PULSEDYN – Documentation

-Rahul Kashyap

(October 8, 2017)

Contents

1. Introduction

PULSEDYN is a software for solving equations of motion of a many-body nonlinear system. This document contains the overview and organization of PULSEDYN. A separate user manual on how to use the code is provided with the code.

2. Directory structure for PULSEDYN

PULSEDYN contains 11 header files and 11 source files. The github distribution for Linux and Windows comes compiled as Code::Blocks project with the executables included. The software is distributed under the GNU General Public License v3. The directory hierarchy is given below.

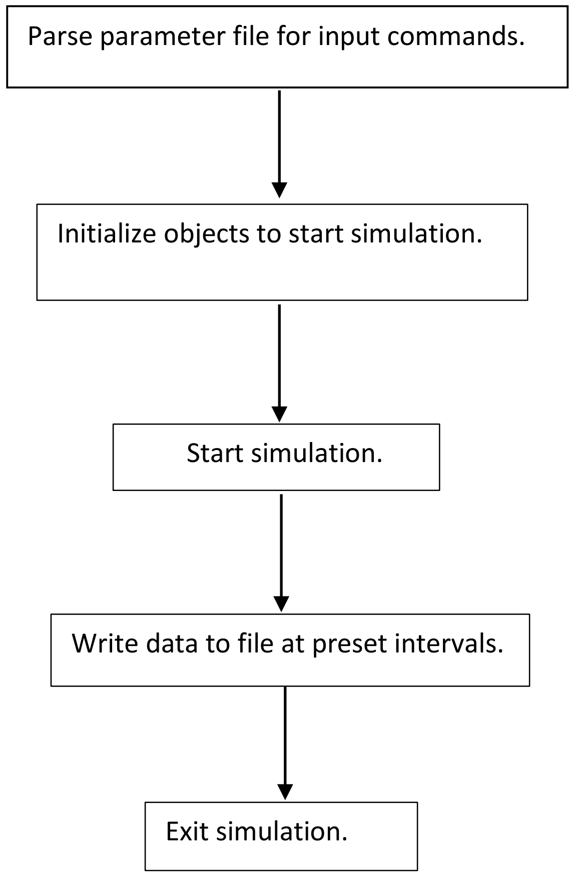
a) /PULSEDYN-***/PULSEDYN – Contains the Code::Blocks project files along with main.cpp. main.cpp contains the parameter file parsing, initialization of the various objects and calling the function to start a new simulation.
b) /PULSEDYN-***/PULSEDYN/src – Contains the source files for PULSEDYN. The following files are contained in the folder.
   i) boundaryConditions.cpp – Contains function for the boundary conditions
   ii) force.cpp – Contains the function for the driving force.
   iii) fpuPotential.cpp – Source file for the Fermi-Pasta-Ulam-Tsingou potential class. Contains the acceleration and potential energy functions for the Fermi-Pasta-Ulam-Tsingou System (cubic + quartic) potential.
   iv) lennardJonesPotential.cpp – Source file for the Lennard-Jones potential class. Contains the acceleration and potential energy functions for the Lennard-Jones potential.
   v) morsePotential.cpp – Source file for the Morse potential class. Contains the acceleration and potential energy functions for the Morse potential.
   vi) Output.cpp – Source file for the Output file class. Contains the functions for file writing to disk.
   vii) Particle.cpp – Source file for the Particle class. Contains member functions for the Particle class.
   viii) Simulation.cpp – Source file for the Simulation class. Since the functions carrying out the simulations have been implemented as template functions, the declarations and definitions of these functions are both present in the header file for this class.
   ix) System.cpp – Source file for the System class.
   x) todaPotential.cpp - Source file for the Toda potential class. Contains the acceleration and potential energy functions for the Toda potential.
c) /PULSEDYN-***/PULSEDYN/include – Contains the header files for PULSEDYN. The following files are contained in the folder.
   i) allIncludes.h – Contains the list various necessary C++ headers as well as the other header files used in the code.
   ii) boundaryConditions.h – Header file for the boundary conditions functions. Contains function definitions.

