SC 2 Final Project
5. Planetary Motion Simulator

Nico Toikka

February 5, 2021

Contents

1 Introduction

The object of the planetary motion simulator project was to create an n-body simulation of planetary objects, where the only effective force was gravity according to Newton's law of universal gravitation. User of the simulation is able to determine the initial values of the planetary objects. Since the simulation is for n-bodies, the simulator could be used to simulate larger (or smaller) systems than a single solar systems, but this would require some modifications to the units of the simulator.

The simulation can be used to approximate position, velocities, accelerations and orbital times of objects in a solar system. It can also demonstrate how the use of different time steps affects a simple simulation. With the simulation program is also included a fancier 3D animation Python program and a more useful 2D plotting Python program, which can be used to approximate the mentioned values along with the simulation.

2 Methods

For simulating the motion of the objects, we use the Velocity Verlet algorithm, where the properties of an object are calculated from the previous properties. For position ($\vec{r}$), acceleration ($\vec{a}$) and velocity ($\vec{v}$) the algorithm gives the following formulas:

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}\Delta t + 1/2\ \vec{a}(t)\ \Delta t^2 \tag{1}$$

$$\vec{a}(t + \Delta t) = F(\vec{r}(t + \Delta t))/m \tag{2}$$

$$\vec{v}(t + \Delta t) = \vec{v} + 1/2\ (\ \vec{a}(t) + \vec{a}(t + \Delta t))\ \Delta t \tag{3}$$

where Δt is time step.

As mentioned in Introduction, we only consider the effect of gravity according to Newton

$$\vec{F} = G\frac{m_1 m_2}{r^3}\vec{r}$$

where $\vec{r}$ is the position vector between the masses m_1 and m_2, G is the gravitational constant and r is the distance between the masses. The equation could also be written with $\vec{r}/r$ being a unit vector.

From Newton's second law of motion we get the acceleration caused by an object with a mass m_2 to an object of mass m_1

$$\vec{F} = m_1\vec{a}$$
$$G\frac{m_1 m_2}{r^3}\vec{r} = m_1\vec{a}$$
$$\vec{a} = G\frac{m_2}{r^3}\vec{r}$$

With these formulas and the user's initial values, the program is able to approximate all the necessary values for the objects in the solar system.

3 Implementation

3.1 Overview

The program is required to

- read user given values from a file
- calculate the evolution of the system according to the given values, and for the calculations to work, keep track of properties of the objects in the system
- write the simulated values to a file.

To meet the requirements the program includes modules for reading and writing to (pre-specified) files (file_mod.f90), creating derived types for calculations and storing values (types_mod.f90), calculating the dynamics of the objects (dynamics_mod.f90) and a module for parameters (params_mod.f90). The simulation itself takes place in the main part of the program (main.f90), where the simulation is implemented as a do-loop for given amount of time steps. In-depth descriptions are included in the source code, and compilation instructions are included in the /src subfolder's README.

3.2 Parameters and types

In it's current state the program has only two global parameters, gravitational constant and the kind of real numbers in the program. For clarity these parameters are stored in the parameters module (params_mod.f90), and storing them in a separate module also makes it easier to expand the program.

The program includes two derived types: vector and object. Types are created in the module types_mod.f90. For dynamics calculations the program is required to have a type for vectors and some basic operators for vectors. Vector type consist of three real

numbers (x, y, z) where each number represents the vector's value in the corresponding dimension. Vector operators implement basic linear operations, and the module also includes a subroutine for calculating the Euclidean length of a given vector.

The program also needs a way to represent the objects as data structures containing vectors and masses. This is done with a derived type "object", that has a description (characters), a mass (real number), a position (vector), a velocity (vector) and an acceleration (vector). Object type doesn't have any operators, since it's use is to store data. The objects are stored in an array during the simulation, where objects can be accessed with an array index.

3.3 Reading and writing to files

The module for reading and writing to files (file_mod.f90) contains the three subroutines: one for reading the input(read_input()), one for writing the output (write_data()) and one for clearing the previous output (clear_output()). The module also includes the variable system (1D object array), which is an array for storing all the objects in the system.

Subroutine read_input() reads the input.dat file from the run/ subfolder line by line. First lines tells the number of objects (n) in the system, next n-lines are the objects to be simulated, and after that the next five lines are instructions for the simulation (time step, length etc.). Specific format for the input file is described in the src/ subfolder's README. The number of objects in the system is used to allocate the necessary space for the system array.

