Project Presentation

Alexandru Hau

1 Overview

The purpose of the project is to simulate the motion of celestial bodies and see how the theory of classical physics (both at a high-school and university level) is applied. The whole project has been done on Python, using only the standard libraries Numpy, Scipy and Matplotlib.

The specific aims of the project are firstly to further understand the physics behind celestial motion of the bodies under various gravitational fields:

- Analyse how planetary orbits look like by input of various parameters
- Check how Kepler and Newton's laws are applied
- Work out how the orbits of some planets may shift after each full rotation (effect called eccentricity)
- Simulate on computer how the planetary dynamics would look if the attractive force between the planets would take other shape (such as logarithmic, sinusoidal or exponential form)
- Understand, in some cases, how heavy planets also move in binary planetary systems, hence grasp the physical concept of mass center
- Study how the energy conservation is applied every time, and realize how the kinetic and potential energies vary

However, the project can also be used for computational learning, as anyone who uses the program can upgrade the coding skills:

- Understand largely or in full detail how the code is designed

- Learn elementary Object-Oriented Programming design, vital for further Informatics projects and courses in any branch

2 Theoretical background

When a body rotates with respect to another stationary object, there is a force acting **radially** from the moving block with respect to the static one. This case is shown below:

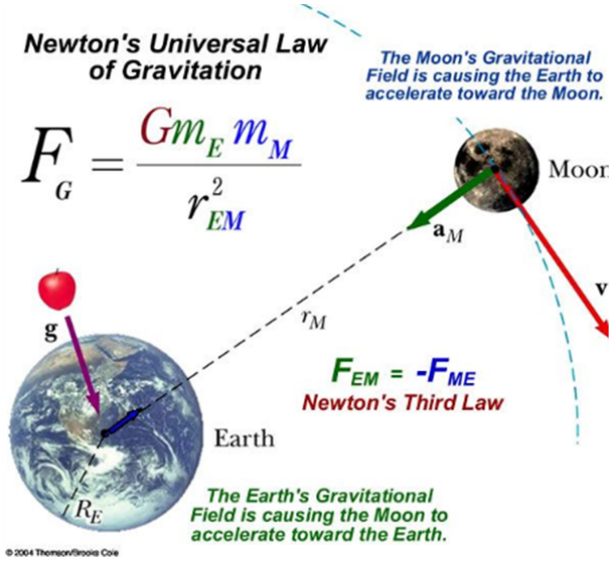

Figure 1: The moon is radially attracted towards Earth. Hence, it has a circular motion

In the case of celestial motion, they are all under the gravitational field influence. As a result, the attractive force between two bodies of masses M and m, at distance r with respect to each other, has the familiar expression:

$$F_g = \frac{GMm}{r^2} \tag{1}$$

If the motion is perfectly circular, this force represents the centripetal force, acting towards the center of the stationary body:

$$F_g = m\omega^2 r, where \omega = \frac{V}{r} \tag{2}$$

As a result, the two expressions can be used to work out the speed of the planet necessary to maintain circular trajectory at a specific orbit (radius):

$$V(r) = \sqrt{\frac{GM}{r}} \tag{3}$$

However, at the specific radius r, if the speed of the planet is different from v(r), then the trajectory is still enclosed and curved, but no longer circular. This is the case to almost all the interspacial blocks and planets, as the existence of perfectly circular trajectory is hindered by galactic collisions, heating from stars and so on. As specified above, the given project plots some shapes which the orbital planets can take.

3 Python files

There are two Python files - Body.py and Simulation.py. They have to be downloaded both in the same file for the simulation to work. Body.py builds the attributes for one particular object - the body. Simulation.py takes as input some parameters shown in the next section, builds up all the objects and runs the simulation. It is this file which has to be played

4 Running the project and putting the parameters

- Open the Simulation.py file
- Run the file and plug in the desired parameters

The required parameters are:

- Number of planets: simply select the number
- Color: choose from the following colors: y,b,r,g,c,m, y stands for yellow, r for red, and so on
- Mass: choose the required mass. It is in Earth Units, so as this adjustment makes it easier to understand the celestial motion. For instance, it would be more helpful to say that planet X

has the mass twice the Earth mass, rather than plugging in the actual value. Note: $M_{Earth} = 6 * 10^{24} kg$

- Distance: choose the required distance. Again, the distance is measured in astronomical units (1AU represents the distance from Earth to Sun which is $1.5 * 10^{11} m$
- For the first body, the program asks only for color and mass. This is considered the frame of reference, hence it is placed in the center of the animation. It can be the Sun, or any other kind of body. **Note: for a realistic animation, choose the first mass as 333000 times the Earth mass (actual Sun mass)**
- Fraction of circular trajectory speed: the circular trajectory speed is the V(r) function defined above. In order to see how much the planetary orbit evolves depending on the initial parameters, it is more convenient to use fractions than plain numbers, especially for beginner level in physics. Simply choose the desired fractions, both for x and y - axis. **Note: if fractions of circular trajectory for x and y are 0 and 1, this means no initial speed on x axis and full circular speed on y-axis. These are parameters for almost perfectly circular trajectory. It is recommended to begin with these parameters and then tune them to see how the system changes.**

Example of input for beginning:

- The number of planets is: 2
- The color of planet 1 is: y
- The mass of planet 1 in Earth masses is: 333000
- The color of planet 2 is: b
- The mass of planet 2 in Earth masses is: 1
- The distance of planet 2 wrt the Sun in AU is: 1
- The fraction of circular trajectory speed of planet on x axis 2 wrt the Sun is: 0
- The fraction of circular trajectory speed of planet on y axis 2 wrt the Sun is: 1

What has just been described above is the revolution of Earth around the Sun. The resut is on the .mp4 file attached with the description.

Of course, more planets can be put in the simulation, and plenty of interesting animations can result.

5 Advantages

- Any values can be chosen. There is no restriction on the parameters, or positions of planets. Hence, all kinds of motions can be analysed.
- Physics laws in classical mechanics can be better understood
- Coding and data analysis skills can be developed during such projects

6 Improvements for the project

- The scale of the plot has to be adjusted manually. The code has been designed for large distances and units. If, for instance, we want to simulate small displacements of the bodies, it will

be poorly shown on this code. For future, I could adjust the scalde to be done automatically

- I could also simulate random collisions with asteroids. Hence, students may also get an idea on how pre-istoric Solar System looked like (e.g: the Heavy Bombardment Event)
- I could also set the animations to be saved automatically, so they can be kept in document at any time and analysed again.