iii) force.h – Header file for the Force class. Contains class declarations, member declarations and function declarations.
iv) fpuPotential.h - Header file for the FPUT potential class. Contains class declarations, member declarations and function declarations.
v) lennardJonesPotential.h - Header file for the Lennard-Jones potential class. Contains class declarations, member declarations and function declarations.
vi) morsePotential.h - Header file for the Morse potential class. Contains class declarations, member declarations and function declarations.
vii) Output.h – Header file for the Output class. Contains class declarations, member declarations and function declarations.
viii) Particle.h - Header file for the Particle class. Contains class declarations, member declarations and function declarations.
ix) Simulation.h - Header file for the Simulation class. Contains class declarations, member declarations and function declarations. The algorithms for the solvers are implemented as template functions whose complete definitions are also present here.
x) System.h - Header file for the System class. Contains class declarations, member declarations and function declarations.
xi) todaPotential.h - Header file for the Toda potential class. Contains class declarations, member declarations and function declarations.

d) /PULSEDYN-***/PULSEDYN/obj/Debug – Contains the object file for main.o which is the object file for main.cpp
e) /PULSEDYN-***/PULSEDYN/obj/Debug/src – Contains the object files for the source files listed in b). The files included in the folder are
   i) boundaryConditions.o
   ii) force.o
   iii) fpuPotential.o
   iv) lennardJonesPotential.o
   v) morsePotential.o
   vi) Output.o
   vii) Particle.o
   viii) Simulation.o
   ix) System.o
   x) todaPotential.o
f) /PULSEDYN-***/PULSEDYN/bin/Debug – Contains the executable for the software along with a sample parameter file named “parameters.txt”.
g) /PULSEDYN-*** – Contains the user manual and the documentation for the software.

3. High level flowchart for PULSEDYN

A high level flowchart for PULSEDYN is shown below. First, the parameter file is parsed for different parameters. Then the various objects and variables in the software are initialized. The simulation is then started using these settings. At regular intervals, the data is written to file. Previous data files if existing in the same folder are overwritten. Once the iterations are completed, the code writes a restart file containing the position, velocity and acceleration at the last time step and then exits. A list of files written is contained in the accompanying user manual.


Parse parameter file for input commands.
Initialize objects to start simulation.
Start simulation.
Write data to file at preset intervals.
Exit simulation.

4. Build instructions

The software comes pre-built with an accompanying executable name PULSEDYN.exe. The executable can be used as a standalone file to run the software. Copy the executable into a folder of your choice. Write up a parameter file called parameters.txt in the same directory as the executable. If you are working on Windows, double clicking on the PULSEDYN.exe file will start the simulation with the parameters in the parameters.txt file. On Linux, the software can be launched by double clicking on the PULSEDYN Linux executable. However, the terminal window showing the progress of the simulation is not launched on double clicking. A better way of launching the software could be to open the terminal window (ctrl+alt+t) and navigating to your working directory. Then launch PULSEDYN by typing the command ./PULSEDYN in the terminal window. You will now see the software running and the progress is shown in the terminal window.

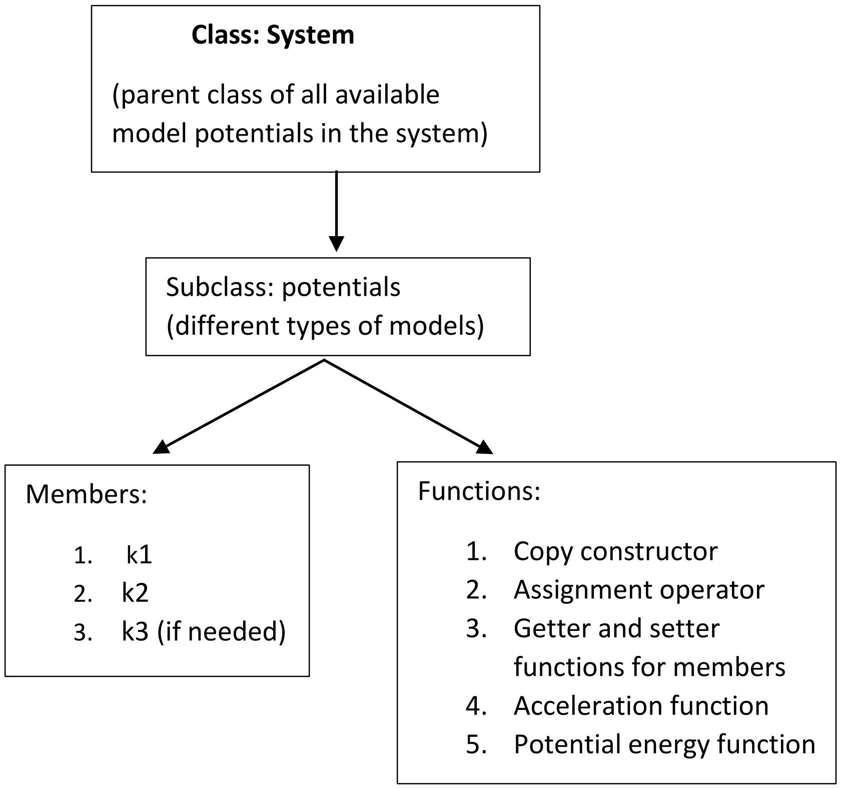
Alternatively, the software can be built by opening the included Code::Blocks project in Code::Blocks and clicking the build button. The software can be run by running the project in Code::Blocks itself, in which case parameters.txt must be included in the folder /PULSEDYN-

