MODELLING THE SOLAR SYSTEM WITH ORDINARY DIFFERENTIAL EQUATIONS

SHAFAQ NAZ SHEIKH
MOONIS ALI KHALID
GITHUB.COM/MOONNESA/FYS3110-2020

Abstract. The main aim of this project is to simulate the solar system using coupled ordinary differential equations. We solve these ODE's numerically by implementing two algorithms, Euler's forward algorithm and the velocity verlet algorithm. In addition we want to object orient our code, this is because there are several coupled ordinary differential equations where the basic equations, except for the various physical constants and variables, are rather similar, so writing the code once, and running it many times, seems like a natural option. We find that the velocity verlet method is superior, when compared with Euler's method.

Contents

Date: 22nd October 2020.

1. Introduction

Differential equations are an essential tool in wide range of applications. This is because many physical phenomena can be modelled by a relationship between a function and its derivatives. Many naturally occurring quantities can be represented as mathematical functions. This includes physical quantities like position, velocity and acceleration, which may vary in both space and time. Thus one of the many applications of differential equations to physics, is the modelling of the solar system. The basic equations which govern the solar system are rather simple, a set of coupled equations that are based on Newton's law of motion due to the gravitational force. Therefore we have to compute the orbits of the planets. In order to do this we study and implement two algorithms, the forward Euler method and the velocity verlet method. These methods will help us solve the equations numerically. First we introduce the equations that govern the motion of the solar system, then we explain the algorithms that we are going to use. We then give an overview of object orientation, as this is something we wish to utilize in this project. Finally we present our results and discuss our findings before giving a conclusion.

2. Theory

2.1. **The differential equations.**

In order to study the solar system, we have to first consider the gravitational force. This is given by Newton's law of gravitaion. The equation for Newton's law of gravitation is:

$$F = G\frac{m_1 m_2}{r^2} \tag{1}$$

where F is the graviational force acting between two objects, m_1 and m_2 are the masses of the objects, r is the distance between the centers of their masses, and G[1] is the gravitational constant. By using Newton's second law of motion, we get the the following componential equations, in three dimensions, we would get three equations, as we are working with x, y and z in cartesian coordinates.

$$\frac{d^2x}{dt^2} = \frac{F_{G,x}}{m_i}, \quad \frac{d^2y}{dt^2} = \frac{F_{G,y}}{m_i}, \quad \frac{d^2z}{dt^2} = \frac{F_{G,z}}{m_i} \tag{2}$$

where $F_{G,x}, F_{G,y}$ and $F_{G,z}$ are the x, y and z components of the gravitational force and m_i is the mass a particular celestial body i. The equations shown in (2) are second-order ordinary differential equations. However, we can rewrite them as two coupled first-order ordinary differential equations, in each coordinate x, y and z.

In order to do this, we have to consider Newton's second law, we know that $\sum \vec{F} = ma$. We get an expression for the relationship between the acceleration of an object, and the force acting on it,

$$\vec{a_i} = \frac{\vec{F_i}}{m_i}$$

[1]In SI units G is approximately $6.674 \times 10^{-11} m^3 kg^{-1} s^{-2}$

where i is the particular body we are looking at. In addition, we have to look at the relationship between acceleration, velocity and position. Let a denote the acceleration, v denote the velocity and x denote the position, then in general we have that

$$\frac{dv}{dt} = a, \quad \frac{dx}{dt} = v$$

This is equivalent to

$$v(t) = x'(t)$$
$$a(t) = v'(t) = x''(t)$$

In three dimensions, we get these six coupled ordinary differential equations, we get two equations for each coordinate.

$$a_x = \frac{dv_x}{dt}, \quad v_x = \frac{dx}{dt} \tag{3}$$

$$a_y = \frac{dv_x}{dt}, \quad v_y = \frac{dy}{dt} \tag{4}$$

$$a_z = \frac{dv_x}{dt}, \quad v_z = \frac{dz}{dt} \tag{5}$$

Here we calculate the acceleration from Newton's law of gravitation. In vector form, in three dimensions, it becomes

$$\vec{F}_{ij} = G\frac{m_i m_j}{r^3}\vec{r} \tag{6}$$

This is similar to equation (1), here $\vec{F}_{ij}$ is the gravitational force between the two bodies i and j, and the vector $\vec{r}$ is the position of the body in cartesian coordinates.

2.2. **Units.** Since we are working with a physical system, we have to consider the units. We will be using astronomical units(AU) to measure length. One astronomical unit of length, known as 1 AU, is the average distance between the sun and the earth, that is $1AU = 1.5 \times 10^{11}m$. When it comes to time, we will use years instead of seconds, as this matches much better with the time evolution of the solar system. We will express the mass of the celestial bodies as a fraction of the mass of the sun. The mass of the sun is given by $M_{sun} = M_\odot = 2 \times 10^{30}kg$. For example, if we consider Earth, then the mass of the Earth is given by $6 \times 10^{24}kg$. Then the mass of the Earth relative to the sun will be given by $M_{Earth} = 6 \times 10^{-6}$.

