Jessica Au, Monica Vivar
MAT 128C
Arsuaga
June 2, 2017

PROGRAMMING PROJECT 2

Introduction
In this project, we will be implementing a molecular dynamic (MD) simulation of Argon atoms in two dimensions. As these are generally stable atoms by nature, we will only consider Van der Waals interactions within this system. As discussed in class, for a MD simulation, the evolution of the system is given by Newton's force equation which is as follows:

$$\vec{F} = m\vec{a}$$
$$-\frac{dE}{dx} = m\frac{d^2x}{dt^2}$$

To integrate this problem, we will be using Verlet's algorithm. Before describing Verlet's algorithm, we will be introducing the Van der Waal's interactions in a system of Argon atoms and reduced units. First, we have that the energy of two atoms at a distance of $r_{i,j}$ of such a system is given by

$$E(r_{i,j}) = 4\epsilon((\frac{A}{r_{i,j}})^{12} - (\frac{B}{r_{i,j}})^6)$$

where in this case, we have $A = B = \sigma$. σ and ϵ determine the scale of this problem, where σ denotes the Van der Waals radius and ϵ denotes the magnitude of the energy and $\frac{\epsilon}{K_B} = 120K$. In reduced units, we have $\sigma = \epsilon = 1$. $r_{i,j}$ is the absolute value of the distance vector between two atoms i and j. In order to solve Newton's equation, we need to compute te derivative of the Energy equation with respect to the derivative. Therefore, we find the derivative, in reduced units to take the form

$$f_{i,j} = 24((\frac{2}{r_{ij}^{13}})) - (\frac{1}{r_{ij}^7}))$$

Implemention of boundary conditions
Next we must set boundary conditions to avoid unwanted boundary effects on the dynamics of the system. We begin with a square region of the plane with dimensions $[0, 2L\sigma] \times [0, 2L\sigma]$ and identify the top, bottom, left and right edges. We want the following to happen at our boundaries

- The distance between any two atoms will have to be computed across the domain and across the edges we identified, *i.e top, bottom, etc.*
- The trajectory of any particle moving beyond the boundaries will continue across the identified edges.

Figure 1 below gives the reader a visual regarding the criteria above for boundary conditions.

