Ordinary differential equations
Building a model for the solar system
FYS3150: Computational Physics

Sigurd Sandvoll Sundberg*
Department of Geosciences, University of Oslo

(Dated: October 16, 2020)

I. INTRODUCTION

II. THEORY

A. Differential equations

The force acting upon objects in our Solar System are governed by Newton's gravitational law, which states that the force acting between two objects is

$$\vec{F}_{ij} = -G\frac{m_i m_j}{|r_{ij}|^2}\hat{r}_{ij} \tag{1}$$

where

1. F_{ij} is the force applied on object i by object j.
2. G is the gravitational constant.
3. m_i and m_j are respectively the masses of objects i and j.
4. $|r_{ij}| = |\vec{r_i} - \vec{r_j}|$ is the distance between j and i.
5. $\hat{r}_{ij} = \frac{\vec{r_i}-\vec{r_j}}{|\vec{r_i}-\vec{r_j}|}$ is the unit vector from object j to i.

We let the force act on the center of mass of the objects, and consider the objects to be point particles.

The total force acting on an object in the Solar System is given by

$$\vec{F}_i = \sum_{i\neq j}\vec{F}_{ij} \tag{2}$$

From Newton's second law of motion we have,

$$\vec{F} = m\vec{a}. \tag{3}$$

Using $\vec{F} = \vec{F}_{ij}$ we can derive that

$$\begin{aligned}\vec{a} &= \frac{\vec{F}}{m}\\ \frac{\partial^2}{\partial t^2}\vec{r} &= \frac{\vec{F}}{m}\end{aligned} \tag{4}$$

As the $\vec{F}$ is dependent on the position $\vec{r}$, we are dealing with coupled partial differential equations. We can rewrite this as a set of coupled ordinary differential equations(ODE),

$$\begin{aligned}\frac{\partial}{\partial t}\vec{v} &= \vec{a}\\ \frac{\partial}{\partial t}\vec{r} &= \vec{v}\end{aligned} \tag{5}$$

Knowing the position of all the objects in the Solar System we can find the force acting upon each object, and from this find their respective position over time, by solving this set of differential equations.

III. ALGORITHMS

In order to find the position for all bodies as time evolves we need to solve a set of couple differential equations 5 for each of the bodies at each time step. In our case for the solar system this set of couple differential equations is only possible to solve numerically.

To start off we want to scale our equations from SI units to more appropriate units for the solar system, these are Astronomical Unit(AU) for length, years(yr) for time and sun masses for mass. By looking at the Earth-Sun system, and assuming circular motion[2], we have

$$F_{\text{Earth}} = G\frac{M_{\text{Earth}}M_\odot}{r^2} = M_{\text{Earth}}\frac{v^2}{r} \tag{6}$$

This gives us

$$GM_\odot = v^2 r = (2\pi\text{AU/yr})^2 1\text{AU} = 4\pi^2\text{AU}^3/\text{yr}^2. \tag{7}$$

Using Sun masses[1], we then have

$$G = 4\pi^2\text{AU}^3/\text{yr}^2. \tag{8}$$

Our discretization of our continuous differential equations are done as follows

$$\begin{aligned}x(t) \to x(t_i) \to x_i \in [x_0, x_1, \ldots, x_i, \ldots, x_{n-1}]\\ v(t) \to v(t_i) \to v_i \in [v_0, v_1, \ldots, v_i, \ldots, v_{n-1}]\end{aligned} \tag{9}$$

and so on, where the initial values are known. If we know the initial time t_0 and the final time $t_{\max}$, and the number

* https://github.com/SigurdSundberg/FYS3150

[1] $M_\odot = 1$, and all other masses scaled as $M_i/M_\odot$.

of integration points n, we get the following expressions for t_i

$$t_i = t_0 + hi, \tag{10}$$

where $i = 0, 1, 2, \ldots, n-1$ and

$$h = \frac{t_{\max} - t_0}{n-1}. \tag{11}$$

To solve the coupled differential equations numerically we will be using two algorithms; forward Euler and velocity-Verlet method. Both methods are based on Taylor expansion

$$f(x \pm h) = f(x) \pm h\frac{d}{dx}f(x) + \frac{h^2}{2!}\frac{d^2}{dx^2}f(x) + \ldots \tag{12}$$