We can find the units for the gravitational constant G, by looking at the Earth-Sun system. We assume that the orbit of the Earth is almost circular around the sun. For circular motion we know that the force must obey the following relation

$$F_G = \frac{M_{Earth}v^2}{r} = \frac{GM_\odot M_{Earth}}{r^2}$$

Rearranging the constants, we get

$$v^2 r = GM_\odot = 4\pi^2 AU^3/yr^2$$

Thus, the units for the gravitational constant are;

$$G = 4\pi^2 AU^3/(yr^2 M_\odot)$$

2.3. **Discretization.** Before we present the algorithms used to solve the ODE's, we have to say something about the differential equations. In order to solve the equations (3-5) numerically, we have to discretize them first. This is done in the same way as in the previous projects. However, we now have an initial value problem.

We discretize the time steps, we first have to define a value for T_0, which is the initial time, and T_{max}, which is the final time. We have to choose some N, which is the total number of integration points. Then the discretized time values are given by,

$$t_i = T_0 + hi \tag{7}$$

for $i = 0, 1, 2, ..., N$. Where h is the step size, it is defined as,

$$h = \frac{T_{max} - T_0}{N} \tag{8}$$

Since this is an initial value problem, we have to provide initial values before we can start to compute the solutions. We have to provide the initial position for the x, y and z coordinates, and we have to provide the initial velocity for each coordinate. We have to provide six initial conditions,

$$\begin{aligned} \text{initial positions} &= (x_0, y_0, z_0) \\ \text{initial velocity} &= (v_{x_0}, v_{y_0}, v_{z_0}) \end{aligned}$$

We wish to compute the orbits of the planets in our solar system as accurately as possible. We extract our initial conditions from data provided by NASA[2]. We can get initial conditions in all three dimensions. At the mentioned website, we need to change from OBSERVER to VECTOR and then write in the planet we are interested in. The generated data contains the x, y and z values, as well as their corresponding velocities.

3. Algorithms

3.1. **Euler's forward algorithm.** Eulers forward method uses the first two terms of the Taylor expansion to approximate the next step. The method can be written as,

$$\begin{aligned} \vec{x}_{i+1} &= \vec{x}_i + h\vec{x'}_i \\ &= \vec{x}_i + h\vec{v}_i \\ \vec{v}_{i+1} &= \vec{v}_i + h\vec{v'}_i \\ &= \vec{v}_i + h\vec{a}_i \end{aligned}$$

Euler's forward method is given by:

$$\vec{x}_{i+1} = \vec{x}_i + h\vec{v}_i \tag{9}$$

$$\vec{v}_{i+1} = \vec{v}_i + h\vec{a}_i \tag{10}$$

where h is the step size, as defined by (8), $\vec{x_i}$ is the position, $\vec{v_i}$ is the velocity and $\vec{a_i}$ is the acceleration. The algorithm can be written in psuedo code as it is shown in algorithm 1.

The number of FLOP's this algorithm uses is $4N$, this does not include the calculation of the acceleration. The local error in the algorithm is $O(h)$,

[2]`https://ssd.jpl.nasa.gov/horizons.cgi#top`

Algorithm 1 Euler's Forward method

```
define $h$
define the initial conditions $\vec{x}_0$ and $\vec{v}_0$
**for** i = 0, 1, 2, ..., N **do**
    find the acceleration $\vec{a}_i$
    $\vec{x}_{i+1} = \vec{x}_i + h\vec{v}_i$
    $\vec{v}_{i+1} = \vec{v}_i + h\vec{a}_i$
```

but the global error in $O(h)$. Euler's forward method does not conserve energy.

3.2. **Euler-Cromer's algorithm.** Another variation of Euler's method is Euler-Cromer's method. Although we did not implement this method in our code, it is still useful to take a look at it. The Euler-Cromer method is given by

$$\vec{v}_{i+1} = \vec{v}_i + h\vec{a}_i \tag{11}$$

$$\vec{x}_{i+1} = \vec{x}_i + h\vec{v}_{i+1} \tag{12}$$

The pseudo code is quite similar to Euler's forward method, it is given in algorithm 2. Euler-Cromer's method also has a global error of $O(h)$, however

Algorithm 2 Euler-Cromer's method

```
define $h$
define the initial conditions $\vec{x}_0$ and $\vec{v}_0$
**for** i = 0, 1, 2, ..., N **do**
    find the acceleration $\vec{a}_i$
    $\vec{v}_{i+1} = \vec{v}_i + h\vec{a}_i$
    $\vec{x}_{i+1} = \vec{x}_i + h\vec{v}_i$
```

this method is energy-conserving.

3.3. **Velocity Verlet algorithm.** The velocity verlet method belongs to a family of verlet methods. These methods are widely used as they are numerically stable and easy to implement. The velocity verlet method is given by,

$$\vec{x}_{i+1} = \vec{x}_i + h\vec{v}_i + \frac{h^2}{2}\vec{a}_i$$

$$\vec{v}_{i+1} = \vec{v}_i + \frac{h}{2}\Big(\vec{a}_i + \vec{a}_{i+1}\Big)$$

The pseudo code for the algorithm is given in algorithm 3. This method is energy conserving, it is much better than Euler's forward method. The number of FLOP's used in this algorithm is around $11N$. This method has a higher computational cost, compared to Euler's forward method, this is because we have to compute the acceleration twice, so more computations are needed.