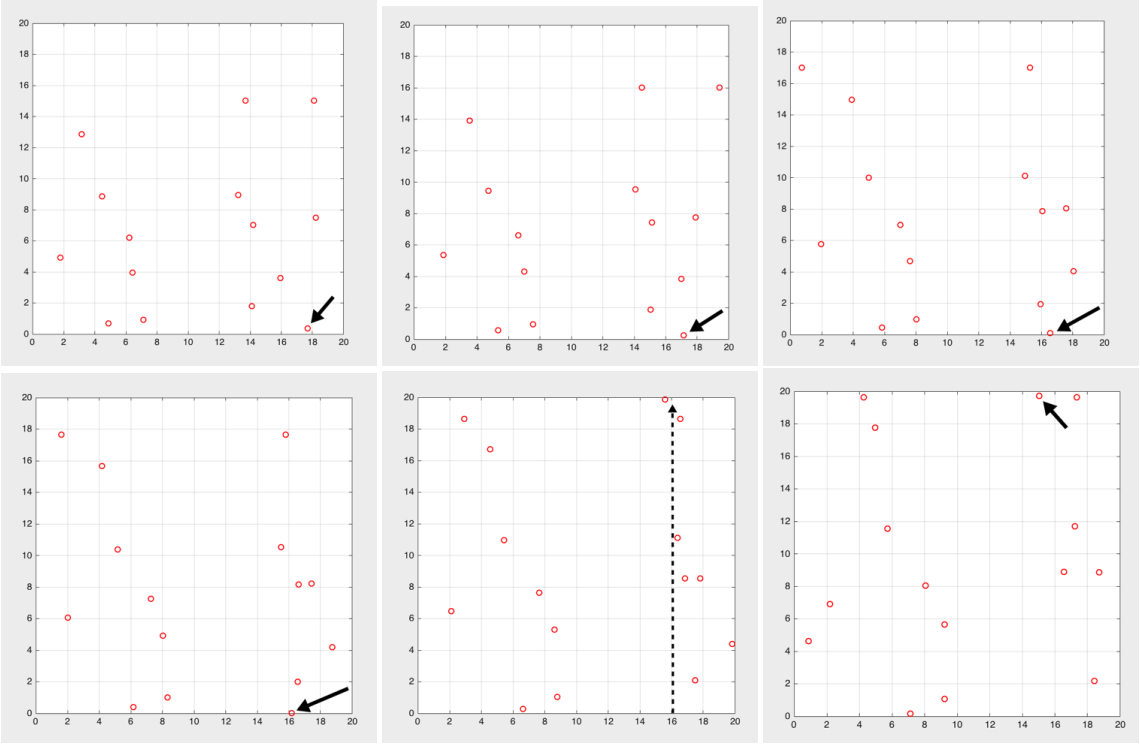

Figure 1: This is a visualization of our boundary conditions. It is supposed to act like a torus, and by following the arrows, the reader will be able to see what happens to an atom as it travels across one of the identified edges. The black arrow indicates which atom to focus on, and the dashed arrow shows where it appears after the boundary conditions are applied.

Initialization of our system

We then begin with the initialization of the system. For N atoms, we have a size L system with edges from 0 to $2 \times L$. We will be fixing a time step, δt, and setting our initial positions to be within this system at least σ apart, which we set to be equal to 1, and initial velocities to be 1, which are then displaced. To use Verlet's algorithm, it is also required to to assign positions at t = -1 where

$$x_{-1} = x_0(i) - v_{x0}(i)\delta t$$

See **Figure 2** on for more on specifically how we decided to intialize our variables.

```
% inputs
N = 15; % number of atoms
L = 10; % size of system

%declare vectors
v = zeros(N,2); % velocity vector
p = zeros(N,2); % position at n
p1 = zeros(N,2); % position at n-1
pn = zeros(N,2); % position at n+1

r = zeros(N, 2); % distance vector
f = zeros(N, 2); % force vector

% Initialization of acceleration vectors
ax = zeros(N,2); % for x component
ay = zeros(N,2); % for y component
sx = zeros(N,2); % dummy variable meant for sum of x component
sy = zeros(N,2); % dummy variable meant for sum of y component

%% initialization of position
% choose N random integers into a N,1 vector
xp = randi([1 2*L], N, 1);
yp = randi([1 2*L], N, 1);

% With sigma = 1, we have deltar = sigma/4
deltar = 0.25;

% displace intial random integers
% addition to position vector
%   1:N are the x components
%   N+1:2*N are the y components

p(1:N) = 2.*(xp-0.5).*deltar;
p(N+1:N*2) = 2.*(yp-0.5).*deltar;

%% initialization of velocity
v0 = 1; % same as 'Computational Physics, Giordano'

% displacement of velocities and randomized for each atom
v(1:N) = abs(2.*(p(1:N)-0.5).*v0);
v(N+1:N*2) = abs(2.*(p(N+1:2*N)-0.5).*v0);
```

```
% initialization of time variables
dt = .02; % change in time, time step
time = 100; % end time
tmax = time/dt; % simulation time

% Initialization of positions at time = n-1
% Required for verlet's algorithm
p1(1:N) = abs(p(1:N)-(v(1:N)*dt));
p1(N+1:N*2) = abs(p(N+1:N*2)-(v(N+1:N*2)*dt));
```

Figure 2: This is taken from our Matlab script, *dynamic.m* which shows how we initialized our variables and displaced them before we used them in Verlet's algorithm, which is shown in **figure 7**.

Implementation of Verlet's algorithm

For the Verlet's algorithm, we need to update x and y components separately. The following equations compute the new position, new velocity, and acceleration for each component. For simplicity sake, we will only be writing the equations for the x component.

The new position of x is given by

$$x_i(n+1) \approx 2x_i(n) - x_i(n-1) + a_{ix}(n)(\Delta t^2)$$

The new x component of velocity is given by

$$v_{ix}(n) \approx \frac{x_i(n+1) - x_i(n-1)}{2\Delta t}$$

The acceleration a_{ix} is given by

$$a_{ix} \sum_{i \neq j} f_{ij} \cos(\theta_{ij})$$

where the value of the force is given in the introduction and we can calculate $\cos\theta_{ij} = \frac{\Delta x}{r_{ij}}$, with Δ representing the separation between atoms i, j along the x coordinate. We calculate this before updating the position and velocity values. To reduce the computational time, we also assumed that atoms with a distance greater than 3σ will not interact with each other and that the forces $|f_{ij}|$ and $|f(ji)|$ are equal. See **Figure 6** on the last page for implementation.

