CHAPTER - III

3. Molecular Dynamics Simulation: Theory, Numerical Methods and Computer Programmes for the proposed model

3.1 Numerical Method of solving Non-linear PDE
(i) Step by Step approach
(ii) Discretization

3.2 Molecular Dynamics
(i) Introduction
(ii) Boundary Condition
(iii) Integration Scheme
(iv) Verlet Algorithm
(v) Types of MD Simulation
(vi) Initial Values
   (a) Choice of Position
   (b) Choice of Velocity

3.3 Implementation of MD for the Proposed Model
(i) Dimensionless Equation
(ii) MD Simulation
(iii) Phase Space
(iv) MD Program Description

3.4 Appendix - Program Listing
3.1 Numerical Method of Solving Partial Differential Equations

Numerical approach is the third method in Physics in addition to the already existing theoretical and experimental methods. With the advent of modern computers (fast and large size memory) many physical problems are attracting renewed interest. Of these, study of non-linear system is of great interest. As there are normally no analytical solutions for non-linear equations one follows the numerical procedures. In the following section (i) we shall summarise the procedures involved in solving non-linear PDE and their pros and cons, and in section (ii) we shall discuss the method which is free from such shortcomings.

(i) Step by Step Approach

The equations of motion associated with the proposed model in this thesis are coupled non-linear partial difference-differential equations (2.9) and no analytical solutions are known to exist. In order to get an analytical solution the following procedure is adopted.

\[
\text{General Equations} \rightarrow \text{Coupled non-linear PDEs} \rightarrow \text{Coupled non-linear ODEs} \rightarrow \text{Uncoupled non-linear ODEs.}
\]

In the last chapter it was shown that the uncoupled nonlinear ODE has an analytical solution. To find the solution of the general difference-differential equation (2.9), one should start with this analytical solution as initial condition and solve numerically the coupled PDEs.

The procedure is complicated, tedious and hence time consuming. For example the steps involved in solving coupled nonlinear ODE from the uncoupled nonlinear ODE is as follows:
Non linear coupled ODEs (2.13) are given as

\[
\frac{1}{4d^2} \Phi_{ss} [\nu^2 - v_o^2] - \frac{gf}{m_1(2f + g)} (4\eta - \eta_{ss} - 2\Phi) = 0 \quad \text{...}(3.1a)
\]

\[
\frac{1}{4d^2} \eta_{ss} [c_a^2 - \nu^2] - \frac{f^2}{m_2(2f + g)} (4\eta - 2\Phi) + \frac{f}{m_2} (\Phi - 2\eta) \frac{1}{m_2 u_o^2} \frac{\partial V}{\partial \eta} = 0 \quad \text{...}(3.1b)
\]

When \( g << f \), the equations become uncoupled ODEs

\[
\frac{1}{4d^2} \Phi_{ss} [\nu^2 - v_o^2] + \frac{g}{m_1} (\Phi - 2\eta) - \frac{g}{2m_1} \eta_{ss} = 0 \quad \text{...}(3.2a)
\]

\[
\frac{1}{4d^2} \eta_{ss} [c_a^2 - \nu^2] - \frac{1}{m_z u_o^2} \frac{\partial V}{\partial \eta} = 0 \quad \text{...}(3.2b)
\]

Equation (3-2b) has an analytical solution of the kink soliton type (tanh function for \( \nu = v_o \)). To solve equations (3-1) numerically using solutions of equations (3-2), the following steps are involved.

**Step 1**: Solve equations (3-2b) analytically and plot it. → Analytical \( \eta \)

**Step 2**: put \( \nu = v_o \) in equation (3-2a) and solve for \( \Phi \) using step 1 → Analytical \( \Phi \)

**Step 3**: solve equation (3-2b) numerically (RK method) to achieve the same analytical result as in step 1. Take initial values \((\eta, d\eta/ds)\) from analytical solution.