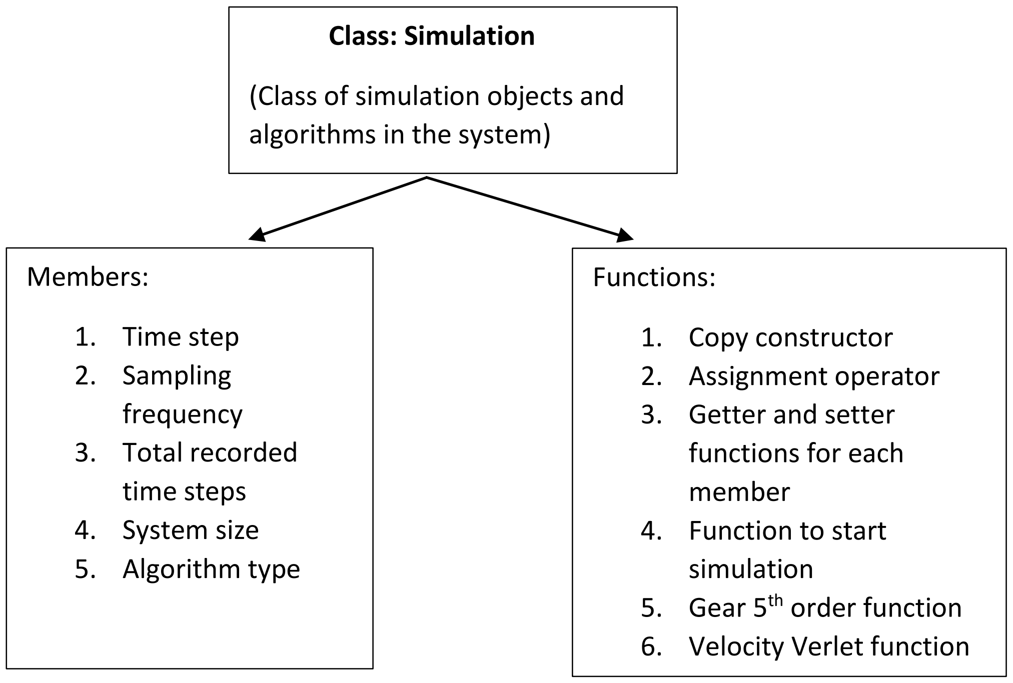
***/PULSEDYN. If any changes are made to any of the source or header files, the PULSEDYN.cbp project will have to be rebuilt. It is necessary to make sure that when rebuilding the project, the older object files and the executable listed in Section 2 previously are deleted first. If compiler optimizations such as –O1, -O2 etc. are turned on to build the code for speed gains, it should be done with the understanding that the error associated with rounding off numbers while performing floating point operations is increased and can be unpredictable. While this is not a major issue for shorter runs (~ 10^5 to 10^6 iterations), it causes significant error build up in simulations involving billions of iterations.

