Tracing the path of satellite in stable orbit by simulating in Python

Nipun Aggarwal, Shubham Jadhav

January 2022

1 Introduction

From our childhood almost everyone curious about our solar system, planets, stars etc. In this project we did work to explore the same childhood curiosity of almost every child.In this project we plotted stable orbits of all planets present in our solar system by using some machine learning algorithms, We have solved differential equations for orbits of each planets by using data provided ,and plotted it on a 2-D graph by using Python 3 libraries.

2 Study Areas

Numerical methods to solve differential equations (in this project finite difference method), Visualizing planets trajectories using python, Some basic classical mechanics concepts

3 Motion under the influence of Central Forces

Central forces are forces between two particles that depend only on the distance between the particles and point from one particle to another. Central forces are conservative . Furthermore, if we look at a central force between two particles we can separate the motion of the canter of mass from the relative motion of the particles reducing the complexity of the problem, in this case centre(sun) and other planets moving about their centre of mass, and forming their stable orbits. With mutual forces between planets providing them centripetal acceleration and angular momentum, and balance of forces due to all planets on a certain palnet results forming an stable orbit around the sun.
Conservation of angular momentum:

$$L = r \times p \tag{1}$$

where r is the orbit radius, p is the momentum.

3.1 Effective Potential

The effective potential provides an useful method for simplifying a three-dimensional central-force problem down to a one-dimensional.
For a particle of mass m, polar coordinates (r, θ)

$$L = \frac{1}{2}m(\dot{r}^2 + r^2\dot{\theta}^2) - v(r) \tag{2}$$

is rather interesting.
It involves only the variable r. And it looks a lot like the equation for a particle moving in one dimension (labeled by the coordinate r) under the influence of the potential.
The equations of motion obtained from varying r and θ are

$$m\dot{\dot{r}} = mr\dot{\theta}^2 - V'(r)$$

$$\frac{d}{dt}(mr^2\dot{\theta}) = 0$$

Consider the example where $V(r) = Ar^2$. This is the potential for a spring with relaxed length zero. Then,

$$V_{eff}(r) = \frac{L^2}{2mr^2} + V(r)$$

$$F_{eff}(r) = \frac{L^2}{mr^3} - V'(r)$$

Consider the example where $V(r) = Ar^2$. This is the potential for a spring with relaxed length zero. Then,

$$V_{eff}(r) = \frac{L^2}{2mr^2} + Ar^2 \tag{3}$$

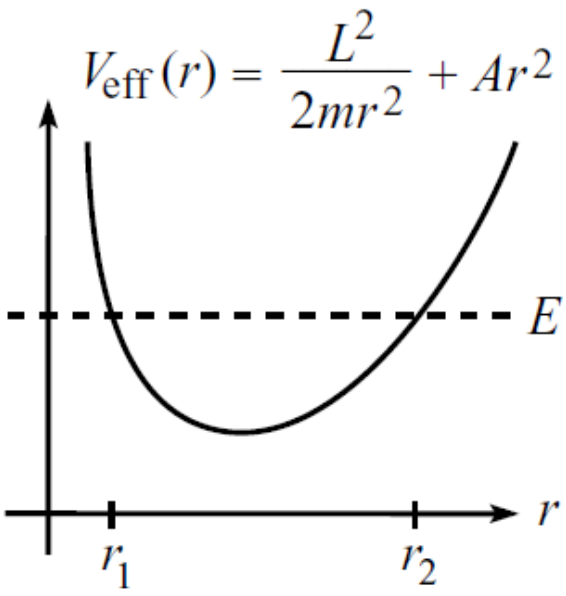

Figure 1: Plot of V_{eff}

The gravitational potential energy of the earth–sun system is

$$V(r) = -\frac{\alpha}{r} \tag{4}$$

where $\alpha = GMm$ M = mass of Sun
m = mass of Earth
In the present treatment, we'll consider the sun to be bolted down at the origin of our coordinate system. Since M >> m, this is approximately true for the Earth–Sun system.