**Step 4**: Now solve equation (3-2a) numerically by using analytical results of steps (1 and 2) as initial values. First try for \( \nu = v_o \) and then extend the study for \( \nu \) close to \( v_o \)

**Step 5**: Solve the general equation (3.1a and b) with a condition \( g << f \) using the result of steps (3 & 4).

**Step 6**: Now analyse the equation (3.1a, b) numerically for different cases \( g = f, g >> f \).

**Step 7**: Time Evolution (solving PDE)
The next stage is to solve the non-linear PDE. This can be done in two ways. One solves for either temporal evolution (space fixed) or spatial evolution (time fixed). It is very difficult to get an overall view of how the system will evolve with both space and time. In order to get a general view of the dynamics, one should run the program for many initial values and many space and time steps and analyse these results. Obviously such an approach is tedious, time consuming and possibly unreliable.

(ii) Discretization:

As far as ODE is concerned there are two methods available to solve numerically, but for PDE there is only one method available.

![Diagram of solving differential equations](image)

**Fig.3.1 Method of solving Differential Equations**

ODE involves one independent variable (i.e. one dimensional) and the solution is a point at a given time. But PDE involves two (or more) independent variables and solution is 2 dimensional i.e. grid point or mesh (**Fig. 3.1**).
To solve PDE numerically, one must discretize the PDE using Taylor series. But fortunately as this work is concerned with lattice dynamics, the equation governing its dynamics is itself in the difference differential form (2.9) and hence there is no need to discretise in space. But temporal part is in differential form and if the problem is formulated using molecular dynamics (MD), the temporal part is taken care of by some standard procedures. In these procedures time is discretized using Euler, Verlet and Beeman techniques [16,18,37]. One can keep either the temperature or total energy constant for the physical system while carrying out the molecular dynamics (MD) studies. The next section describes in detail the MD techniques.

Depending upon the computer capacity, the size of system can be chosen say $10^2 - 10^4$ particles. The simulation difficulty increases with the system dimensions. The abilities of hardware and software are taken into consideration to decide the dimension of system to be simulated. We have simulated for a system containing 1000 unit cells (2000 particles).

3.2 Molecular Dynamics

(i) Introduction

Molecular dynamics studies have the following advantages. The molecular dynamics method as the name suggests calculates properties of many particle systems using the dynamics (equations of motion) at microlevel, and one can obtain the static as well as the dynamic properties of the system [37,38]. The approach taken by the MD method is to solve the equations of motion numerically on a computer.
Gases, liquids and solids are examples of systems that contain many mutually interacting particles. Given the dynamics at the microscopic level, one can understand the observed macroscopic behaviour of these systems.

One way to understand the behaviour of a many particle system (correlation of macroscopic properties and microscopic dynamical quantities) is to begin from the known intermolecular interaction and equation of motion and do a computer simulation of its dynamics. This approach can be applied to a system of several hundred to a million particles depending upon the power of the computer. It provides a lot of insight into the behaviour of gases, liquids and solids.

(ii) Boundary Conditions

The ultimate goal of this simulation is to understand the behaviour of bulk system, i.e. system containing particles of the order of \(N = 10^{25}\). In bulk system the fraction of particles near the surface of the system is negligibly small compared to total number of particles of the system. However, the number of particles that can be studied in a MD simulation is typically \(N = 10^3 - 10^5\). For these small systems, the fraction of particles near the surface is significant, and hence the behaviour of such a system would be dominated by surface effects and so not very realistic. The most common way to minimize surface effects and to simulate more closely the properties of a bulk system is to use what is known as periodic boundary conditions (PCB).

For one dimensional lattice chain, this boundary condition is easily specified. We simply join the two remote ends of the same chain that connect the lattice points. This is called Born-
van Karman Periodic Boundary Condition. (Fig.3-2). Alternatively one can keep the ends fixed and it is called fixed end boundary condition.