A. Forward Euler

The forward Euler algorithm is a simple numerical method to solve coupled ODE. Whilst simple it performs poorly when it comes to Hamiltonian systems, as it is not energy conserving. It uses the two first terms of the Taylor expansion to approximate the next step,

$$x_{i+1} = x_i + hv_i + O(h^2) \tag{13}$$

$$v_{i+1} = v_i + ha_i + O(h^2) \tag{14}$$

where x, v, a are vectors of up to dimension 3, and where i denotes the time step. h is the length of the time step. An algorithm for performing the forward Euler for the entire time period studied, i.e. doing foward Euler for all time steps reads as follows

Algorithm 1: Forward Euler

initialize x_0 and v_0;
for *i = 0,1,2,...,N-1* **do**
  compute a_i
  $x_{i+1} = x_i + hv_i$
  $v_{i+1} = v_i + ha_i$
end

This calculation has $4N$ FLOPs, without considering the FLOPs for calculating the acceleration, but trades efficiency for accuracy as the local error goes as $O(h^2)$ for both x and v.

B. Velocity-Verlet

For the majority of this project we will be using the velocity Verlet algorithm as it is known to preserve energy, making it better at working with Hamiltonian systems. It uses the three first terms of the Taylor expansion. The next time step is then given by

$$x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i + O(h^3) \tag{15}$$

$$v_{i+1} = v_i + \frac{h^2}{2}(a_{i+1} + a_i) + O(h^3) \tag{16}$$

Hvis tid legg til Appendix med utledningen av dette

We see that we would need both a_i and a_{i+1} to find v_{i+1}, where a_{i+1} is defined by the position x_{i+1}. That leads to the following algorithm where we first find a_i, then only after finding x_{i+1}, we find a_{i+1}.

Algorithm 2: Velocity Verlet

initialize x_0 and v_0;
for *i = 0,1,2,...,N-1* **do**
  compute a_i
  $x_{i+1} = x_i + hv_i + \frac{h^2}{2}a_i$
  compute a_{i+1}
  $v_{i+1} = v_i + \frac{h^2}{2}(a_{i+1} + a_i)$
end

This calculation has $9N$ FLOPs, without counting the number of FLOPs included in calculating the acceleration, which we have to do twice for each N. This is a greater amount of FLOPs comared to forward Euler, but at the trade off more precise conservation of energy.

IV. METHOD

A. Object Orientation and memory

In larger projects having to copy-paste code when adding more objects is both tedious and prone to error. So we want to design our code base such that we only have to write the code one time, and can keep adding objects to our system without any trouble. This is here achieved by having a **class** for our planets(objects) and a **class** for our system. Including we have a solver **class** which encompasses both the implemented forward Euler method and the velocity-Verlet method, and can be expanded upon if more differential equation solvers are needed. This makes our code base versatile and adaptable to many different problems and we can *write once, run many times*.

One essential problem which can be encountered however is memory limitation when vast amount of objects are added to the system, if we where to store the position of all objects at all time steps as double precision numbers, we would have for a three dimensional problem we would need to store $3N_{\text{objects}}n_{\text{time steps}}$ double precision numbers. A sample situation with $n = 10^7$, we would need roughly 2 giga bytes of memory to store the position of the planets in our Solar System. If we wanted to store more information such as the force acting on each planet at each time step, we would need roughly 4 giga bytes of memory. Such that storing values for each time step at all times becomes extremely inefficient and memory intensive. To get around that we overwrite the current set of data for each calculation, only storing the information that is needed to compute the next time step. To store the data we write the data to files at intervals, as not all data points are needed to study the cases of interest in this article.

V. CONCLUSION

[1] Morten Hjorth-Jensen. Computational physics, lecture notes fall 2015. *Department of Physics, University of Oslo*, page 173, 2015.

[2] J. L. Simon, P. Bretagnon, J. Chapront, M. Chapront-Touze, G. Francou, and J. Laskar. Numerical expressions for precession formulae and mean elements for the Moon and the planets. *Astronomy and Astrophysics*, 282:663, February 1994.