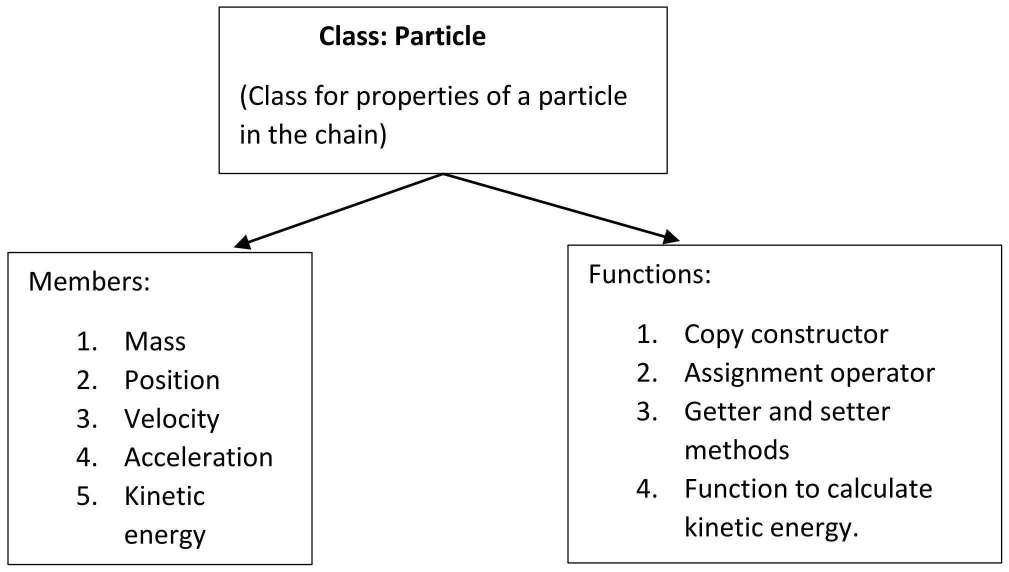
Another source of error in numerical simulations arising out of floating point calculation errors is if the system is not rescaled to a good set of units. In general, values of variables being used can cause errors if they are of disparate orders of magnitude. Such values can cause the code to become unstable during runtime and either exit or print out unphysical results while seemingly running to the end of simulation. Potentials where the bond length explicitly enters the equation of motion can give rise to errors if particles get too close to each other. The forces can increase drastically and the time step specified may not be enough to resolve the time scales properly leading to large errors and numerical instability. It is therefore required that some caution be shown when choosing parameters for the simulations.

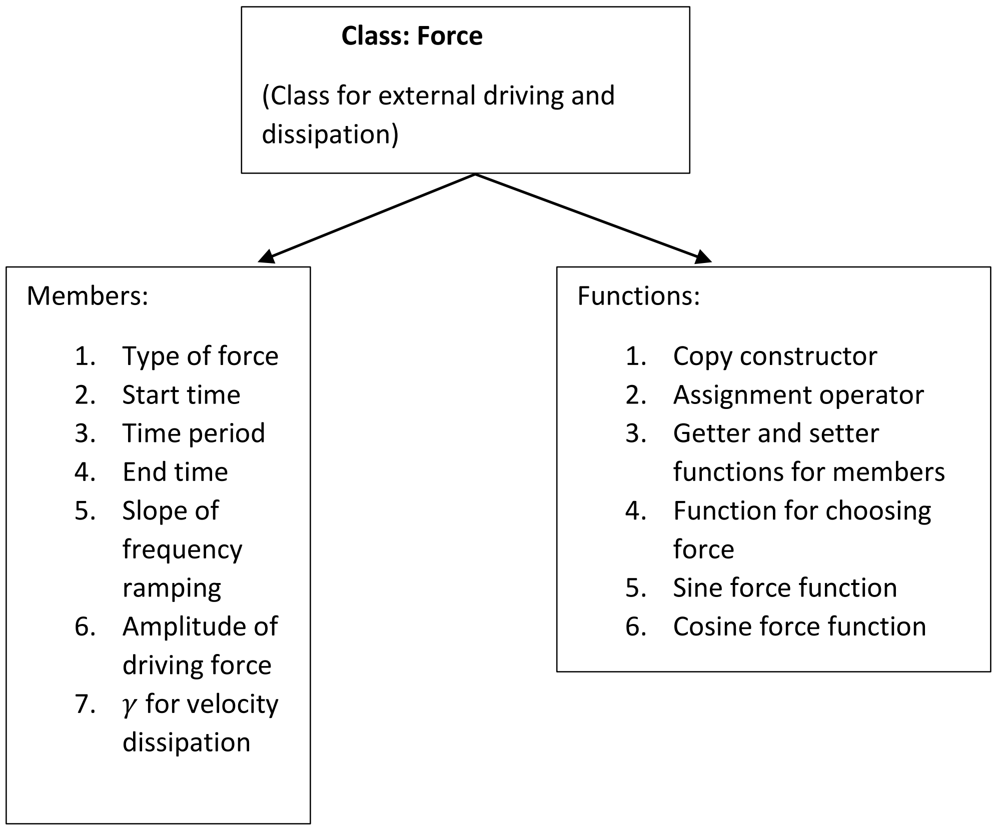
5. Code organization

PULSEDYN is organized as follows. It contains a System class with the potential models, a Particle class with the properties of a particle in the chain, a Simulation class for integration algorithms, a Force class for external driving and dissipation and an Output class to write data to files. It also contains standalone functions for the boundary conditions. The parameter file parsing is done in main.cpp. Some block diagrams of the Class organization are presented here.