![Fig.3.2 Implementation of Periodic Boundary Condition](image)

To arrive at the dynamical variables (displacement, velocity) for \( n \)th particle at a time \( t + \Delta t \), one should have the knowledge of acceleration at time \( t \), which in turn depends upon the dynamical variables of \( (n + 1) \)th and \( (n - 1) \)th particles at time \( t \) (Fig.3-3).

![Fig.3.3 Time Evolution of an Ensemble](image)

(iii) Integration scheme (Discretization of time)

To solve the equation of motion in a computer, we construct a finite difference scheme for the differential equations to the highest possible order and from the difference equation we then derive the recursion relation for the positions and velocities [37,38]. These algorithms are
performed step by step. At each step the position and velocity are obtained first at time $t_1$ then at $t_2 > t_1$ etc. Hence the integration proceeds in the time direction. This is the time integration algorithm.

The recursion relation must allow efficient evaluation. In addition, the scheme must be numerically stable. Naturally, there will be an error involved due to the transfer from a time description in terms of continuous variables with differential operator ($df$) to a description with discrete variable and finite difference operator ($\Delta t$).

The most straightforward discretization of the differential form stems from the Taylor expansion. The idea is to find the algorithm of a discrete version from the differential operator with suitable assumptions and then expand the variable in a Taylor series. There are many integration schemes available due to Euler, Verlet, Beeman, Runge-Kutta etc. [37].

(iv) Verlet Algorithm

In this work, the verlet integration scheme has been adopted and given below is a short summary of the algorithm. Consider the motion of particle in one dimension and write the equation of motion in the form

$$\frac{dv}{dt} = a(t) \quad \ldots(3.3a)$$

$$\frac{dx}{dt} = v(t) \quad \ldots(3.3b)$$
The goal of finite difference method is to determine the values of $x_{n+1}$ and $v_{n+1}$ at time $t_{n+1} = t + \Delta t$. The nature of the integration algorithm can be understood by writing.

\[
\begin{align*}
x_{n+1} &= x(t + \Delta t) \\
v_{n+1} &= v(t + \Delta t) 
\end{align*}
\] ...\(3.4\)

and expanding in a Taylor’s series i.e.

\[
\begin{align*}
x_{n+1} &= x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 \\
v_{n+1} &= v_n + a_n \Delta t + ...
\end{align*}
\] ...(3.5a) ...(3.5b)

From equation (3-5a)

\[
x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 ...
\]

similarly \[x_{n-1} = x_n - v_n u_n \Delta t + \frac{1}{2} a_n (\Delta t)^2 ...
\]

From these one arrives at the following relations

\[
\begin{align*}
x_{n+1} &= 2x_n - x_{n-1} + \frac{1}{2} a_n (\Delta t)^2 \\
v_{n+1} &= v_n + \frac{1}{2} (a_n + a_{n+1}) \Delta t
\end{align*}
\] ...(3.6) ...(3.7)

This is the verlet integration scheme (3.6,3.7).

(v) Types of MD simulation

There are two types of MD simulation depending upon the nature of the problem.

a) Constant energy -- micro canonical ensemble

b) Constant temperature -- canonical ensemble

The present work involves constant temperature MD simulation. To maintain constant temperature the system is in contact with a heat reservoir. This is numerically achieved by scaling velocities at each time step so that the kinetic energy of the system will be a constant.
Constant velocity implies the constancy of temperature. Because of the constant temperature the total energy of the system will fluctuate.

**(vi) Initial values**

From numerical/mathematical point of view, MD simulation is an initial value problem.

**(a) Choice of position**

One starts the simulation with the system of particles that are kept in positions in

1) regular array (ground state configuration) or
2) array + small random displacement or
3) random positions

The initial positions chosen in the present work is ground state configuration + random displacement i.e. particles are displaced around the equilibrium positions. The reason for the choice of this is as follows:

As we have an ensemble of lattice points consisting of about 2000 particles, it is good to start with the ground state configuration. Ground state configuration for the proposed model is obtained by analytical method.(Chapter II). To be more realistic, ground state + fluctuation are assumed for initial state configuration. This fluctuation is achieved using random numbers. For η the initial value is around -1 and for Φ it is around -2.