$$(\frac{1}{r^2}\frac{dr}{d\theta})^2 = \frac{2mE}{L^2} - \frac{1}{r^2} + \frac{2m\alpha}{rL^2} \tag{5}$$

With all the $1/r$ terms floating around, it might be easier to solve for $1/r$ instead of r. Using

$$(\frac{dy}{d\theta})^2 = -\frac{\frac{dr}{d\theta}}{r^2} \tag{6}$$

square on the right-hand side gives

$$(\frac{dy}{d\theta})^2 = -(y - \frac{m\alpha}{L^2})^2 + \frac{2mE}{L^2} + (\frac{m\alpha}{L^2})^2 \tag{7}$$

If we take $z = y - \frac{m\alpha}{L^2}$

$$(\frac{dz}{d\theta})^2 = -z^2 + (\frac{m\alpha}{L^2})^2(1 + \frac{2EL^2}{m\alpha^2}) = -z^2 + B^2 \tag{8}$$

where,

$$B = (\frac{m\alpha}{L^2})\sqrt{1 + \frac{2EL^2}{m\alpha^2}} \tag{9}$$

If we take $z = Bcos(\theta - \theta_0)$

$$\int \frac{dz}{\sqrt{B^2 - z^2}} = \int d\theta \tag{10}$$

We know that $z = \frac{1}{r} - \frac{m\alpha}{L^2}$

$$\frac{1}{r} = \frac{m\alpha}{L^2}(1 + \epsilon cos\theta) \tag{11}$$

where

$$\epsilon = \sqrt{1 + \frac{2EL^2}{m\alpha^2}} \tag{12}$$

Equation (11) will be maximum when $cos\theta = 1$ which is when r will be min

$$r_{min} = \frac{L^2}{m\alpha(1 + \epsilon)} \tag{13}$$

If $\epsilon < 1$

$$r_{max} = \frac{L^2}{m\alpha(1 - \epsilon)} \tag{14}$$

4 Finite Difference Method

To solve the ODE boundary value problems is the finite difference method, where we can use finite difference formulas at evenly spaced grid points to approximate the differential equations. This way, we can transform a differential equation into a system of algebraic equations to solve.
In the finite difference method, the derivatives in the differential equation are approximated using the finite difference formulas. We can divide the the interval of [a,b] into n equal subintervals of length h as shown in the following figure.

Commonly, we usually use the central difference formulas in the finite difference methods due to the fact that they yield better accuracy. The differential equation is enforced only at the grid points, and the first and second derivatives are:

$$\frac{dy}{dx} = \frac{y_{i+1} - y_{i-1}}{2h}$$
$$\frac{d^2}{dx^2} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

These finite difference expressions are used to replace the derivatives of y in the differential equation which leads to a system of n+1 linear algebraic equations if the differential equation is linear. If the differential equation is nonlinear, the algebraic equations will also be nonlinear.

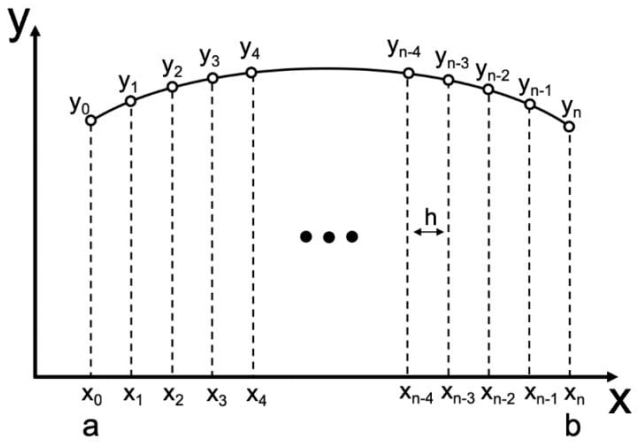

Figure 2: Representation of Finite Difference Method

5 Concepts used for simulation

Newtons law of gravitation and motion : by using newtons law of gravitation we found out gravitational force on each planet due to every planet

$$a = \frac{GM}{r^2} \tag{15}$$

By solving above differential equation we found out coordinates of trajectory of orbit of each planet and plotted it on a graph by using matplotlib.