Data collection and results We are asked to calculate an average of measurable quantities along the trajectory of a particle and across particles. By subdividing the time intervals, we can take averages of our results for each time interval across atoms.

We can estimate the termperature T t which the system is running. One approach we can use to estimate T is by using the equipartition theorem that tells us

$$K_B T = \frac{m}{2}\langle (v_x^2 + v_y^2)$$

where $\langle , \rangle$ represents the mean taken over the trajectories and particles. We found the mass of each Argon atom to be 9.96×10^{-25} kg and the Boltzmann constant to be 1.38^{-23}. The following figures represent our results

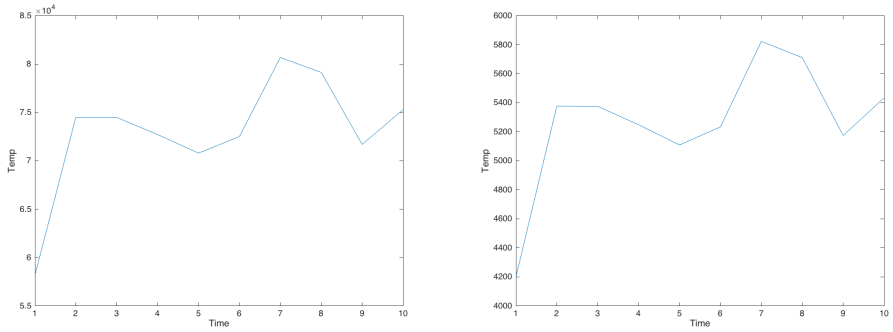

Figure 3: These graphs show the average temperature change over time for our program (*right*), dynamics.m, as well the program that was used in *"Computational Physics"* by Giordano and Nakanishi *(left)*. Although the magnitude for our values were different, the behavior is very similar. Possibilities concerning the large values will be discussed below in **Issues and Possible Remedies**. We see the temperature grow exponentially between $t = 1$ and $t = 2$ and continues to fluctuate between a certain range. From our simulation, we found this range to be 7×10^4 and 8×10^4.

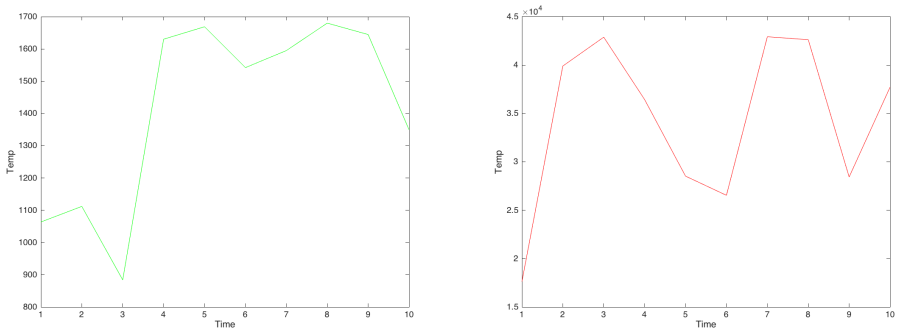

Figure 4: Here, we used the same temperature equation and observe how the behavior changes as we change the length. On the left is when we have the length to be 5, and on the right we have the length of 20. The main difference between these two graphs, would be the range of fluctuation after $t = 3$. For the smaller length, we see more erratic behavior, while the larger length shows a similar behavior to **Figure 3**.

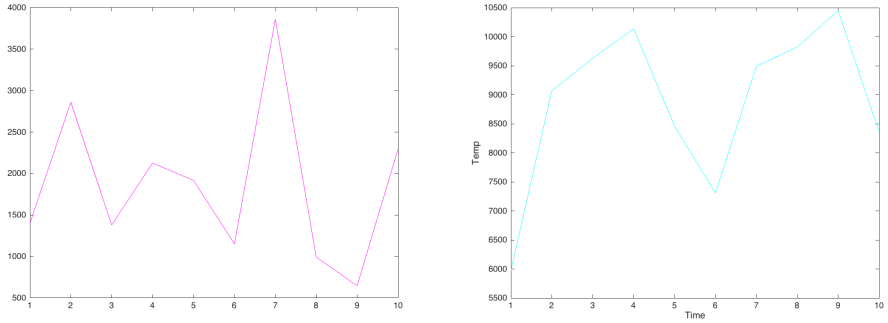

Figure 5: Here, we changed the amount of atoms from N = 15 to N = 5 (on the *left*) and N=25 (*on the right*). We see very different behaviors for both of these cases. For the N = 5, the behavior does not seem stable and fluctuates very much. For N = 25, we do see an initial growth exponentially, but the fluctuation ranges seems to be larger compared to the other graphs above.