**(b) Choice of velocity**

For a collection of particles, the velocity of particles obey the distribution laws. From statistical considerations, velocity, temperature and average of displacements can be related and from this velocity can be calculated. The initial velocity is purely dependent on temperature.
3.3 Implementation of MD for the Proposed model

(i) Dimensionless equations

The dimensionless form of the equations have an advantage in the numerical methods. Too large or too small values can be avoided in the calculations. The number of arithmetic operations can also be minimised so that the efficiency of the program increases.

Considering equation (2.9)

\[ m_1 \ddot{\Phi}_n = F_1(\eta_{n+1} + \eta_{n-1} + 2\eta_n) - 2F_1F_2 + F_1(\Phi_{n+1} + \Phi_{n-1} - 2\Phi_n) \]

\[ m_2\ddot{\eta}_n = F_2(\eta_{n+1} + \eta_{n-1} + 2\eta_n) + [f - 2F_2] \Phi_n + f_2(\eta_{n+1} + \eta_{n-1} - 2\eta_n) - 2F_2\eta_n + \frac{4\epsilon_0\eta_n}{u_o^2} [1-\eta_n] \]

where \( F_1 = \frac{fg}{2f+g} \); \( F_2 = \frac{f^2}{2f+g} \); \( \Phi_n = \frac{\rho_n}{u_o} \); \( \eta_n = \frac{u_{2n}}{u_o} \)

Multiplying by \( \frac{u_o^2}{4\epsilon_0} \) and rewriting one gets

\[ \frac{u_o^2}{4\epsilon_0} m_1 \Phi_n = \gamma_3(\eta_{n+1} + \eta_{n-1} + 2\eta_n) - 2(\gamma_1 + \gamma_3) \Phi_n + \gamma_1(\Phi_{n+1} + \Phi_{n-1}) \]

\[ \frac{u_o^2}{4\epsilon_0} m_2\eta_n = [\gamma_2 + \gamma_4](\eta_{n+1} + \eta_{n-1}) + [\gamma - 2\gamma_4] \Phi_n - \eta_n^3 + \eta_n (1-2\gamma-2\gamma_2+2\gamma_4) \]

where \( \gamma = \frac{f u_o^2}{4\epsilon_0} \); \( \gamma_1 = \frac{f_1 u_o^2}{4\epsilon_0} \); \( \gamma_2 = \frac{f_2 u_o^2}{4\epsilon_0} \); \( \gamma_3 = \frac{F_1 u_o^2}{4\epsilon_0} \); \( \gamma_4 = \frac{F_2 u_o^2}{4\epsilon_0} \)

By defining \( \bar{m} = \frac{m_1}{m_2} \) and \( \sqrt{t} = \left[ \frac{4\epsilon_0}{m_2 u_o^2} \right]^{1/2} t \)
one gets the dimensionless equations

\[
\frac{m}{\hbar^2} \frac{d^2\Phi_n}{dt^2} = \gamma_3(\eta_{n+1} + \eta_{n-1} + 2\eta_n) - 2(\gamma_1 + \gamma_3) \Phi_n + \gamma_1(\Phi_{n+1} + \Phi_{n-1}) \quad \text{...(3.11a)}
\]

\[
\frac{d^2\eta_n}{dt^2} = [\gamma_3 + \gamma_4](\eta_{n+1} + \eta_{n-1}) + \gamma_3^2 \Phi_n - \eta_n^3 + \eta_n(1 - 2\gamma - 2\gamma_2 + 2\gamma_4) \quad \text{...(3.11b)}
\]

These are coupled non-linear differential difference equations in dimensionless form.

