1 Introduction

Few problems in the history of science have drawn as much fascination as the movement of celestial bodies in space. What we now denominate the N-body problem, has been a subject of study since the beginning of history and is still being studied nowadays. One of the main methods to study this problem is through numerical simulations.

Astronomers have historically made their discoveries by looking through the eyepiece of a telescope. Nowadays, however, scientists can take advantage of the known laws of physics to create computer programs that mimic how the Universe works. Those programs, or simulations, can be used together with observations from telescopes as two faces of the same coin, to compare what is happening against what we think is supposed to happen. As the strength of astronomical observations relies on having a good telescope and using it correctly, the same applies to simulations and the numerical tools that are used to create them. The strength of computational astrophysics relies upon the quality of the numerical methods employed.

Our contribution to the field of computational astrophysics is the exploration and creation of new methods that optimize simulations of the gravitational N-body problem. We take advantage of the recent, paramount popularity of Machine Learning methods to find tools that can suit our problem. But let us take things one step at a time.

1.1 Part 1. The gravitational N-body problem

One of the first models of the Universe that is known to history dates from the fourth century BCE. Eudoxus believed that the universe was arranged as a series of nested spheres sharing the same center. The center would be the Earth and there would be one sphere for each of the five known planets, the Sun, and the Moon. Finally, one last sphere contained the stars. Aristotle, being a contemporary of Eudoxus, contributed to improving the model by adding some spheres to counteract the motion of the previous planetary sphere (see a more complete historical description in Americo (2017)). In the second century AD, the Alexandrian astronomer Ptolemaeus formulated a new model in his *Almagest* and Planetary Hypotheses (Toomer (1998)). This model was exceedingly complex. In order to replicate the perceived movement of the celestial bodies in the sky, this model indicated that the movement of each of the five planets is formed by two circles; the epicycle and the deferent (see Figure 1.1). The difficulty of the model lies in the fact that the deferent moves off-center with respect to the Earth but this movement was necessary to account for the “imperfections” that had been observed by Greek and Babylonian astronomers.

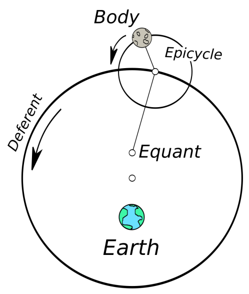
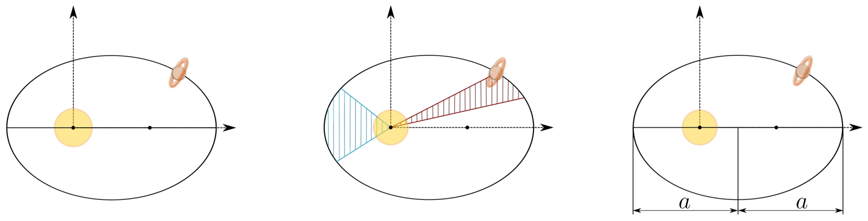


Figure 1.1: Simplified version of Ptolomaeus’ model of the Solar System.

Ptolemaeus’ *Almagest* became a very relevant text for astronomers in the Islamic world, and by the year 850 al-Fārghanī, a Persian astronomer, had used the current advances to update the astronomical theory explained by Ptolomaeus. In 1543, Copernicus proposed the first heliocentric model of the Solar System in his work: *On the Revolutions of the Celestial Spheres* (Copernicus (1543)). Based on the complex model by Ptolomaeus, he simplified it by positioning the Sun in the center of the Universe instead of the Earth. Between 1609 and 1619, Johannes Kepler published his famous three laws of planetary motion. Challenging the models by Copernicus and Brahe, he stated that the planets followed elliptical orbits with the Sun located in one of the foci (Figure 1.2a). He also established that a planet swept equal areas in equal times, which means that it travels faster when close to the Sun (Figure 1.2b). Finally, his third law states that the orbital period is proportional to the cube of the semi-major axis of the orbit (Figure 1.2c).



(a) First law. **(b)** Second law. **(c)** Third law.

Figure 1.2: The three Kepler laws of planetary motion.

The publication of Isaac Newton’s *Philosophiae Naturalis Principia* in 1687 (Newton (1687)) provided for the first time a unified equation of universal gravitation that explained Kepler’s empirical results. This law states that the two bodies attract each other with a force ($\vec{F}$) that is proportional to their masses (m) and inversely proportional to the square of the distance between them ($\vec{r}_{12}$);

$$\vec{F} = G\frac{m_1 m_2}{|\vec{r}_{12}|^3}\,\vec{r}_{12}. \tag{1.1}$$

Newton's universal law of gravitation (Equation 1.1) remains valid to our days for many cases. However, with the presentation of Albert Einstein's theory of General Relativity in 1915, it became clear that gravity is a perceived consequence of the motion through space-time. This conclusion led to the final understanding of phenomena such as the abnormal orbit of Mercury and the existence of black holes. However, for the remainder of this work, we will ignore relativistic effects and focus on systems dominated by classical mechanics, i.e. Newtonian mechanics.