Subroutine write_data() writes the names and the positions of the objects in the system line by line to the end of the pre-specified output file, output.dat, in the run/ subfolder.

Subroutine clear_output() deletes the previous output file, and also creates and initializes a new output file.

The output files are formatted as comma-separated values, and the clear_output subroutine creates a header for this type of formatting. As mentioned before, the input file complies to a program specific format, that can be read from the /src subfolder's README. Input and output units are SI-standard units, except for simulation length input, which is in days.

3.4 Dynamics of the objects

Changes in the objects positions, velocities and accelerations are calculated in dynamics_mod.f90. Calculations are done according to the formulas mentioned in the Theory section of the report, and the implementation of the formulas is via two subroutines.

Acceleration of an object is calculated in the function get_acc(object). The subroutine goes through all the objects in the system with a do-loop, calculates each objects gravitational force on the given object, sums the forces and divides with the given object's mass.

Changes in the system's object's properties are calculated in the subroutine update_sys(). The subroutine goes through all the objects in the system with a do-loop and calculates the new positions, velocities and accelerations of the objects as is described in formulas 1-3. For acceleration the subroutine uses the get_acc function.

3.5 Simulation and information during simulation

Simulation is implemented in the main part of the program (main.f90). Here the program reads and writes data according to given instructions and uses the update_sys subroutine of the dynamics module.

Data is first read from the input file and the previous output is cleared. After that simulation is done with a do-loop, where every k-steps info of the system is printed and every m-steps data is written to the output. Step intervals k and m are implemented to the do-loop with modulus.

Along with the simulation loop there are two subroutines: info(step) and pos_info(). Info prints information of the system as is required in the assignment, and pos_info is used to get the current positions of the objects in the system.

4 Results

4.1 Tasks

The project had three tasks to solve. These tasks can be simplified to

1. Use real values to simulate the relative motion of Sun and Jupiter. Adjust your time step and approximate Jupiter's orbital period.

   (a) What sort of time step did you start from and what was the largest time step for a reasonable approximation. Compare the approximation to Jupiter's real

orbital period

(b) Using the real orbital period of Jupiter compare the simulation's length of the orbit to a full orbit. Report the time step used for this approximation.

(c) What is the approximate radius of motion and period of motion for the Sun?

2. Find the appropriate objects and time steps to approximate:

(a) a year,

(b) a month.

3. Simulate the whole Solar System (Sun and planets).

(a) How good are the approximations of the orbital periods, if you use the same time step as in task 1?

(b) Find a good time step to approximate the orbital period of Mercury.

These tasks were solved using the 2D visualization program (2Dvis.py) in the src/ subfolder. Time steps and periods were manually input to the input file, and results were found through experimentation. The figures are in included in appendix A, Figures.

Input values used for analysis are included in the /run subfolder as .dat files. Subfolder's README includes more information of the input files. The subfolder also contains two 3D animations of the Solar System, one with inclinations and one without.

4.2 Results

1. Jupiter's orbital period in the simulation was found to be around 4400 days with a time step of 100 minutes.

(a) For Jupiter's orbital period I started with a low time step of 10 minutes and gradually increased it until the results started varying. Small differences were found after the 200 minute mark, and at a time step of ten hours the differences became notable (though still relatively small). Comparison is presented in Figure 1.

Jupiter's real orbital period is approximately 4330 days, so the resulting orbital period is 70 days longer.

(b) With a time step of 100 minutes, the simulation's Jupiter's orbit was 5 degrees short of a full orbit.

(c) By limiting the axes to show only the motion of the Sun, we see that the Sun's period of motion is approximately the period of motion for Jupiter and the radius of motion is roughly $0.5 \cdot 10^7$ km.

2. (a) A year is the time it takes for Earth to orbit the Sun. Similar to task 1, I started from a clearly too low time of simulation of 350 days with a time step of 10 seconds and increased the time of the simulation gradually until the orbit closed in the 2D plot. Differences in period started at a time step of around one hour. Proper time step for this simulation is around one to ten minutes. The effect of time step in this simulation is demonstrated in Figure 3.

   A year in the simulation takes 368 days.

   (b) A month is the time it takes for the Moon to orbit Earth. In the simulation we assumed Earth to start stationary (as was assumed with the Sun in previous simulations). Task was solved similarly to part a, and the effect of time step is demonstrated in Figure 4. Proper time step for this simulation is around three minutes, with notable differences rising at around six minutes.