(ii) MD simulation

The work involves constant temperature MD procedure to investigate the lattice displacement pattern for the diatomic chain, when the system is in thermal equilibrium. The MD simulation is started with the initial position (displacement) and velocity of the particles in the chain to represent a typical low energy configuration. The time history (time evolution) of each particle after a run is called the sample function for the data [displacement \(u(jh)\) & velocity \(u(jh)\) where \(h\) is the integration time interval and \(j\) is the time step]. The total data consists of an ensemble of sample functions. The zero of time in MD is set after a large number of integrations are performed in order to eliminate initial bias.

\[
\bar{u(jh)} = \frac{1}{N} \sum_{n=0}^{N} u_n(jh) \quad \text{is called ensemble average and}
\]

\[
\langle u(jh) \rangle = \frac{h}{M} \sum_{j=0}^{M} u_n(jh) \quad \text{is called time average}
\]

\(N\) is number of cells and \(M\) is the number of time steps. \(\langle u_n \rangle\) is the ensemble average which is evaluated as the time average over the run. The initial displacements were chosen around -1 for \(\eta\) and around -2 for \(\Phi\) for the double well model potential and these are given below:
\[ \eta = -1 \pm \text{RN} \]
\[ \Phi = -2 \pm \text{RN} \]

RN (random number) values lie between 0.0 and 0.1.

The dimensionless temperature \( \bar{T} \) of the diatomic chain is calculated using

\[ \bar{T} = \frac{1}{2} \left< \bar{m} \Phi^2 + \eta^2 \right> \]

\[ \eta(0) = \bar{T}_0^{\frac{3}{2}}, \quad \Phi(0) = (\bar{T}_0 / \bar{m})^{1/2} \]

...(3-12)

where \( \Phi \) and \( \eta \) are the velocities of the masses \( m_1 \) and \( m_2 \) respectively.

The initial temperature is given by \( \bar{T}(0) = \bar{T}_0 \). The constant temperature simulation can be carried out in either one of the two ways, which are illustrated as below (Fig. 3.4).

![Diagram](image)

**Fig. 3.4 Two different Methods of Maintaining Constant Temperature for a system**
The scaling of velocity will ensure the constant temperature at each MD integration time step [18]. As large number of iterations are to be performed in the first method, we choose the second method to optimise computer time. The velocities are rescaled after each time step according to

\[ \Phi(t) = S(t) \cdot \Phi_n(t) \]

\[ \eta(t) = S(t) \cdot \eta_n(t) \]

where the scale factor \( S(t) \) is evaluated after each time step as follows:

\[ S(t) = \frac{1}{T_0} \left( \frac{1}{2N} \sum_{n=1}^{N-1} \Phi_n^2(t) + \frac{1}{N} \sum_{n=0}^{N-1} \eta_n^2(t) \right)^{1/2} \]  

...(3-13)

This rescales the temperature \( T(t) \) after each time step to the constant temperature \( T_0 \). The constant temperature MD simulation of the non-linear diatomic chain using step size \( h = 0.05 \) for a system of 2000 particles (1000 unit cells) subject to PBC were performed. The zero of time was set after 10000 integration steps in order to eliminate initial bias. MD run for \( 2^{14} \) time steps were carried out and at the end i.e. at \( t = 2^{14} \) time steps, snapshots of the lattice displacement pattern were taken.

(iii) Phase space

In order to study the time evolution dynamics of the non-linear diatomic chain, phase space studies are more suitable. Values of dynamical variables \( (\eta, \dot{\eta}, \Phi, \dot{\Phi}) \) are stored for \( 2^{14} \) time steps for a selected particles unit cells \( (250^{th}, 500^{th} \text{ and } 750^{th}) \).
(iv) MD program description

The programme was developed in C language and executed under Unix environment of Intel 486 machine. It is given in Appendix. A brief description of the programme and the subroutines is given below.