6 Code Explanation

Let's come to the core of the project i.e. the code. We've used the Python programming language to implement the concepts and theory. For the sake of simplicity, we first plotted the orbit for a single planet and then extended the concept to multiple planets by making a list of variables instead of a single variable.
As discussed above, we implemented the finite difference numerical method so as to get the trajectory of the planets and took the initial parameters from the table for our calculation: Now, since we have the initial variables x_0, y_0, vx_0, and vy_0, we calculate the desired variables x_1, y_1, vx_1, vy_1 using finite difference method on the aforementioned formulas generating the following equations:

$$\begin{aligned} x_1 &= x_0 + vx_0 \times \delta t \\ y_1 &= y_0 + vy_0 \times \delta t \\ vx_1 &= vx_0 + Fx \times \delta t \\ vy_1 &= vy_0 + Fy \times \delta t \end{aligned}$$

where,

$$\begin{aligned} F_x &= -\frac{GMx_0}{(x_0^2 + y_0^2)^{1.5}} \\ F_y &= -\frac{GMy_0}{(x_0^2 + y_0^2)^{1.5}} \end{aligned}$$

denote the interaction force between sun and the chosen planet.
After calculating the values x_1, y_1, vx_1, and vy_1, we plot these x and y coordinates simultaneously using the matplotlib library. To obtain the whole trajectory, we assign the obtained variables x_1, y_1, vx_1, vy_1 back to x_0, y_0, vx_0 and vy_0 and iteratively repeat the above process until the desired number of frames. We have successfully obtained our result for a single planet.
Now, let's generalise the above code for n planets. We need to make only minor changes. Firstly, we'll make a list of variables $x_0, y_0, vx_0, vy_0, M, R$ instead of maintaining a single variable. Now, we'll execute the exact same logic as above, just for n different planets, one at a time.
So, the above equations would look like:

$$\begin{aligned} x_1[i^{th}planet] &= x_0[i^{th}planet] + vx_0[ith_planet] \times \delta t \\ y_1[i^{th}planet] &= y_0[i^{th}planet] + vy_0[i^{th}planet] \times \delta t \\ vx_1[i^{th}planet] &= vx_0[i^{th}planet] + Fx[i^{th}planet] \times \delta t \\ vy_1[i^{th}planet] &= vy_0[i^{th}planet] + Fy[i^{th}planet] \times \delta t \end{aligned}$$

where

$$\begin{aligned} Fx[i^{th}planet] &= -\Sigma\frac{GM\delta x}{(\delta x^2 + \delta y^2)^{1.5}} \\ Fy[i^{th}planet] &= -\Sigma\frac{GM\delta y}{(\delta x^2 + \delta y^2)^{1.5}} \end{aligned}$$

because we have to consider the interaction between ith planet and all other planets.
Similar to what we have done for a single planet,

Name	x_0	y_0	$v_{x,0}$	$v_{y,0}$	mass	radius
Sun	0	0	0	0	$1.989 \cdot 10^{30}$	695700000
Mercury	-46000000000	0	0	-58980	$0.33011 \cdot 10^{24}$	2439700
Venus	-107480000000	0	0	-35260	$4.8675 \cdot 10^{24}$	6051800
Earth	-147095000000	0	0	-30300	$5.972 \cdot 10^{24}$	6371000
Mars	-206620000000	0	0	-26500	$6.4171 \cdot 10^{23}$	3389500
Jupiter	-740520000000	0	0	-13720	$1898.19 \cdot 10^{24}$	71492000
Saturn	-1352550000000	0	0	-10180	$568.34 \cdot 10^{24}$	54364000
Uranus	-2741300000000	0	0	-7110	$86.813 \cdot 10^{24}$	24973000
Neptune	-4444450000000	0	0	-5500	$102.413 \cdot 10^{24}$	24341000