   A month in the simulation takes 26.7 days.

3. See Figure 5 for the whole Solar System.

   (a) From Figure 6 we can see that with Jupiter's time step Mercury for example goes approximately two thirds of a period more than with Mercury's time step. We also see less variance in the orbits with a smaller time step. Mercury's time step is determined in part b of this task.

   (b) For finding the appropriate time step for Mercury, I first determined the period of motion for Mercury with unnecessarily small time step and increased the time step to a proper one.

   A time step of ten minutes was found to be suitable for simulating the orbit of Mercury. This is one tenth of the time step used for Jupiter. Also for curiosity, the orbital period for Mercury in the simulation is 85.5 days, two and a half days shorter than the real period.

   See Figure 7 for visualization.

5 Conclusions

From the results of the project we see that simplification of situations in simulation can lead to notably different values compared to real life (Jupiter's simulated period vs. real period), and that in some cases the precision of the simulation doesn't increase the accuracy of the simulation. Accuracy of the simulation is limited by the used algorithm, so to increase the accuracy we would need an other way to calculate the values of the objects.

Notable problems and possible improvements to the program are:

- Velocity Verlet is calculated so that some objects are affected by forces caused by objects that are already in the next time step. For example the program calculates objects O1 new values, and due to do-loop it uses the new values of O1 to calculate the new values for object O2, rather than using the old values. Possible solution is to store temporary values for objects and apply them all at the same time.

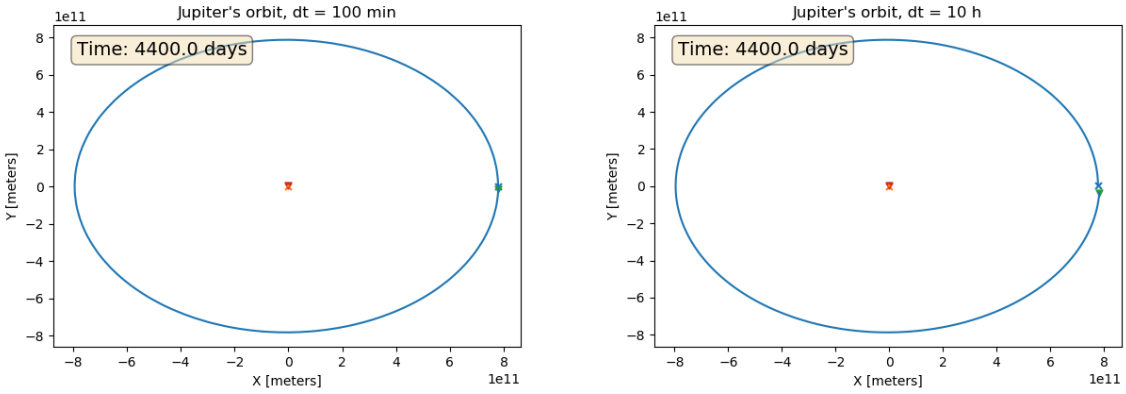
- Vector operators require specific format. For example vector· scalar works, but scalar· vector doesn't. This should be easily fixed with some time, and if the program is expanded it should be fixed to reduce possible errors.

- The program uses astronomically small units, which causes the values to be larger than needed. Unit conversion from meters to AU is a possible improvement.

- Time period analysis could be implemented to the program with some kind of if-conditions. Once the object would reach a specific distance from its initial position the program would stop and analyse the path and time.

A Figures

Legends are omitted from figures with two objects and from the figures of the inner planets. X represents the starting position of an object and triangle the final position.



