Project

Before beginning the project, it is highly recommended that you read this document in its entirety.

In this project, you will create a code to simulate the motions of the bodies in the solar system. You will then use this code to perform a series of experiments. Finally, you must write a report describing the work you have done and your findings.

1. Introduction

In 1543 Copernicus proposed that the known planets (at that time only the planets as far out as Saturn) were in circular orbits around the sun. This theory was revised by Kepler and Brahe who showed that the orbits were in fact elliptical. Since then the orbits of the planets have been very accurately calculated and can be precisely predicted far into the future. NASA uses powerful computers to determine both the position of the planets and the path of its space missions, including the Mars landing probes.

This project aims to simulate the motion of the planets in two-dimensions and to predict launch conditions for satellites to successfully reach planets.

2. Numerical Integration

The solar system represents a gravitational "many body problem", i.e., one that cannot be solved analytically, which means that a numerical integration scheme must be used. In this project we will use the three-step Beeman scheme. The Beeman algorithm is a stable method which predicts the position at the next time step by combining the current acceleration with the acceleration from the previous time step. This new position can then be used to calculate the new acceleration which, in turn, predicts the new velocity. The algorithm is given by

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t + \frac{1}{6}[4\vec{a}(t) - \vec{a}(t-\Delta t)]\Delta t^2,$$
$$\vec{v}(t+\Delta t) = \vec{v}(t) + \frac{1}{6}[2\vec{a}(t+\Delta t) + 5\vec{a}(t) - \vec{a}(t-\Delta t)]\Delta t,$$

where Δt is the small time-step, t is the current time and $\vec{r}$, $\vec{v}$, and $\vec{a}$ represent the position, velocity, and acceleration vectors for the motion.

3. Project Task: Simulation

Write an object-oriented Python program to simulate the orbit of the following planets around the sun: Mercury, Venus, Earth, Mars, and Jupiter. You should treat the simulation as a "many body

problem", so that adding new planets would not involve changing the design of the code. Your code should:

1. Read the planet details and simulation parameters from a file. The structure of this file is entirely up to you. One possibility is storing the information in the JSON format and using Python's standard libraray for parsing JSON files. Please see an example data file **`parameters_solar.json`** and an example Python program **`parse_json.py`** for reading the file.
2. Initialise the position of the planet to be on the positive x-axis, with an orbital radius that is read from the file. Initialise the velocity of the planet to be in the positive y direction with a magnitude from the Kepler circular orbit approximation:

$$v = \sqrt{\frac{GM}{r}}$$

   where M is the mass of the Sun, and r is the orbital radius.
3. Implement the Beeman integration scheme to update the position and velocity of the planets and the sun at each time step.
4. Show the orbit of the planets as they move around the sun in a graphical display.
5. Calculate and print the orbital periods of the planets in Earth years. You may also write them to file if you wish, but this is not required. Your code needs to automatically detect the time step at which a planet completes a full orbital cycle.
6. Regularly write out to a file the total energy of the system, i.e., the sum of kinetic and gravitational potential energy.

▶ Total Energy

For information on how to set the initial positions and velocities and how to determine the acceleration due to gravitational attraction, see the Orbital Motion checkpoint.

You will need to choose an appropriate time step for your simulation. A time step that is too short will not run your simulation for a long enough period of time. A time step that is too long will produce unstable orbits. A reasonable place to start is somewhere around a few hundred to one thousand timesteps per Earth year, but you should experiment to find a value that works for you.

4. Project Task: Experiments

Once you have a working code, you must work through the experiments below. **You must complete both experiments 1 and 2 and choose one of either experiment 3 or 4.**

These experiments will require you to extend your code. The source code you submit must be capable of performing the actions associated with the experiments you run. To receive full credit on the source code, a single version of the code must be capable of performing all experimental tasks. That is, it is not sufficient to submit multiple versions of the code with minor modifications for each experiment.

Experiment 1: Orbital Periods

This experiment is mandatory.

