Requirement list

Introduction

Years ago, solving differential equations was restricted to the simplest ones. Actually, the only ones with an exact solution were the two body problem and the harmonic oscillator. Fortunately, the computational power has risen since then. This program is intended to solve a N-body problem, with N much bigger than 2!, which is impressive. The initial idea is to save the positions of every particle, for each of the time steps (whose definition depends on a parameter of the input file), in order to make a visual animation of the atoms moving with time (and later, to be able to extract physical properties from it, which will be a very interesting application of the program). Although it may seem that the program is oriented to a professional audience (i.e., physical graduates, and maybe some chemical graduates), its is not restricted to them only. The GUI will be able to minimize the interaction of the user with the program. It will be restricted to filling in the boxes of the GUI with the input parameters. The program will be designed to return a *.axsf* file format, prepared to be provided to the *XcrySDen* software, in order to visualize the animation.

Moreover, the program will be even simpler with the creation of an executable (.exe in Windows and a regular executable in Linux), which leaves the execution of the program to a double click in a file!. For more experienced users, there will be the option to execute the program through command-line (with the option to deploy the GUI or read a data file from a provided path). This users may give the program a more productive use, being able to study properties of materials such as the melting point of this material.

The program will compute the evolution of a monoatomic system using a *Lennard-Jones* potential and by solving Newton's second law (using Verlet algorithms to improve performance in simpler computers). The initial positions will be the ones of an FCC lattice (initially, BCC and SC support will be provided in the future) and velocities will be random, following a Maxwell-Boltzmann distribution (more distributions will be included once the program is finished).

The data that will be required by the program are: the two parameters of the Lennard-Jones potential (sigma and epsilon), the temperature, the cutoff distance, from which the potential will be 0 in order to reduce computational requirements; the cutoff neighbour distance, the maximum distance used to compute the neighbour list; the total simulation iterations and the time step of each one; and finally the number of unit cells in a given direction, used to compute the initial conditions (since velocities are random, more than one unit cell is required). For simplicity, the number of cells will be the same in each direction, so the dimension of the supercell created will be N^3, being N the number of unit cells in each direction.

When the program ends, apart from the *.axsf* file with all the positions, control files will be returned. This includes the initial positions of the atoms (which does not depend in any random quantity), the first neighbour list, initial velocities packaged in a histogram, energies for every time step (to check energy conservation) and a plot with the energy evolution with time.

REQUIREMENT LIST

The requirements which identification code (ID) has not ‘.a’ are guaranteed to be implemented and those which ID ends with ‘.a’ are higher level algorithms that may be implemented but are not guaranteed to be. The functionalities of the code are grouped in blocks, in order to improve readability.

1 INPUT/OUTPUT:

- 1.1 The program will read the data from an input text file.
- 1.2 The program will read the parameters of the input file independently on the order.
- 1.3 The program will distinguish different system of units of the parameters given.
- 1.4 The program will execute the code with default values if the path to the text file is not correct.
- 1.5.a If there is not given a text file, the program should display a GUI to write the data.
- 1.6.a The code will have some error checking implemented, giving hints about the source of the error to help the user.
- 1.7.a An executable of the file will be created, in order to simplify its execution.

2 INITIALISATION:

- 2.1 The program will compute the initial positions according to a FCC lattice.
- 2.2 The program will compute the initial velocities according to a Maxwell-Boltzmann distribution.
- 2.3 The program will print the initial positions in a file.
- 2.4 The program will print the initial velocities in a histogram.
- 2.5.a The program will display a set of kinds of unit cells to choose between them.
- 2.6.a The program will display a set of distributions to choose between them.
- 2.7.a The program will execute with custom initial positions.
- 2.8.a The program will execute with custom distributions.

3 NEIGHBOUR LIST:

- 3.1 The program will compute Verlet’s neighbour list
- 3.2 The program will print the initial neighbour list in a file

4 POTENTIAL, ENERGIES, AND FORCES:

- 4.1 The program will implement the double shifted Lennard-Jones potential.
- 4.2 The program will print energies (potential, kinetic, total).
- 4.3.a The program will have an option to switch the number of shifts of the Lennard-Jones potential.
- 4.4.a The program will display a set of potentials to choose between them, different to the Lennard-Jones one.
- 4.5.a The program will have an option to use custom 2-body potential, which formula will be inputed.