(a) Time step of 100 minutes

(b) Time step of 8 hours

Figure 1: Comparison of Jupiter's orbit around the Sun with different time steps

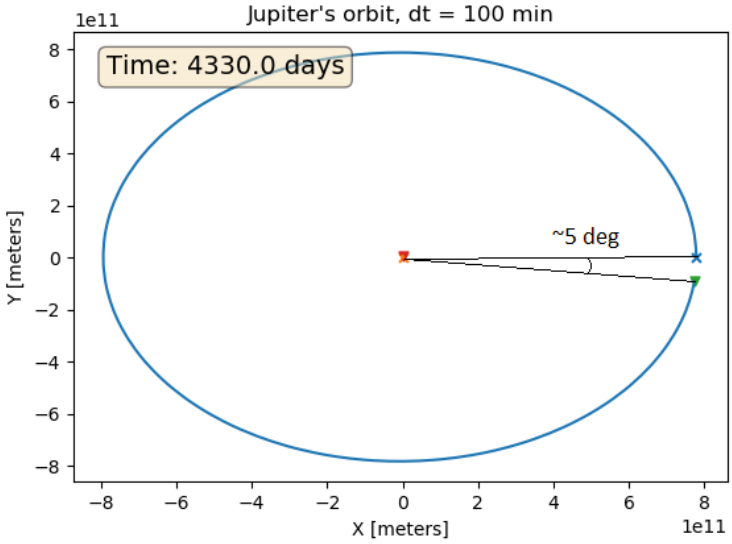
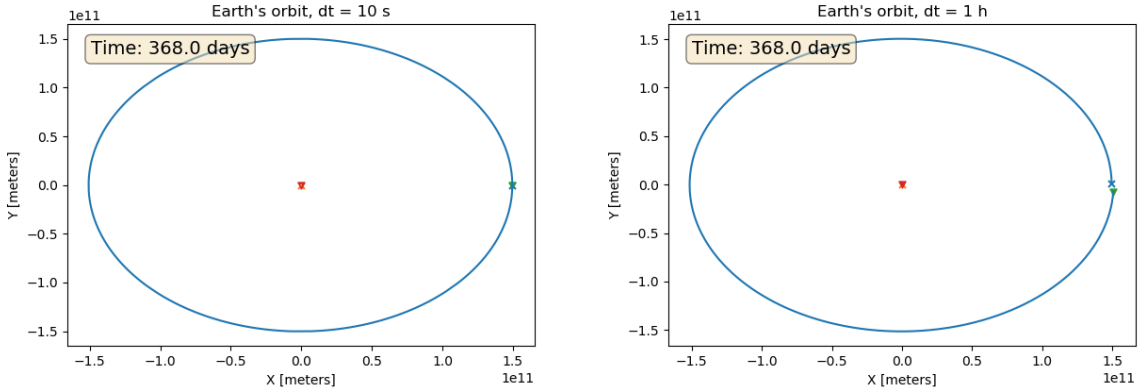
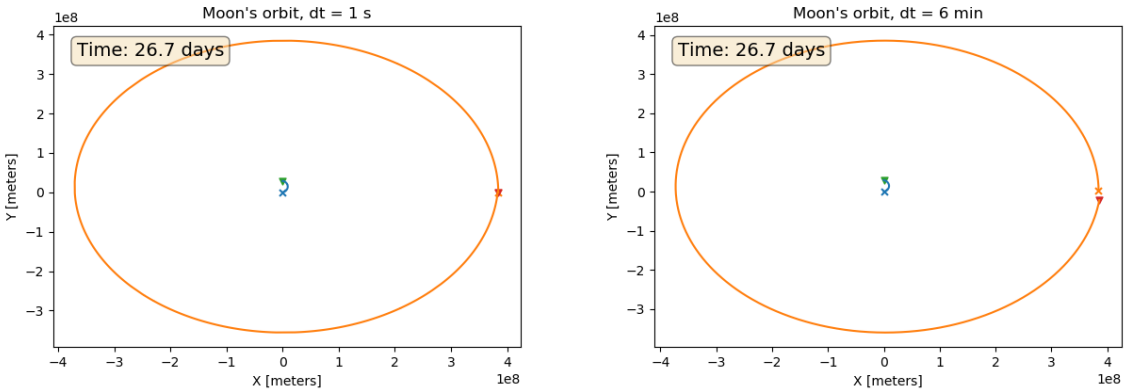


