5CCP211C-Introduction to numerical modelling,
Physics Dept., King's College London

# Planetary Motion

Akshat Chandel; Shafkat Mehmood; Taimur Ahmad Khan

November 29, 2018

Date performed ..... November 27, 2018
Group ..... 2

## 1 Introduction

In this report classical Molecular Dynamics is used to model planetary model in accordance with the laws of Newton and Kepler. Classical Molecular Dynamics is a technique by which the atomic trajectories of a system of N particles can be generated by numerical integration of Newton's Equations of Motion, for a specific interatomic potential, with certain initial conditions (IC) and boundary conditions (BC).

Newton's equations which are used to model the planetary system are given below:-

$$F(r) = \frac{GMm}{r^2} \tag{1}$$

$$E(r) = -\frac{GMm}{r} \tag{2}$$

$$U(r) = -\frac{GM}{r} \tag{3}$$

Kepler's Law for time is given below:-

$$T^2 = \frac{4\pi^2}{GM}a^3 \tag{4}$$

In the above equations G is the gravitational constant, M is the mass of the sun, m is the mass of an individual planet, r is the distance of an individual planet from the sun, E is gravitational potential energy and F is force.

There are 3 different algorithms used to find the trajectory for Classical Molecular Dynamics Euler's algorithm, Verlet and velocity Verlet.

- Euler's Algorithm: Euler's methods that only the first term of a Taylor expansion is required to approximate a function, so $y(x) = y(x_0 + h) = y(x_0) + hy'(x_0)$ and $y_{k+1}(x) = y(x_0 + h) = y_k(x_0) + h_k y'(x_k)$ for the k+1th term. So for example if y(x) was a function for velocity we could simply find the difference between the Taylor expansions to find the change in velocity.
- Verlet Algorithm: A taylor expansion of the position fuction of the particle is taken about both $t + dt$ and $t - dt$ . The velocity of the particle is given by the difference between these two Taylor Expansions which gives $v_i = \frac{r_i(t+dt)-r_i(t-dt)}{2dt} + order(dt^2)$
- Velocity Verlet: In Velocity Verlet the position is used to calculate the the new velocity and the new velocity is used to calculate the new acceleration. To do this, again, Taylor expansions are used $r_i(t + dt) = r + i(t) + v_i dt$ for position and $v_i(t + dt) = v_i(t) + \frac{f_i(t)+f_i(t+dt)}{2m_i} dt$. The acceleration can then be calculated from difference in speed.

An essential quantity which needs to be considered when applying the three aforementioned algorithms is time. The time step needs to be chosen such that a compromise is reached between efficiency and speed. A time increment which is too small will lead to a high degree of inefficiency whereas a time increment too large will lead to inaccurate solutions where energy conservation and reversibility are affected. A general overview of Classical Molecular Dynamics is given below:-

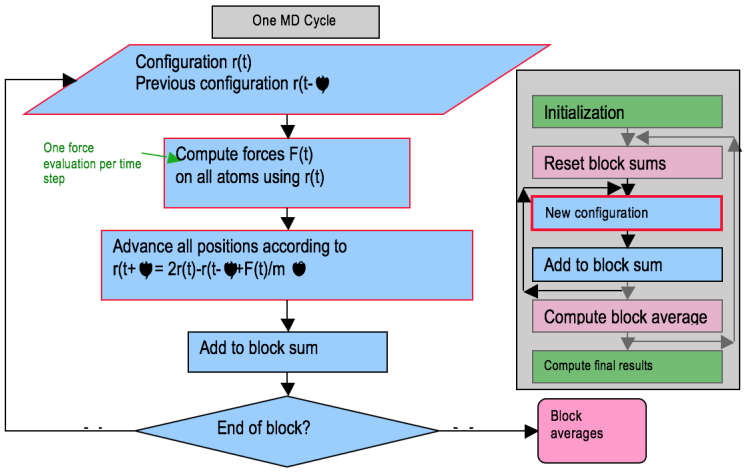


Figure 1: A flowchart demonstrating a Classical Molecular Dynamics Alogrithm

In this specific coding assignment, classical molecular dynamics is used to theorize large mass structure motions e.g Planetary Motion. Newton's equations are integrated using the Verlet Algorithm. Using the turtle function which produces a two dimensional animation representing motion, a system is created with a large mass Sun and four orbiting planets

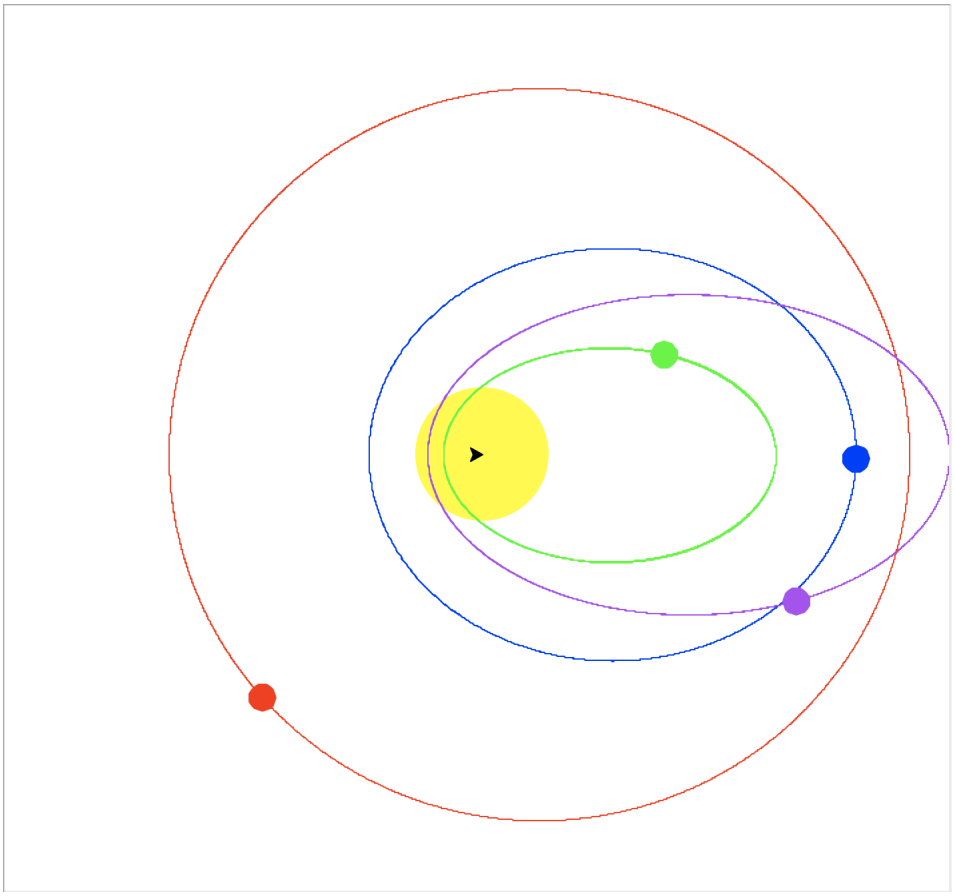

Figure 2: The animation at G=1, dt=0.1

of different colors. The force from the Sun affects the motion and orbit of the smaller mass planets. This force is dependent on G which is set in the reduced function. Note that in this case, the gravitational force from different planets does not affect the motion and orbits. Figure 2 given below is an example of the system in the animation. Both elliptical and circular orbits can be seen.

## 2 Structure of Code

A python code was written in order to simulate this process using classical molecular dynamics. The code is begun by importing modules required for mathematical computation and graph plotting such as numpy and matplotlib. Initially two key constants dt, the time increment and the value of G are defined to initialize the system and the cumulative values of time are stored in a list in order to plot the graph. Then lists are defined to store various values of the planets such as the potential, force ,radius and time period for each planet.

## 2.1 Planets

In order to define characteristics of the planets themselves a class is created called 'Planets' and for each planet such as the name, radius, mass, distance from sun and colour. A class is also defined for characteristics of the sun such as size and initial location. In this system the turtle module is introduced and initialized. This is how the solar system is illustrated as this module used as it acts as a drawing board with a 'turtle' used to illustrate figures on this board.

## 2.2 Solar System

Another class is defined for the whole system creating a list in which to store planets. Next a function is defined which mathematically computes how the system changes in accordance with the chosen algorithm, so the future position, acceleration, distance are found. For example the change in position on the x axis and y axis is given by the following code:-

```
rx = self.star.getX() - p.getX()
ry = self.star.getY() - p.getY()
r = math.sqrt(rx ** 2 + ry ** 2)

p.accX = G * self.star.getMass() * rx / r ** 3
p.accY = G * self.star.getMass() * ry / r ** 3

p.movePos(p.getX() + p.vX*dt + (p.accX*dt**2)/2
, p.getY() + p.vY*dt + (p.accY*dt**2)/2)
```

getx() returns the current x value of the planet's position as defined in the previous class. The change in position is dervived from classical equation of motion and coded into Verlet algorithm. Using classical mechanical values, the position of the planet is changed dependant on the force from the Sun (G) present and previous accelerations and positions. This entire code is repeated using new values of R, to find new acceleration which in turn with all values calculated is used to calculate velocity of the planet. The values for Energy, Potential, Force after being found using Newtonian equations are added to the empty lists initialized in the start of the code. This is demonstrated by the code below:-

```
V = -G * self.star.getMass() / r**2
potential_planets.append(V)
E = -G * self.star.getMass() * p.mass / r
potenergy_planets.append(E)
F = G * self.star.getMass() * p.mass / r**2
force_planets.append(F)
```

The Euler method is also included but is commented out and can be used to compute the system when the user wants.

## 2.3 Initiate Galaxy

Next, a function called initiate galaxy is created where characteristics of each planet such as size and initial position is defined. The time period is calculated using the ratio equation

derived from Kepler's Laws of Planetary Motion which is written in the introduction. The derived form used is;

$$t^2 = c^3 \tag{5}$$

Here t is the time period and c is the radial semi major axis of revolution. Finally for each planet the semi minor and semi major axis are defined as the smallest distance from the sun and the largest distance from the sun respectively and these values are printed. Note that in the code below these values found are corresponding only to the first Planet.

```
Pmax1 = max(radius_orbit[0::4])
Pmin1 = min(radius_orbit[0::4])
print 'semi major axis for green planet 1 is', Pmax1
print 'semi minor axis is green planet 1 is', Pmin1
c= 0.5*(Pmax1+Pmin1)
t = c**1.5
timeperiod_planets.append(t)
```

When these values are printed they are printed in groups of four. So for example for each time step four values will be calculated, one for each planet (A1,B1,C1,D1), in the next time step four more values will be outputted (A2,B2,C2,D2) and so on. The method by which to separate these values for each planet i.e to have groups of values for the same planet in lists: (A1,A2,A3,A4,...), (B1,B2,B3,B4,...) for each of the four planets, is done by the following line of code;

```
radius_orbit[0::4]
radius_orbit[1::4]
```

The line of code above; [0::4] splits the groupings of four into lists that have the first value of each grouping or the values regarding Planet 1. We take the zeroth term for the first planet as in python, counting starts from 0. This is repeated for all the other planets where the nth planet with need the nth term of each grouping i.e [n::4].

# 3 Discussion of Data

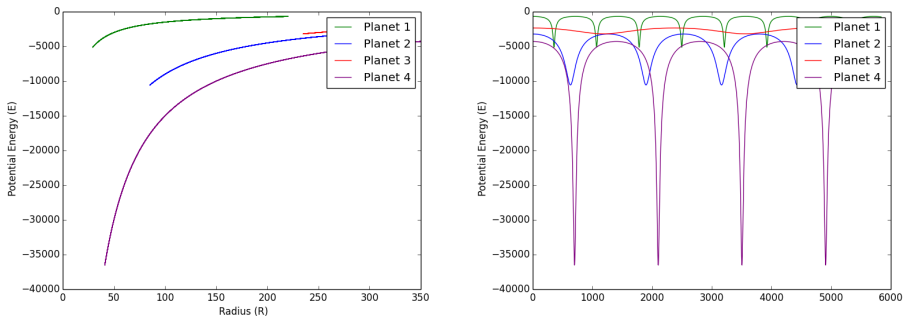


Figure 3: Energy change

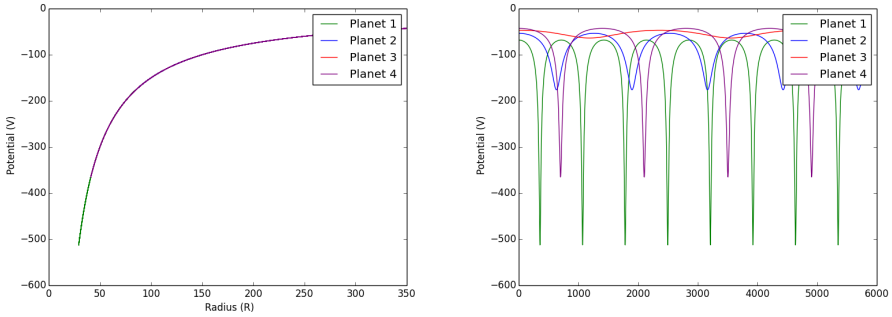


Figure 4: Potential change

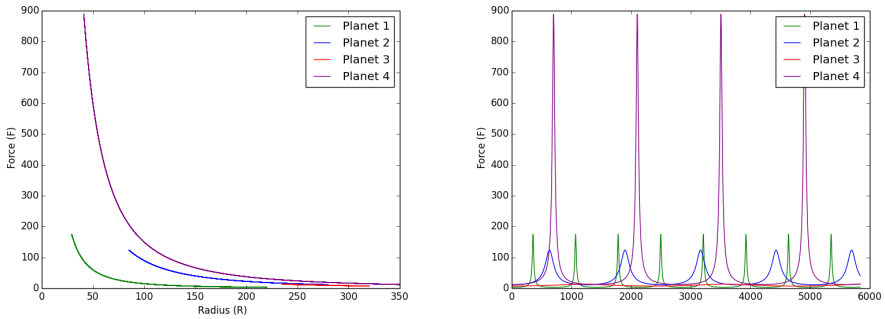


Figure 5: Force change

# 4 Conclusion

Figure 3 and 4, show that as the radius of the respective planets increases the potential energy also increases but never hits zero. Planets with smaller orbital radii have smaller changes in energy, but have larger changes in potential energy, in comparison to planets with larger orbital radii that have larger changes in energy but smaller changes in potential energy.

Figure 5 shows the change in force compared to the orbital radius. As the orbital radius of the planet increases, respectively the gravitational force caused by the sun decreases. Comparing figure 2, which displays the orbital paths of the planets with the initial conditions that the gravitational constant G is one and that the timestep is 0.1, with figure 5 shows that planets with orbits that are closer to the sun have larger changes in force. The presented data follows the math equation (1).

Furthermore figures 6-9 shows the orbital paths shows the orbits of respective planets, with varied initial conditions. In the animation we observed that when the planets approached the sun, they would accelerate in their orbit and then decelerate once they had gone past the sun. This phenomena occurs due to the conservation of angular momentum and energy, which causes the change in the velocity.

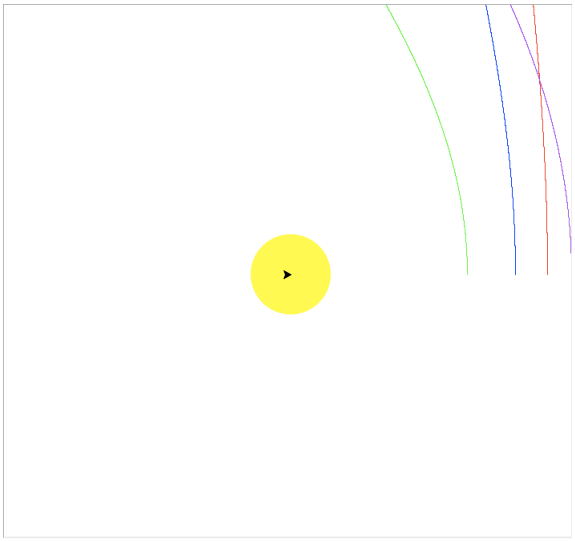

Figure 6: G=0.1, Timestep=0.1

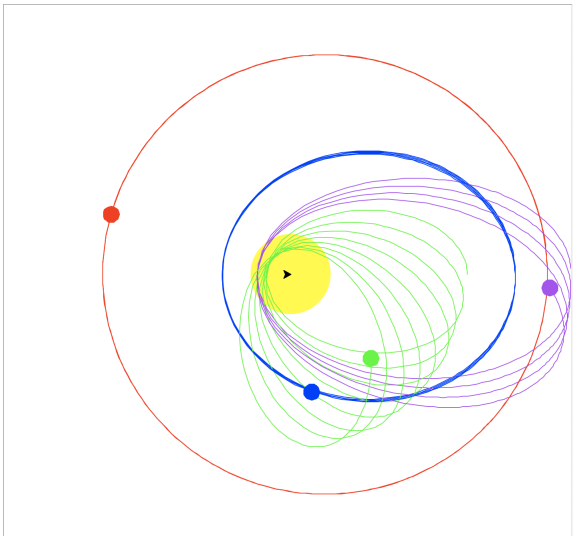

Figure 7: G=1, Timestep=1

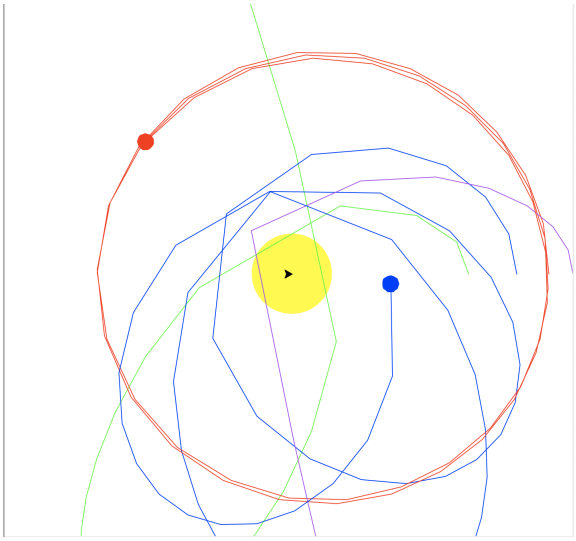

Figure 8: G=1, Timestep=10

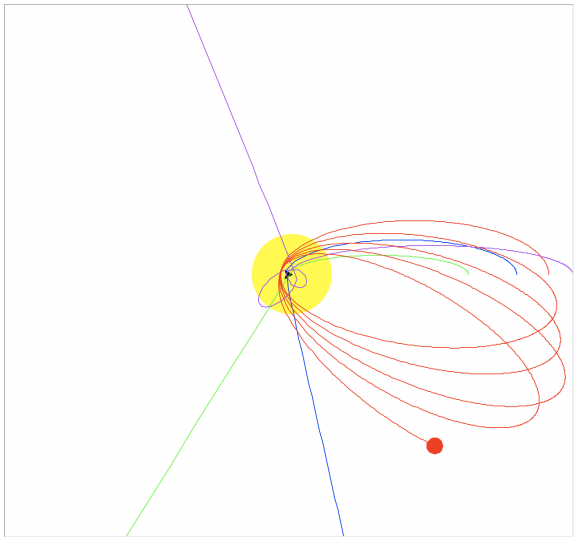

Figure 9: G=10, Timestep=0.1