Issues and possible remedies

There were a couple issues that we faced during the implementation of this simulation. (1) We had issues with the randomization of initial points and their displacement. In some cases, our initial positions would be found to be NaN. Although this did not happen frequently, this was an issue that we tried our best to resolve by observing the distribution of the initial positions and how they displaced. As a possible solution, we could have tried to implement a seed in which a working set of initial positions would be set every single run. However, it didn't really seem random and not a permanent solution if we wanted to see all possible behaviors of the molecular dynamics. (2) The other issue was the instability of our system after a long period of time. It seemed that the force of the atoms grew exponentially which caused the acceleration of the atoms to grow exponentially as well. As it was commented in our matlab script, *dynamics.m*, as a possible remedy, we chose to make the force between atoms constant if they were within a certain distance of each other. The reasoning behind this was that by observing the force equation

$$f_{ij} = 24(\frac{2}{r_{ij}^{13}} - \frac{1}{r_{ij}^{7}})$$

when the distance r_{ij} is about 1, we find that this force becomes equal to 24. As this distance gets smaller and smaller, the force would grow very large leading to the system blowing up. Due to the fact that we

could not fully enforce the criteria where the atoms needed to be at least σ from each other, this was one of our solutions. By using a multiple of 24 as a constant and dividing this number by the small distance $r_{ij} < 1.1$, we were able to obtain a smaller system which delayed the eventual blow up. This value was taken from the reading in *"Computational Physics"* by Giordano and Nakanishi. **Figure 6** below are some screenshots taken at different times as an attempt to show our simulation.