Figure 2: Simulation of Jupiter's orbit with the time of the real orbital period



(a) Time step of 10 seconds (b) Time step of one hour

Figure 3: Comparison of Earth's orbit around the Sun with different time steps



(a) Time step of one second (b) Time step of six minutes

Figure 4: Comparison of the Moon's orbit around Earth with different time steps

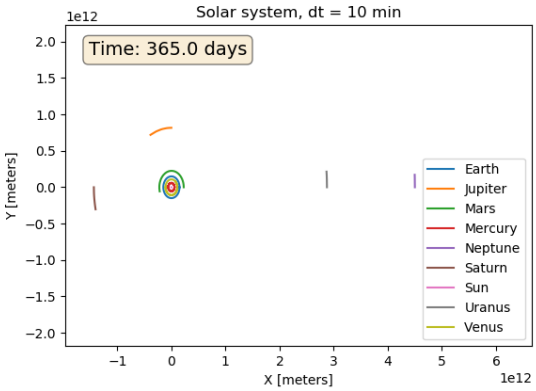
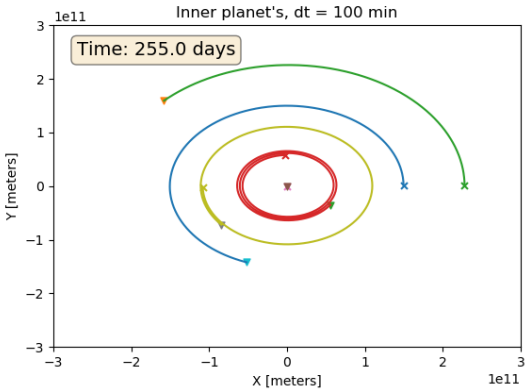
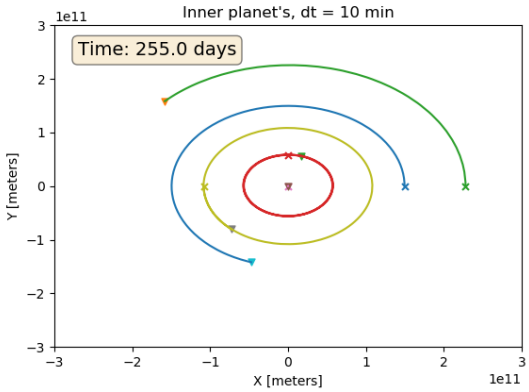


Figure 5: Simulation of the Solar System for a real year with a time step appropriate to Earth's orbit

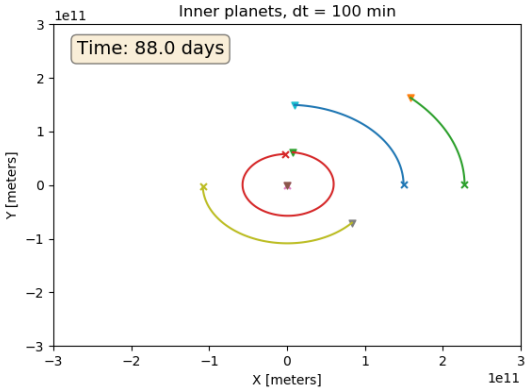


(a) Time step appropriate for Jupiter's orbital period

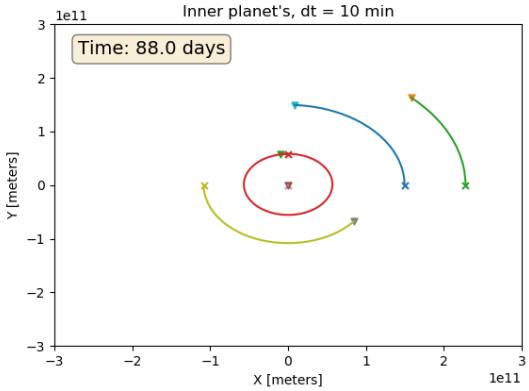


(b) Time step appropriate for Mercury's orbital period

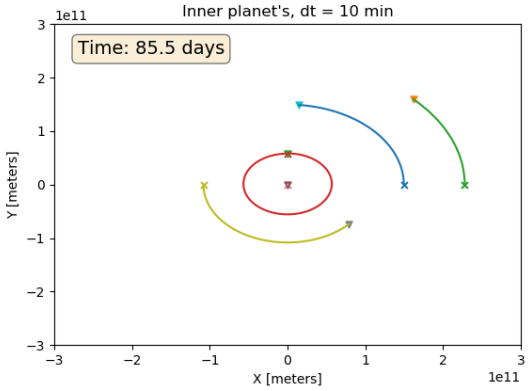
Figure 6: Limited plot of the Solar System with a time of Venus' real orbital period and differing time steps. Objects radially from the origin are: the Sun (origin), Mercury, Venus, Earth and Mars.



(a) Time step appropriate for Jupiter's orbital period



(b) Time step appropriate for Mercury's orbital period



(c) Mercury's period of motion

Figure 7: Figures used in the analysis of task 3b. Objects are the same as in Figure 6.