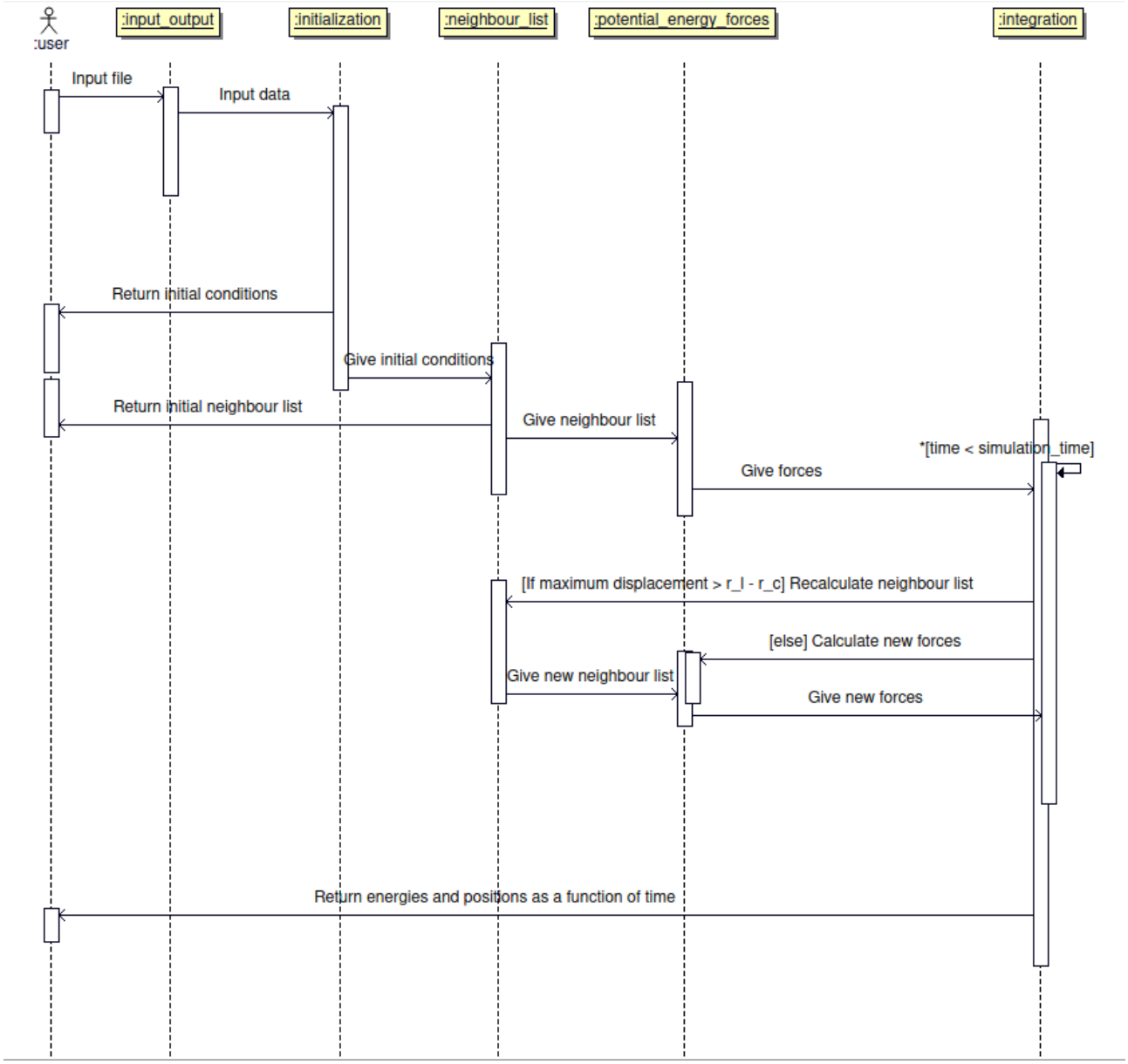
5 INTEGRATION OF NEWTON'S 2nd LAW:

- The program will integrate the Newton equation using Velocity-Verlet algorithm.
- The program will print positions in a text file, for every time step.
- 5.3.a The program will return a *.axsf* format file, to visualize the movement of the atoms.
- 5.4.a The program will display a set of integration algorithms to choose between them.
- 5.5.a The program will have an option to custom the integration algorithm.

6 POST-PROCESSING

- 6.1.a The program will have an option to custom a paired distribution function.
- 6.2.a The program will return physical parameters of the given crystal.

SEQUENCE CHART AND USE CASE DIAGRAM



This image above shows the sequence chart of an execution of the code. As it was said earlier, the code is intended to minimize the interaction between the user and the program. Once the user gives the input, there is no more interaction with the code (as it can be seen in the use case diagram placed below), apart from the use of the output files. The sequence chart is quite self-explanatory. With the data given in the input file, initial positions/velocities are generated (and returned in a file). Once the positions are generated, they are used to create the neighbour list (which is also returned) and, with the neighbour list, the initial accelerations are generated. Finally, a loop with as many time steps as provided in a parameter of the input, propagates the positions. The new positions are used to calculate the displacement and, depending on the value of it, the neighbour list is updated or not. This neighbour list is used to compute the new accelerations, used to calculate new positions and so on.