Class: System
(parent class of all available model potentials in the system)
Subclass: potentials (different types of models)
Members:
1. k1
2. k2
3. k3 (if needed)
Functions:
1. Copy constructor
2. Assignment operator
3. Getter and setter functions for members
4. Acceleration function
5. Potential energy function


Class: Simulation
(Class of simulation objects and algorithms in the system)
Members:
1. Time step
2. Sampling frequency
3. Total recorded time steps
4. System size
5. Algorithm type
Functions:
1. Copy constructor
2. Assignment operator
3. Getter and setter functions for each member
4. Function to start simulation
5. Gear 5th order function
6. Velocity Verlet function


Class: Particle
(Class for properties of a particle in the chain)
Members:
1. Mass
2. Position
3. Velocity
4. Acceleration
5. Kinetic energy
Functions:
1. Copy constructor
2. Assignment operator
3. Getter and setter methods
4. Function to calculate kinetic energy.



Class: Force

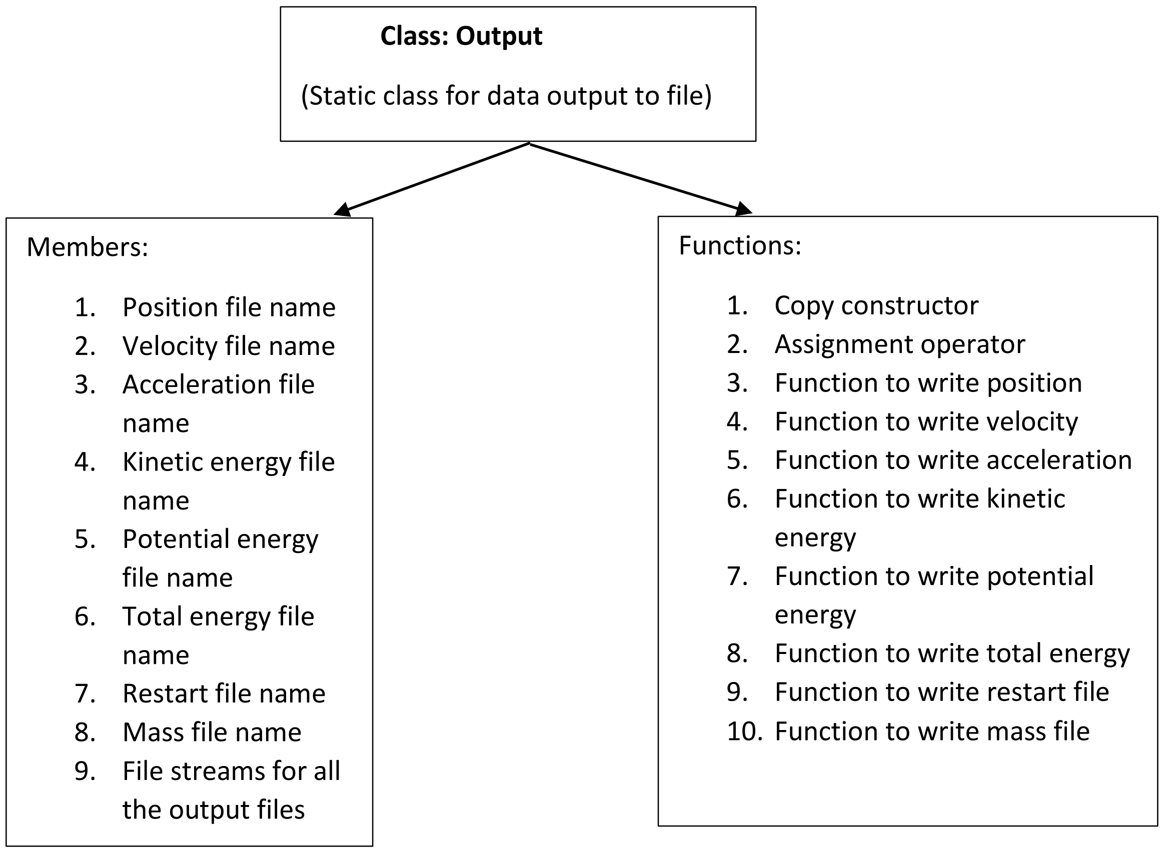
(Class for external driving and dissipation)