The lattice chain is represented as an array of structure in the programme. Individual structures (represented as array index) called CELL is equivalent of one unit cell. There are 2 such structures bot and top corresponding to previous and current time step. Each structure (CELL) contains the dynamical variables viz. \( q \) - displacement, \( \Phi \) - displacement, \( \eta \) - velocity, \( \phi \) - velocity, \( \eta \) - acceleration, and \( \Phi \) - acceleration. The flow chart is given in figure (3.5).

Details of Subroutines

Initialisation

Because the system is deterministic, the nature of the motion is determined by the initial conditions. The appropriate choice of initial condition is ground state (equilibrium state) configuration. This subroutine initialises displacement around ground state configuration with the help of RN generating routine. As the random number generator provided by C compiler is not effective, we have used different RN routine [39]. Velocities are then initialised as given in the equation 3.12. They are stored in the bottom array of structure (bot CELL). This procedure is repeated for 1000 unit cells using ‘for’ loop.

Verlet subroutine

After initialisation of all particles in the chain is over, main program calls verlet subroutine which determines the time evolution of the system for one time step. This subroutine
together with Shift subroutine will be executed for 10000 times. In order to eliminate initial bias about 10000 time runs are needed. The actual first time step starts only after the 10000th time step.

Using both array of structures, this subprogram calculates dynamical variables for next time step using verlet algorithm and store it in top array of structures.

\[ x_{n+1} = x_n + v_n \Delta t + \frac{1}{2} a_n(\Delta t)^2 \]  
\[ v_{n+1} = v_n + \frac{1}{2} (a_{n+1} + a_n)\Delta t \]  

\((3.14a)\)  
\((3.14b)\)

This verlet integration scheme needs acceleration to find displacement. So this subroutine calls accn subroutine. This procedure is executed for 1000 unit cells.

After calculating new displacement, it again calls accn to find out new acceleration. Using this, it calculates velocity. Following this, calculation of \( S(t) \) is carried out. Then velocities of all the 2000 particles are rescaled. The newly found dynamical variables are stored in top array of structures.

**Accn**

This subroutine calculates acceleration for each unit cell using equations of motion

\[ \frac{1}{m} \frac{d^2 \Phi_n}{dt^2} = \gamma_3 (\eta_{n+1} + \eta_{n-1} + 2\eta_n) - 2(\gamma_1 + \gamma_3) \Phi_n + \gamma_1 (\Phi_{n+1} + \Phi_{n-1}) \]  
\[ \gamma_3 (\eta_{n+1} + \eta_{n-1} + 2\eta_n) - 2(\gamma_1 + \gamma_3) \Phi_n + \gamma_1 (\Phi_{n+1} + \Phi_{n-1}) \]  

\((3.15a)\)

\[ \frac{d^2 \eta_n}{dt^2} = [\gamma_2 + \gamma_4] (\eta_{n+1} + \eta_{n-1}) + \gamma_3 \Phi_n - \eta_n^3 + \eta_n (1-2\gamma_2-2\gamma_4) \]  

\((3.15b)\)
This subroutine is executed for 1000 times. This subroutine is called from verlet subroutine and after the acceleration is calculated for all particles for the current time step and it is stored for the execution of next time step. After the execution is over the control is transferred to verlet or pre-verlet routine. Periodic boundary conditions are implemented in this routine.

**Shift**

This follows verlet subroutine. It overwrites the array bot with the value of the array top. Once the integration scheme for the current time step is over it is treated as previous time step. Using bot value (i.e. previous step), displacements are evaluated for next step (current step) and stored in the top.

