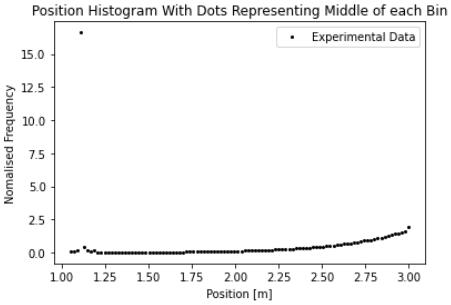
Out[82]:
```
Text(0.5, 0, 'Position [m]')
```



In [83]:
```
range_avg = np.linspace(min(positions), max(positions), 100)
# Gives the average value of the position within each bin to help us fit a curve to the data
```

In [84]:
```
plt.plot(range_avg, amplitudes_x, 'o', color = 'black', markersize =2)

plt.legend(['Experimental Data'])

plt.title("Position Histogram With Dots Representing Middle of each Bin")
plt.xlabel("Position [m]")
plt.ylabel("Nomalised Frequency")
```

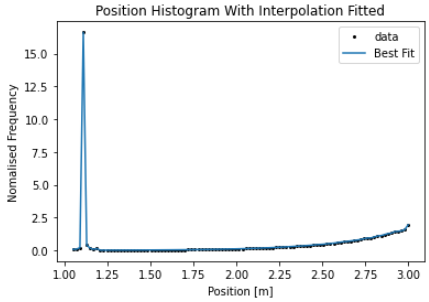
Out[84]:
```
Text(0, 0.5, 'Nomalised Frequency')
```



Using interpolate to Find U_{PMF}

In [85]:
```
f1 = interpolate.interp1d(range_avg, amplitudes_x,kind = 'nearest')
```

In [86]:
```
xnew = np.linspace(min(range_avg), max(range_avg),100)

plt.plot(range_avg, amplitudes_x, 'ko', xnew, f1(xnew), '-', markersize=2)

plt.legend(['data', 'Best Fit'], loc = 'best')
plt.title("Position Histogram With Interpolation Fitted")
plt.xlabel("Position [m]")
plt.ylabel("Nomalised Frequency")

plt.show()
```

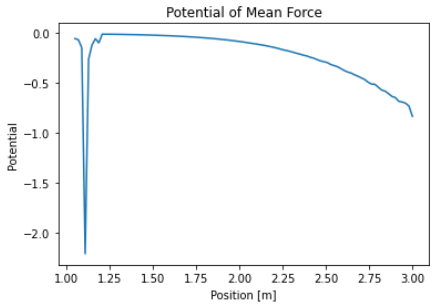


In [87]:
```
k = 1.380649 * 10 **-23
T = T(v_rms)
```

In [88]:
```
U_PMF = -k*T*np.log(1+f1(xnew))
# Equation 6 using the interpolation function
```

In [89]:
```
plt.plot(xnew, U_PMF)

plt.title("Potential of Mean Force")
plt.xlabel("Position [m]")
plt.ylabel("Potential")
```

Out[89]:
```
Text(0, 0.5, 'Potential')
```



Lennard-Jones Potential Plot

In [90]:
```
def Lennard_Jones_Pot(e, σ, r):
    return 4*e*((σ/r)**12-(σ/r)**6)
```

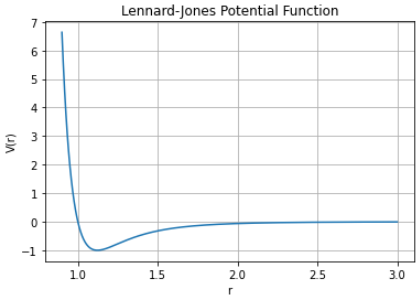
Lennard-Jones Potential Plot

In [90]:
```
def Lennard_Jones_Pot(e, σ, r):
    return 4*e*((σ/r)**12-(σ/r)**6)
```

In [91]:
```
e = 1
σ = 1
```

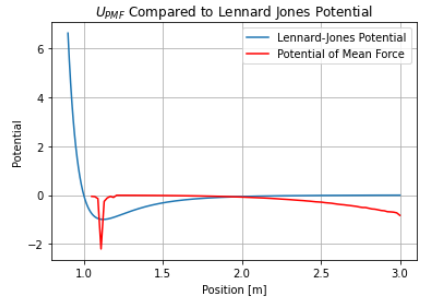
In [92]:
```
r = np.linspace(0.9,3,500)

plt.plot(r,Lennard_Jones_Pot(1, 1, r))

plt.xlabel("r")
plt.ylabel("V(r)")
plt.grid(True)
plt.title("Lennard-Jones Potential Function")
```

Out[92]: Text(0.5, 1.0, 'Lennard-Jones Potential Function')



Compare with Lennard-Jones Potential

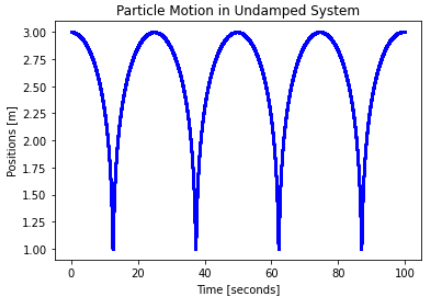
In [93]:
```
plt.plot(r,Lennard_Jones_Pot(1, 1, r))

plt.plot(xnew, U_PMF, color= 'r')

plt.legend(["Lennard-Jones Potential","Potential of Mean Force"])

plt.title("$U_{PMF}$ Compared to Lennard Jones Potential")
plt.xlabel("Position [m]")
plt.ylabel("Potential")
plt.grid(T)
```



In [94]:
```
γ = 0
# Damping Coefficient is now equal to 0
```

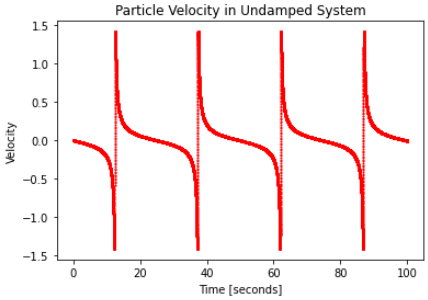
In [95]:
```
# Keep the same initial condition of starting at x=3 and carrying out 100,000 steps with a timestep of 0.001 seconds.
positions = verlet_algorithm(3, 0.001, 100000)[0]
velocities = verlet_algorithm(3, 0.001, 100000)[1]
time = verlet_algorithm(3, 0.001, 100000)[2]
```

In [96]:
```
plt.plot(time ,positions,'bo', markersize='1')
plt.xlabel("Time [seconds]")
plt.ylabel("Positions [m]")

plt.title("Particle Motion in Undamped System")
```



In [97]:
```
plt.plot(time,velocities,'ro', markersize='1')
plt.xlabel("Time [seconds]")
plt.ylabel("Velocity")
plt.title("Particle Velocity in Undamped System")
```

Out[97]: Text(0.5, 1.0, 'Particle Velocity in Undamped System')



In [98]:
```
plt.plot(positions, velocities, color = 'g')
plt.xlabel("Position [m]")
plt.ylabel("Velocity [m/s]")
plt.title("Position Velocity in Undamped System")
```

Out[98]: Text(0.5, 1.0, 'Position Velocity in Undamped System')