Figure 3: Initial parameters used for this simulation

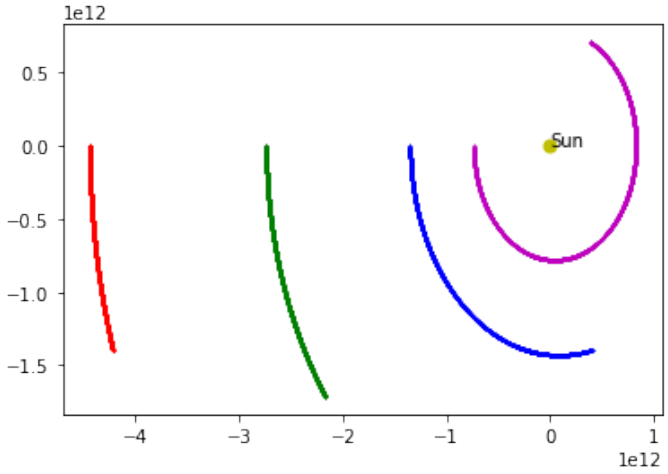

Figure 4: Final Simulation results

we plot x1[i^{th} planet] and y1[i^{th} planet] simultaneously using the matplotlib library. And we assign the obtained variables x1[i^{th} planet], y1[i^{th} planet], vx1[i^{th} planet], vy1[i^{th} planet] back to x0[i^{th} planet], y0[i^{th} planet], vx0[i^{th} planet] and vy0[i^{th} planet] and iteratively repeat the process for each planet, at different frames until we get the whole trajectory.

```
import matplotlib.pyplot as plt
# Constants
G = 6.67408 * 1e-11
delta_t = 86400
# Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
n = 9
x0 = [0, -46000000000, -107480000000, -147095000000, -206620000000, -740520000000, -135255
y0 = [0, 0, 0, 0, 0, 0, 0, 0, 0]
vx0 = [0, 0, 0, 0, 0, 0, 0, 0, 0]
vy0 = [0, -58980, -35260, -30300, -26500, -13720, -10180, -7110, -5500]
M = [1.989*1e30, 0.33011*1e24, 4.8675*1e24, 5.972*1e24, 6.4171*1e23, 1898.19*1e24, 568.34*
R = [695700000, 2439700, 6051800, 6371000, 3389500, 71492000, 54364000, 24973000, 24341000
color = ['wo', 'wo', 'wo', 'wo', 'wo', 'mo', 'bo', 'go', 'ro']

frames = 3000

x1 = [0] * n
y1 = [0] * n
vx1 = [0] * n
vy1 = [0] * n

# Solving
for i in range(frames):
    for j in range(1, n):
        # Calculations
        x1[j] = x0[j] + vx0[j] * delta_t
        y1[j] = y0[j] + vy0[j] * delta_t
        Fx = 0
        Fy = 0
        for k in range(n):
            if(j != k):
                delta_x = x0[j] - x0[k]
                delta_y = y0[j] - y0[k]
                Fx -= G*M[k]*(delta_x) / ((delta_x**2 + delta_y**2) ** 1.5)
                Fy -= G*M[k]*(delta_y) / ((delta_x**2 + delta_y**2) ** 1.5)
        vx1[j] = vx0[j] + Fx * delta_t
        vy1[j] = vy0[j] + Fy * delta_t
        # Updations
        x0[j] = x1[j]
        y0[j] = y1[j]
        vx0[j] = vx1[j]
        vy0[j] = vy1[j]
        # Plotting
        plt.plot(x1[j], y1[j], color[j], markersize = 0.5)
# Plotting Sun
plt.plot(0, 0, 'yo', markerSize = 7)
plt.annotate("Sun", (0, 0))
plt.show()
```