1.1.1 The two-body problem

Two celestial bodies attract each other through their gravitational force following Newton's law of gravitation (Equation 1.1)). The force exerted by each body on the others is proportional to its mass and inversely proportional to the distance separating them (see Figure 1.3).

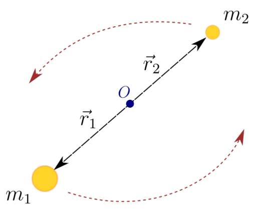


Figure 1.3: Simplified representation of the two-body problem.

Thanks to the equations derived by Kepler for the motion of a body orbiting another, the trajectory of a two-body problem can be calculated analytically. This means that knowing the state of a body around another one, finding its state at any future time is a fast and relatively easy problem to solve. However, it should be taken into consideration that a challenge arises when the eccentricity of the orbit is close to 1 (we will explain the meaning of the eccentricity in Subsection 1.1.2). In this case, it becomes challenging to achieve convergence in the solver for Kepler's equation (Elipe et al. (2017)).

1.1.2 State representation

Every system of N bodies has $6N$ degrees of freedom. The state of the system ($\vec{s}$) can be defined with 6 independent variables for each of the bodies present. The most common method to fully describe a system is with their positions (3 values per particle for a 3-dimensional space) and their velocity (also 3 values for a 3-dimensional space):

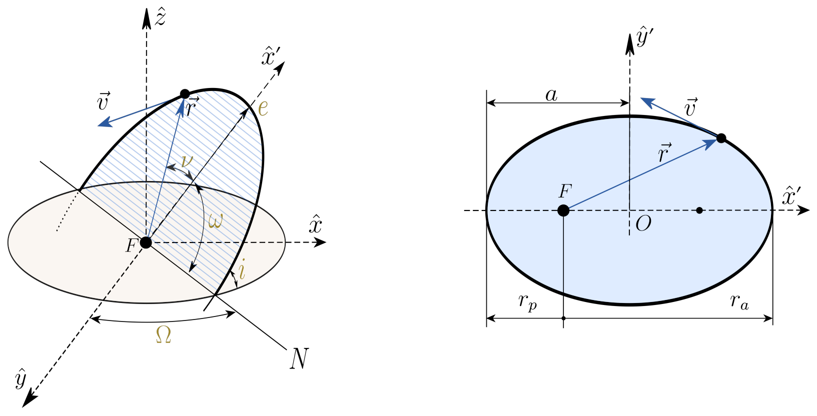
$$\vec{s} = [x,\ y,\ z,\ v_x,\ v_y,\ v_z]. \tag{1.2}$$

A particular case of the two-body problem is that in which one body is significantly more massive than the other. This is for example the case of our Solar System, as the Sun is more massive than any other body around it. If we imagine a system formed only by the Sun and one minor body orbiting it (like a small planet or an asteroid), the minor body will orbit the major following an ellipse. In this case, the state of the minor body can be described by its position and velocity, but it becomes more convenient in many cases to use Keplerian elements.

Keplerian elements describe a body focusing on the shape of its orbit (Curtis (2019)). The system is fully described with 6 elements which are shown in Figures 1.4a and 1.4b. In the figures, N represents the line of nodes, i.e., the intersection line between the orbital plane and the ecliptic plane. The ecliptic plane is defined by the equator of the central body.

Semi-major axis (a): the semi-major axis is a measurement of the size of the orbit. It is half the size of the major axis of the ellipse. It is a quantity with units of length.

Eccentricity (e): describes the shape of the orbit. For an ellipse, the eccentricity is a value between 0 and 1. When the eccentricity is 1, the trajectory becomes a parabola, and for eccentricities larger than 1, a hyperbola. This means that the orbiting body is no longer bound to the central one, but has escaped the system. It is an adimensional quantity.



(a) Three-dimensional representation.

(b) Planar orbit representation.

Figure 1.4: Representation of the different elements that define an orbit.

Inclination (i): the inclination is the angle between the orbital plane and the ecliptic plane. It is a positive number between 0° and 180°.

Argument of perigee (ω): angle between the line of nodes N and the eccentricity vector, which is the vector from the center body to the periapsis of the orbit. It therefore defines the position of the periapsis. It is an angle between 0° and 360°.

Right ascension of the ascending node (Ω): angle between the y-axis $(\hat{y})$ and the line of nodes (N). It is an angle between 0° and 360°.

True anomaly (ν): the true anomaly, defines the position of the object in its orbit. It is the angle between the x-axis of the orbit $\hat{x}'$ and the position vector (see Figure 1.4b). It is defined between 0° and 360°.