Members:

1. Type of force
2. Start time
3. Time period
4. End time
5. Slope of frequency ramping
6. Amplitude of driving force
7. γ for velocity dissipation

Functions:

1. Copy constructor
2. Assignment operator
3. Getter and setter functions for members
4. Function for choosing force
5. Sine force function
6. Cosine force function



Class: Output

(Static class for data output to file)

Members:

1. Position file name
2. Velocity file name
3. Acceleration file name
4. Kinetic energy file name
5. Potential energy file name
6. Total energy file name
7. Restart file name
8. Mass file name
9. File streams for all the output files

Functions:

1. Copy constructor
2. Assignment operator
3. Function to write position
4. Function to write velocity
5. Function to write acceleration
6. Function to write kinetic energy
7. Function to write potential energy
8. Function to write total energy
9. Function to write restart file
10. Function to write mass file

Boundary Condition functions (stand-alone):

1. Left boundary function for acceleration
2. Right boundary function for acceleration
3. Left boundary function for potential energy
4. Right boundary condition for potential energy

A description of each of the important functions in the above mentioned class is given below.

System and Potential Classes

1. Acceleration function – (Name: f_accel)(arguments: Vector of Particle objects)(output: void)

   Calculates and sets the acceleration of each particle based on the chosen potential model. Invokes boundary conditions functions when it does it for end of chain particles.

2. Potential energy function –(Name: f_calcPe) (arguments: vector of Particle objects)(output: total potential energy)

   Calculates and sets the potential energy of each spring in the system based on the chosen potential model. Invokes boundary conditions when it reaches ends of chains.

Simulation Class

1. Simulation starter function – (Name: f_startSim)(arguments: Vector of Particle objects, vector of parameters for Potential, vector of Force parameters)(output: void)

   Chooses which algorithm to use based on input. It is a template function. So the vector of Potential parameters can be chosen at runtime.

2. Gear 5th order solver – (Name: f_gear5)(arguments: vector of Particle objects, vector of Potential parameters, vector of Force parameters)(output: void)

   Function runs the update steps and calculates next iteration in time based on Gear 5th order predictor-corrector method. When the write condition is satisfied, it writes data to file. Can be used both for Hamiltonian and non-Hamiltonian systems.

3. Velocity-Verlet – (Name:f_velocityVerlet)(arguments: vector of Particle objects, vector of Potential parameters, vector of Force parameters)(output: void)

Function runs the update steps and calculates next iteration in time based on Gear 5th order predictor-corrector method. When the write condition is satisfied, it writes data to file. Can be used only for Hamiltonian systems. Run time is faster than Gear 5th order algorithm, but cannot be used with driven or dissipative systems. If supplied force and dissipation, the function ignores then non-Hamiltonian part.

Force Class

1. Force choose function – (Name: f_forceCalc)(arguments: vector of Particle objects, current time)(output: void)

   Chooses which driving force to apply – sine or cosine.

2. Sine force – (Name: f_sine)(arguments: vector of Particle objects, current time)(output: force)

   Returns the sine force.

3. Sine force – (Name: f_cosine)(arguments: vector of Particle objects, current time)(output: force)

   Returns the cosine force.

Output Class

1. Write functions – (Name: writeTo****File)(arguments: value to be written, endline check)(output: void)

   Writes data to file for the different variables. If endline is yes then goes to next line else it leaves a space and writes value.

Stand-Alone Functions

1. Left boundary for acceleration – (Name: f_leftBoundaryConditionsAccel)(arguments: vector of Particle objects)(output: value of dx for the left most particle to be used in the acceleration function)
2. Right boundary for acceleration (Same as above except for rightmost particle)
3. Left boundary for potential energy – (Name: f_leftBoundaryConditionsPe)(arguments: vector of Particle objects)(output: value of dx for left most particle to be used in the potential energy function)
4. Right boundary for potential energy (Same as above except for rightmost particle)

Notes: