**Planetary Motion**

**James McLeish**

The aim of this project was to design a program that investigated and demonstrated how planets and comets move around the sun. This is done by starting with the Newtonian equations of motion followed by a simplistic Euler algorithm which is also known as the Newton-Stormer-Verlet algorithm. With substitutions for ‘r’ and ‘v’ in the equations of motion, the Verlet algorithm can be used to update the velocity of an object and then from that updated velocity, find the updated positions. Along with the equations of motion, there is also an equation for energy as the energy in the Earth-Sun system should be constant since gravitation is a constant force, i.e. Total energy = kinetic energy +potential energy.

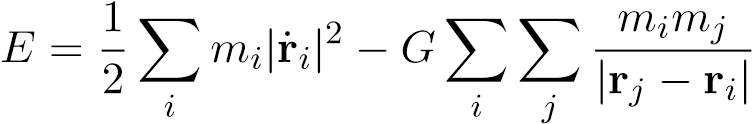
The basis of the entire project stems from the Newtonian equation below.

This equation states that, a force acting on an object, F, is equal to the gravitational constant G multiplied by the mass of the two objects multiplied together divided by their distance apart squared, ‘m1’, ‘m2’ and ‘r’ respectively. Newton’s equation can apply to objects of any size, but for this project we will be focusing solely on the forces acting between planets, mainly the Sun and the Earth. From this equation, if we presume we can write it as standard Newtonian equations, such as ṙ = v and v̇=F/m, then we can write a simplistic Euler algorithm, the Verlet algorithm, as a recursive set of equations which are written below.

Updated velocity:

Updated position:

These two equations will allow us, provided the required variables are input, to be able to accurately plot the orbit of Earth around the Sun with a high degree of accuracy. To ensure that the equations are working properly, an equation for the conservation of energy will also be used which follows:



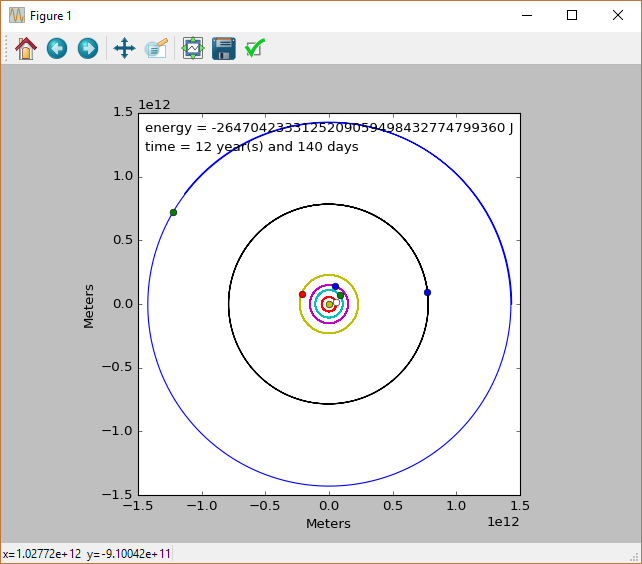
This equation is simply stating that the total energy in a system, e.g. Sun-Earth system, is equal to the kinetic energy minus the potential energy. The potential energy equation is more complex than the usual ‘mgh’ as it is taking the energy of the entire system. This is an issue as it isn’t at a height so it is instead using the gravitational constant multiplied by the multiple of the masses divided by the absolute value of their distance apart, i.e. the force of the objects.

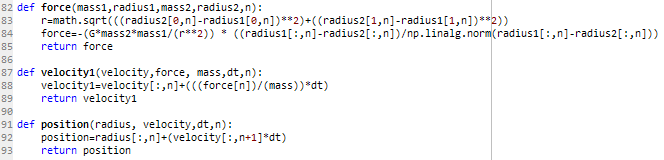
Along with simulating at least one planets using the Verlet algorithm above, the following are the all the aims in this project:

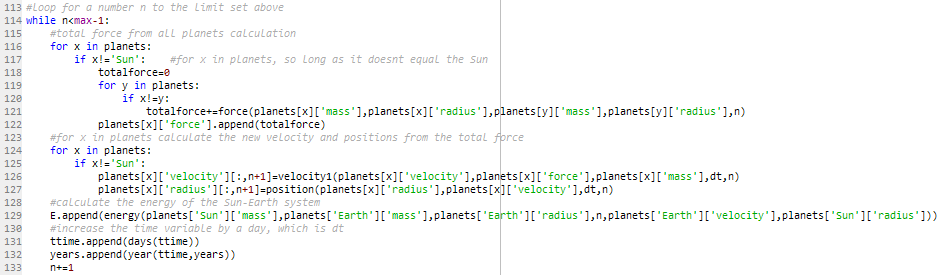
1. Simulate only the motion of an earth-like planet around the sun. The mass of the sun is 1*.*989 1030*kg*, and the radius of the earth is on average 149*.*59787 million kilometres (i.e. *R* = 149*.*59787 109*m*). Assume that the sun is fixed and that only the earth moves (a pretty good approximation in practice)
2. While the equations above are written in three dimensions, conservation of angular momentum makes the orbits of planets planar. All in all, for a single planet, you will be left with only two degrees of freedom.
3. First, verify that you get a closed orbit for a planet with the earth’s initial position **r**0 = (*R,*0) and tangential initial velocity *v*0 = (0*,*29*.*8*km/s*). Check that the total energy (kinetic and potential) remains constant. Your report should also derive that this value of *v*0 is the only one compatible with a (near) circular orbit.
4. Now vary the total energy of the orbit by changing the value, but not the direction, of *v*0, between 0 and a maximum value to be determined below.
5. Plot the now elliptical and hyperbolic trajectories, and the positions of the aphelion/perihelion. For the elliptical trajectories, calculate the orbital eccentricity (ratio of the minor to the semi-major axes).