**Snapshot**

After $2^{14}$ time steps of verlet execution, the main program transfers the control over to this routine. It stores the values of dynamical variable for 2000 particles on to the given output data file.
Initialise position

Initialise velocity

Next time step

New Position (Verlet)

New Velocity (Verlet)

Calculate $S(t)$ scaling velocity

If $Time=2^{14}$

Snap shot

Stop

Fig. 3.5 MD Simulation Flow chart
#Include <stdio.h>
#include <math.h>
#include <stddef.h>
#include <stdlib.h>
#define MAX 1000
#define defaultfile "md.in"

int NCELL, NTIME, nskip;
double f, f1, f2, m1, m2, eo, Uo, g;
double eta1, eta2, phi1, phi2, To, h;
double m1, gama1, gama2, gama3, gama4;
long seed;
char sfilename[20];
char p1filename[20];
char p2filename[20];
char p3filename[20];
typedef struct {
    float etadisp;
    float etavelo;
    float etaaccn;
    float phidisp;
    float phivelo;
    float phiaccn;
} cell;

cell bot[MAX];
cell top[MAX];

void readdata()
{
    char change;
    FILE *in, *snapfile;

    in = fopen("md.in", "r");
    fscanf(in, "%d\n\n\n\n\n\n", &seed, &NCCELL, &NTIME, &To, &h);
    fscanf(in, "%d\n\n\n\n\n\n", &f, &f1, &f2, &m1, &m2);
    fscanf(in, "%d\n\n\n\n\n\n", &g, &eo, &Uo);
    fscanf(in, "%d\n\n\n\n\n\n", &eta1, &eta2, &phi1, &phi2);
    fscanf(in, "%d\n\n\n\n\n\n", &nskip, &sfilename);
    fscanf(in, "%s\n\n\n\n\n", &p1filename, &p2filename, &p3filename);
    fclose(in);
}
printf("\n Define input data ");
printf("\n ------------------ ");
printf("seed : %ld\n",seed);
printf("NCELL : %g\n",NCELL);
printf("NTIME : %g\n",NTIME);
printf("To : %g\n",To);
printf("h : %g\n",h);
printf("f : %g\n",f);
printf("f1 : %g\n",f1);
printf("f2 : %g\n",f2);
printf("m1 : %g\n",m1);
printf("m2 : %g\n",m2);
printf("g : %g\n",g);
printf("eo : %g\n",eo);
printf("U0 : %g\n",Uo);
printf("etadisp range : (%g,%g)\n",eta1,eta2);
printf("phidisp range : (%g,%g)\n",phi1,phi2);
printf("verlet skip : %d\n",nskip);
printf("snapshot file : %s\n",filename);
printf("p1 file name : %s\n",p1filename);
printf("p2 file name : %s\n",p2filename);
printf("p3 file name : %s\n",p3filename);

printf("\n Press any key to continue : ");
scanf("%c",&change);

snapfile=fopen(filename,"w");
fprintf(snapfile,"\n Defined input data ");
fprintf(snapfile,"\n ------------------\n ");
fprintf(snapfile,"seed : %ld\n",seed);
fprintf(snapfile,"NCELL : %g\n",NCELL);
fprintf(snapfile,"NTIME : %g\n",NTIME);
fprintf(snapfile,"To : %g\n",To);
fprintf(snapfile,"h : %g\n",h);
fprintf(snapfile,"f : %g\n",f);
fprintf(snapfile,"f1 : %g\n",f1);
fprintf(snapfile,"f2 : %g\n",f2);
fprintf(snapfile,"m1 : %g\n",m1);
fprintf(snapfile,"m2 : %g\n",m2);
fprintf(snapfile,"g : %g\n",g);
fprintf(snapfile,"eo : %g\n",eo);
fprintf(snapfile,"U0 : %g\n",Uo);
fprintf(snapfile,"etadisp range : (%g,%g)\n",eta1,eta2);
fprintf(snapfile,"phidisp range : (%g,%g)\n",phi1,phi2);
fprintf(snapfile,"verlet skip : %d\n",nskip);
void accn(cell neus[])
{
    int N, Nminus, Nplus;
    float term1, term2, term3;
    for (N = 0; N <= NCELL - 1; N++)
    {
        Nminus = N - 1;
        Nplus = N + 1;
        if (N == 0) Nminus = NCELL - 1;
        if (N == NCELL - 1) Nplus = 0;