Check how closely the orbital periods of the planets in your simulation match their actual orbital periods, e.g. as given by this NASA page.

Experiment 2: Energy Conservation and Alternative Integration Methods

This experiment is mandatory.

First, check whether (or not) energy is conserved during your simulation. Illustrate your results graphically.

In checkpoint 5, you implemented the Euler-Cromer integration method as it is also good for equations of motion. As a reminder, the Euler-Cromer method is:

$$\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t$$
$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t+\Delta t)\Delta t$$

Modify your simulation code to use the Euler-Cromer method and compare its energy convervation properties with the Beeman method with appropriate figures or tables.

We use schemes like Beeman and Euler-Cromer as they are meant to conserve total energy with time. A much simpler method, known as Direct Euler or Forward Euler, is used to solve many systems of ordinary differential equations. In the Direct Euler method, the position and velocity are updated as follows:

$$\vec{r}(t+\Delta t) = \vec{r}(t) + \vec{v}(t)\Delta t$$
$$\vec{v}(t+\Delta t) = \vec{v}(t) + \vec{a}(t)\Delta t$$

Direct Euler is almost (but not quite) just Euler-Cromer with position and velocity updates in reverse order. Modify your simulation code again to use the Direct Euler method for updating position and velocity. Run this version of your code for a few hundred years with the same timestep and compare graphically the evolution of total energy vs. time with the other methods. As an optional extra, experiment with changing $\vec{a}(t)$ to $\vec{a}(t+\Delta t)$ in the velocity update for Direct Euler. As a hint, you should see significant differences between the behavior of Direct Euler and the other two methods. If you do not, something is wrong.

Think hard about the best way to implement all three methods in your simulation code. Try to avoid duplicating lots of code. This will help make it more readable and easier to debug as well.

Note
All other experiments should only be performed using the Beeman method.

Experiment 3: Satellite to Mars

Choose between this one and experiment 4.

Suppose you were to launch a satellite from Earth to perform a fly-past of Mars. Search for an initial velocity (or range of velocities) that enables a satellite to get close to Mars. Check and report on the following:

- How close to Mars your satellite gets.
- What your satellite's journey time is and how it compares to that of NASA's Perseverance mission.
- Whether your satellite ever gets back to Earth.

To perform this experiment you will need to extend your code. You should launch the satellite at the start of your simulation. Take care to start the satellite from just off Earth (so that there is no division by zero) and also to ensure that its mass and starting velocity are realistic - a probe could get to Mars in a few days if it were fired fast enough, but carrying that much fuel would be prohibitive!

Experiment 4: Planetary Alignment

Choose between this one and experiment 3.

How often do planetary alignments occur? Modify your code to detect planetary alignments based on

all planets being within some threshold (say 5°) of the mean angle. Do this for the five innermost planets (i.e., out to and including Jupiter). You may optionally add some or all of the three outermost planets.

5. What to Include with Your Source Code

The source code you submit must be capable of running the default simulation as outlined in Section 3 and all of the experiments you have performed. You primary codebase (i.e., the main classes that define the simulation) should be capable of running both the default simulation and the experiments. It is fine for this to be done using multiple test files that run your simulation code with different settings. If you write other codes to read in the data produced and make plots, it is not necessary to include these, but you may.

Your source code submission must include:

- all source code files that define your simulation
- all "test" files used to run your simulation and the experiments
- a README file with instructions on how your code should be used

The README file should be a plane text file. It should contain a list of all the files included and a few word description of each. It should also contain instructions for how to run the code both to perform the default simulation outlined in Section 3 and the experiments.

Note

Remember that your source code submission should be a zip file containing all of the above. **Do not submit each source code file individually.**

6. Project Task: Report

Write up the work you have done as a project report in the style of an article for a technical journal (e.g., for the APS or MNRAS). See Guidance on Writing the Project Report for full instructions on writing your report.

7. Project Marking Scheme

The project will be assessed according to the Project Marking Scheme.