Algorithm 3 Velocity Verlet method

```
define $h$
define the initial conditions $\vec{x}_0$ and $\vec{v}_0$
for i = 0, 1, 2, ..., N do
    find the acceleration $\vec{a}_i$
    $\vec{x}_{i+1} = \vec{x}_i + h\vec{v}_i + \frac{h^2}{2}\vec{a}_i$
    find the acceleration in the next time step $\vec{a}_{i+1}$
    $\vec{v}_{i+1} = \vec{v}_i + \frac{h}{2}(\vec{a}_i + \vec{a}_{i+1})$
```

3.4. **Object oriented implementation.** One of the aims of this project is to use object oriented programming. The concept of classes and object-oriented programming first appeared in the Simula programming language in the 1960s. Simula was invented by the Norwegian computer scientists Ole-Johan Dahl and Kristen Nygaard, and the impact of the language is particularly evident in C++.

Write once and run many times, is one of the central points of object orientation. Different people put different meanings into the term object-oriented programming, some use the term for programming with objects in general, while others use the term for programming with class hierarchies. Here, we apply the second meaning. We can put related classes together in families such that the family can be viewed as one unit. This idea helps to hide details in a program, and makes it easier to modify or extend the program. A family of classes is known as a class hierarchy. As in a biological family, there are parent classes and child classes. Child classes can inherit data and methods from parent classes, they can modify these data and methods, and they can add their own data and methods. This means that if we have a class with some functionality, we can extend this class by creating a child class and simply add the functionality we need. The original class is still available and the separate child class is small, since it does not need to repeat the code in the parent class. Although we could write the code for this project without classes, the introduction of classes enables us to write code that is either more elegant, or easier to extend at a later stage.

4. Results

5. Discussion

6. Conclusion

The three regression metho.... insert your conclusion here...

7. References

- All codes can be found in the github repository `https://github.com/moonnesa/FYS3150/tree/master/Project3`
- Langtangen, H., 2016. A Primer On Scientific Programming With Python. Heidelberg: Springer, pp.567,568.
- Mørken, K., 2017. Numerical Algorithms And Digital Representation. Blindern: Department of Mathematics, University of Oslo, pp.321-330.

- Morten Hjorth-Jensen. Computational Physics, Lecture notes, Fall 2015, chapter 8. 2015

Derivation of the bias and variance components of the MSE. Firstly we write the MSE in vector notation. Given $y = f + \epsilon$, we have that $\epsilon \sim \mathcal{N}(0, \sigma^2)$. So $\mathbb{E}[\epsilon] = 0$ and $Var(\epsilon) = 0$

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] = \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}]$$

By adding and subtracting $\mathbb{E}[\tilde{\boldsymbol{y}}]$ we get,

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] = \mathbb{E}[(\boldsymbol{f} + \boldsymbol{\epsilon} - \tilde{\boldsymbol{y}} + \mathbb{E}[\tilde{\boldsymbol{y}}] - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}}]$$

Expanding the brackets and calculating further we get,

(13)

$$\begin{aligned} F = \{F_x \in F_c :& \\ &\cap (\text{minPixels} < |S| < \text{maxPixels}) \\ &\cap (|S_{\text{conected}}| > |S| - \epsilon)\} \end{aligned}$$

$$= \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + 2\mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon}]$$
$$+ 2\ \mathbb{E}[\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}])] + 2\ \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])]$$

$$\begin{aligned} = \mathbb{E}[(\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}] + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + 2\mathbb{E}[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\boldsymbol{\epsilon}] \\ +2\mathbb{E}[\boldsymbol{\epsilon}(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}])] + 2\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])] \end{aligned}$$

$$\begin{aligned} = (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}} + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}] \\ +\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + 2[(\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}])\mathbb{E}[\boldsymbol{\epsilon}] \\ +2\mathbb{E}[\boldsymbol{\epsilon}]\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})] \\ +2\mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}}](\boldsymbol{f} - \mathbb{E}[\tilde{\boldsymbol{y}}]) \end{aligned}$$

This is equal to

$$= (\tilde{\boldsymbol{y}} - \mathbb{E}[\tilde{\boldsymbol{y}})^{\boldsymbol{2}} + \mathbb{E}[(\mathbb{E}[\tilde{\boldsymbol{y}}] - \tilde{\boldsymbol{y}})^{\boldsymbol{2}}] + \mathbb{E}[\boldsymbol{\epsilon}^{\boldsymbol{2}}]$$

Taking the sum over all is we get

$$\mathbb{E}[(\boldsymbol{y} - \tilde{\boldsymbol{y}})^2] = \frac{1}{n}\sum_i(\boldsymbol{f_i} - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}} + \frac{1}{n}\sum_i(\tilde{\boldsymbol{y}}_{\boldsymbol{i}} - \mathbb{E}[\tilde{\boldsymbol{y}}])^{\boldsymbol{2}} + \sigma^2$$