        term1 = (gama2 + gama4) * (neu[Nplus].etadisp + neu[Nminus].etadisp);
        term2 = (1.0 - (2.0 * gama) - (2.0 * gama2) + (2.0 * gama4) * neu[N].etadisp);
        term3 = pow(neu[N].etadisp, 3) + (gama3 * neu[N].phidisp);
        neu[N].etaaccn = term1 + term2 + term3;

        term1 = neu[Nplus].etadisp + neu[Nminus].etadisp + (2.0 * neu[N].etadisp);
        term2 = -2.0 * (gama1 + gama3) * neu[N].phidisp;
        term3 = gama1 * (neu[Nplus].phidisp + neu[Nminus].phidisp);
        neu[N].phiaccn = (1.0 / m) * ((gama3 * term1) + term2 + term3);
    }
}

void initialise()
{
    int N;
    float random(long *idum);
    for (N = 0; N <= NCELL - 1; N++)
    {
        bot[N].etadisp = eta1 + ((eta2 - eta1) * ranom(&seed));
        bot[N].phidisp = phi1 + ((phi2 - phi1) * random(&seed));
        bot[N].etavelo = sqrt(To);
        bot[N].phivelo = sqrt(To / m);
    }

    accn(bot);
}
void verlet()
{
    int N;
    float scale, etavel_2sum=0.0, phivel_2sum=0.0;
    float term1, term2;
    float hby2=h/2.0, hsqby2=(h*h)/2.0;

    for(N=0;N<=NCELL-1;N++)
    {
        term1=bot[N].etadisp+(h*bot[N].etavelo);
        term2=hsqby2*bot[N].etaaccn;
        top[N].etadisp=term1+term2;

        term1=bot[N].phidisp+(h*bot[N].phivel); 
        term2=hsqby2*bot[N].phiaccn;
        top[N].phidisp=term1+term2;
    }

    accn(top);

    for(N=0;N<=NCELL-1;N++)
    {
        term1=bot[N].etavelo;
        term2=hby2*(top[N].etaaccn+bot[N].etaaccn);
        top[N].etavelo=term1+term2;

        term1=bot[N].phivel;
        term2= hby2*(top[N].phiaccn+bot[N].phiaccn);
        top[N].phivel=term1+term2;

        etavel_2sum+=pow(top[N].etavelo,2);
        phivel_2sum+=pow(top[N].phivel,2);
    }

    scale=((m/NCELL)*phivel_2sum)+(etavel_2sum/NCELL);
    scale=sqrt(To)*(1.0/sqrt(0.5*scale));

    for(N=0;N<=NCELL-1;N++)
    {
        top[N].etavelo*=scale;
        top[N].phivel*=scale;
    }
}
void shift()
{
    int N;

    for(N=0;N<=NCBL-1;N++)
        bot[N]=top[N];
}

void snapshot(char name[], cell neu[])
{
    int N;
    FILE *newfile;

    newfile=fopen(name, "a");

    for(N=0;N<=NCBL-1;N++)
        fprintf(newfile, "\n %d %f %f",N,neu[N].etadisp,neu[N].phidisp);

close(newfile);
}

void main()
{
    int N;
    int time1, time2;
    float A,B;
    FILE *p1file;
    FILE *p2file;
    FILE *p3file;
    readdata();

    m=m1/m2;
    A=(g*f)/((2.0*f)+g);
    B=pow(f,2)/((2.0*f)+g);
    gama =f*power(Uo,2))/(4.0*eo);
    gama1 =f1*power(Uo,2))/(4.0*eo);
    gama2 =(f2*power(Uo,2))/(4.0*eo);
    gama3 =(A*power(Uo,2))/(4.0*eo);
    gama4 =(B*power(Uo,2))/(4.0*eo);

    initialise();