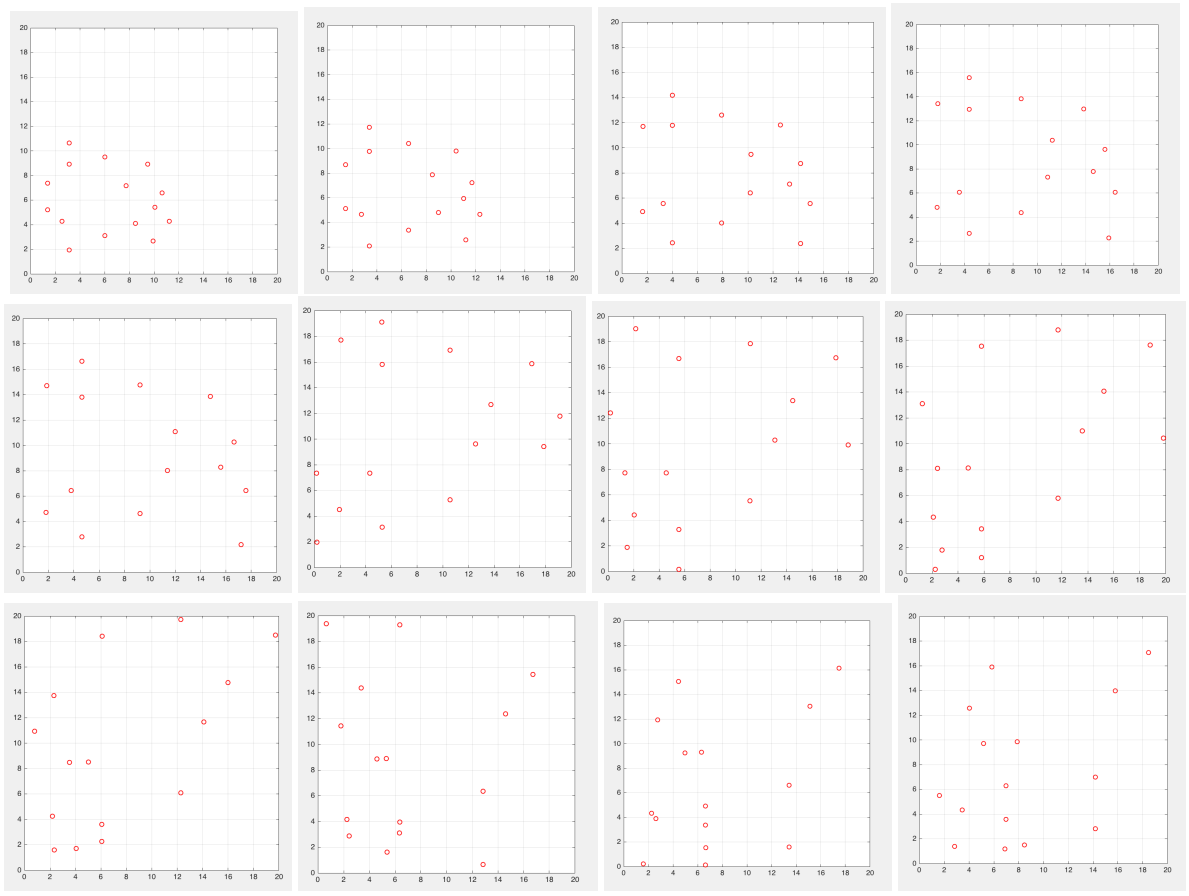

Figure 6: This is a demo displaying screenshots taken from our simulation in increments of $t = 2$. The panels go from left to right, for each row.

```
%verlet algorithm to solve system
for t = 1:tmax

    % find distances for all atoms and their forces
    for i = 1:N
        for j = 1:N
            r(i,j) = (sqrt((p(j)-p(i))^2 + (p(j+N)-p(i+N))^2));

            if i ~= j && r(i,j) < 3.0 && r(i,j) > 1.1
                f(i,j) = abs(24*((2/(r(i,j)^13))-(1/r(i,j)^7)));

            % To strengthen distance criteria of less than 1; explained in write up. (2)
            elseif i ~=j && r(i,j) > 0.0 && r(i,j) <= 1.1
                f(i,j) = (168)/r(i,j); %168 is 24*7
            else
                f(i,j) = 0;
            end
        end
    end


    %% find acceleration of both components
    % for loop to find the force*cos(...), and force*sin(...)
    for i = 1:N
        for j = 1:N
            dx = (p(j)-p(i)); % distance between x components
            dy = (p(j+N)-p(i+N)); % distance between y components

            if j ~= i
                sx(i,j) = f(i,j)*(dx/(r(i,j)));
                sy(i,j) = f(i,j)*(dy/(r(i,j)));
            else
                sx(i,j) = 0;
                sy(i,j) = 0;
            end
        end
    end

    % Acceleration is the sum of what was found in for loop above
    ax = sum(sx);
    ay = sum(sy);
```

```
    % Enforcing boundary conditions
    for i = 1:N
        for j = 1:2
            if (pn(i,j) > L);
                pn(i,j) = pn(i,j) - 2*L;
            end
            if (pn(i,j) < 0.0);
                pn(i,j) = pn(i,j) + 2*L;
            end
        end
    end

    % Update velocity vector
    v(1:N) = abs(pn(1:N)-p1(1:N)/(2*dt));
    v(N+1:2*N) = abs(pn(N+1:2*N)-p1(N+1:2*N)/(2*dt));

    %% Update old position vectors
    % Position vector at t-1
    p1(1:N) = p(1:N);
    p1(N+1:2*N) = p(N+1:2*N);

    % position vector at t
    p(1:N) = pn(1:N);
    p(N+1:2*N) = pn(N+1:2*N);

    %% Calculating the temperature of the system given the equipartition theorem.
    % molecular mass of argon: 39.948 g/mol, convert to kg.
    B = 1.38*10^-23; % Boltzmann Constant
    T= (((9.96e-25)/2)*mean(v(1:N).^2+v(N+1:2*N).^2))/(B);

    % Simulation
    plot(p(1:N),p(N+1:N*2), 'ro');
    grid;
    axis([0 2*L 0 2*L]);
    axis square;
    pause(0.1);

end
```

Figure 7: This is taken from our Matlab script, *dynamic.m* which shows our implementation for Verlet's Algorithm, as well as the calculation for the temperature and show we generated our simulation.