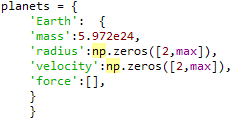
Also for the elliptical trajectories, test Kepler’s second law

1. Construct a criterion to distinguish elliptical and hyperbolic trajectories (only one ever comes back to the start). Determine the escape speed of an earth-like planet from the sun’s gravitational well. Determine the corresponding energy, and compare it to the theoretical result, which you can derive from *T* + *V* |initial = *T* + *V* |*r*=∞

Alongside is a screenshot of the animation in progress, which shows that multiple planets are being simulated through the use of dictionaries and functions, all which have a near perfect 2D circular orbit around the sun. Along with the orbits, the energy value of the Sun-Earth system and how ‘long’ the orbits have been going on for.

The method by which I completed this task was such that the Verlet equations above for force, updated velocity and updated position were all their own functions. This meant that I could simply put the equations into the code and just substitute the letters for the variable names that I used. The only difference was in the force equation, due to it being in 2D, also coding in Pythagoras’ theorem to determine the distance between the 2 objects, r, and multiplying the force by a unit vector to ensure the force is always going the correct direction.

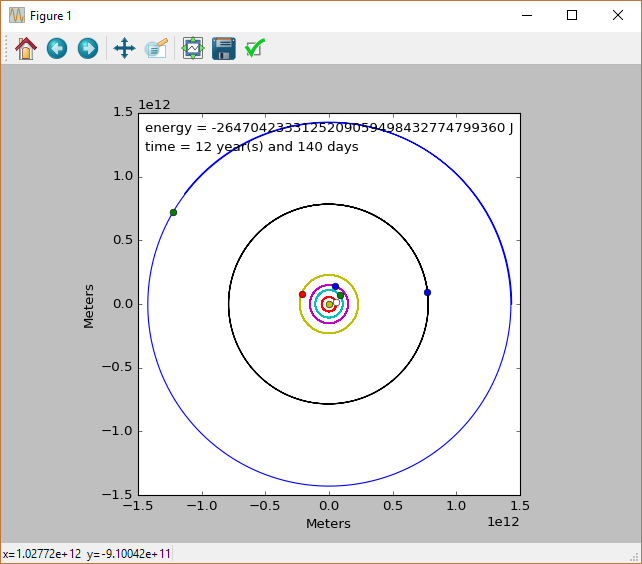
The other main part of my method is the loop which calls the functions and updates the positions and velocities of the planets in my program. Inside the ‘while’ loop which runs to a preset value, max, there are two for loops, one to calculate the force and one to calculate the updated positions and velocities. This is done separately as the total force exerted on a planet by all of the others needs to be calculated before some of them start moving to new positions. Once this total force has been calculated the new velocity and position, coded as radius, can be calculated. This was originally done by having lots of global arrays holding all the information, however for the final version a dictionary was used called ‘planets’ to make looping through all the planets simpler as they were all stored in the same place.

The ‘planets’ dictionary mentioned above contains all the variables of every planet added to the system. The example below which is Earth, shows that each planet contains a mass, radius, velocity and force variable. The radius and velocity variables are large arrays which are assigned starting values in a separate section but it allows the orbit to be plotted along with the animation when running the program

It was coded in this way to try and make it as simple as possible whilst also allowing the addition of new planets to be easy to do.

**Results**

Below on the can be seen the final finished screenshot of the animation that was created, which contains all planets from Mercury to Saturn orbiting in near circular orbits. This was done to complete the first 3 aims of this project which were simulate the motion of Earth around the Sun and verify that a closed orbit occurs. This can clearly be seen as every planet creates a circular orbit and returns to its starting point at a time close to its real life orbital period. Orbits with Earths starting velocity being altered will be shown further down but it will clarify that 29.8Km/s is the only speed to provide a nearly perfect circular orbit



To show that 29800m/s is the only speed at which a circular orbit occurs and to alter the energy of the system, the simulation was run with speeds of 40km/s, 29.8km/s and 20km/s and all plotted on the next image with a yellow dot for the sun plotted for better visualising the orbits. 40k/s is the largest orbit, the normal orbit. 29.8km/s is in blue and is circular and the orbit with a starting velocity of 20km/s is the closest orbit which is also not circular. This was also done to satisfy the fourth and fifth aims in regards to altering the total energy of the system and plotting the elliptical and hyperbolic trajectories that occur. The energy of each is also plotted below which shows the higher the speed and larger the orbit, the lower the total energy. 40km/s is the lowest energy and 29.8km/s it the middle line which both show energy is conserved as the line appears straight. However for some reason the 20km/s energy is a lot more varied and shows a series of tan lines although it still oscillates around a middle value. This could be caused by an error in my code that I did not notice or because the system gains its force from all other planets in the solar system it could be causing issues passing so close to Mercury and Venus. I suspect the later as it works for the higher and normal speed orbits.