Therefore, the state of the system $\vec{s}$ can also be defined as:

$$\vec{s} = [a,\ e,\ i,\ \omega,\ \Omega,\ \nu]. \tag{1.3}$$

Additionally, it is important to define the periapsis and apoapsis as the closest and furthest points in the orbit from the central body, respectively.

1.1.3 Hamiltonian Systems and Conservation laws

In systems where the forces are derived from a potential function, the equations of motion can be written as a Hamiltonian system (Easton (1993)). The Hamiltonian formalism is the mathematical structure in which to develop the theory of conservative mechanical systems (Yahalom (2024)). A system of N particles interacting via Newtonian gravitational forces is a Hamiltonian system that can be described using Hamilton's formulation. The system is then formed by two first-order ordinary differential equations for each particle n as

$$\frac{d\vec{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \vec{p}_i}, \qquad \frac{d\vec{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \vec{q}_i}, \qquad i = 1, ..., N; \tag{1.4}$$

where $\vec{q}$ and $\vec{p}$ represent the position and momentum vectors, respectively. The momentum can be calculated by multiplying the mass of a particle by its velocity vector. The Hamiltonian $\mathcal{H}$ represents the total energy of the system and is formed by two terms: one for the kinetic energy and one for the potential energy. It is defined as a function of the position and momentum vectors of the system, the masses m of the bodies, and the universal gravitational constant G as

$$\mathcal{H} = \underbrace{\sum_{i=0}^{N-1} \frac{||\vec{p}_i||^2}{2m_i}}_{\text{Kinetic Energy}} \underbrace{- G \sum_{i=0}^{N-2} m_i \sum_{j=i+1}^{N-1} \frac{m_j}{||\vec{q}_j - \vec{q}_i||}}_{\text{Potential Energy}}. \tag{1.5}$$

In Equation 1.5, the Hamiltonian is independent of time. Therefore,

$$\frac{d\mathcal{H}}{dt} = 0, \tag{1.6}$$

which means that the total energy of the system is conserved. A change in energy in a numerical simulation is an indication of the system not following the laws of nature, and can therefore be used as an indication of the quality of numerical simulations (see Subsection 1.1.5 and Chapters 2 to 5).

The second conservation law is the conservation of angular momentum. Angular momentum is defined as the cross product of the position and velocity vectors (see Figure 1.5) in the form:

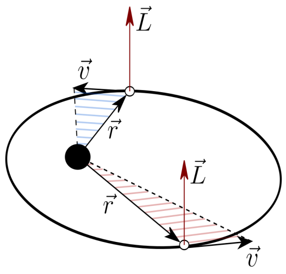


Figure 1.5: Conservation of angular momentum for a system of two bodies.

$$\vec{L} = \sum_{i=0}^{N} m_i \vec{r}_i \times \frac{d\vec{r}_i}{dt}. \tag{1.7}$$

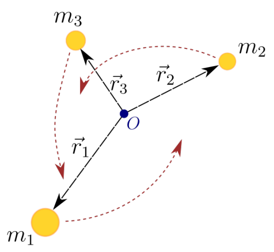
The derivative of the angular momentum equation with respect to time is 0, which means that angular momentum is conserved (Curtis (2019)). This implies that the movement of one body around another remains in a single plane.

1.1.4 The *N*-body problem

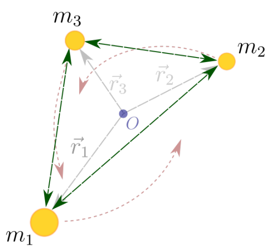
Pairs of celestial bodies are usually not found in isolation. Most of them are influenced by other bodies or are a part of a larger system. Thus, planets are found in planetary systems, stars are born in groups denominated star clusters, and all of those are also grouped in galaxies. The Universe is a system of *N*-bodies that interact with one another. In reality, most problems can be simplified by assuming that only the closest, or more massive, bodies will have an influence over the one being studied. For example, the gravitational potential of the center of our galaxy will always influence the movement of every object in the Solar System. However, because this effect will be small compared to other more relevant ones, for example, the influence of the Sun or Jupiter, we can choose to ignore this far-away influence depending on the phenomenon that is to be studied.

The simplest case of the *N*-body problem is a system with three bodies. In a system as the one represented in Figure 1.6a where three equal-mass stars move around their center of mass, each body interacts with the other two (Figure 1.6b) with a force defined by Newton's equation (Equation 1.1). Unlike in the two-body problem where one particle followed an ellipse around the other one[1], for a system of three bodies, the trajectories followed are complex and vary substantially depending on the initial conditions chosen. Three examples of those trajectories are shown in Figure 1.7. We will talk in more depth about chaos in the *N*-body problem and its implications for numerical integration in Subsection 1.1.6. As the number of bodies increases, so does the level of complexity of the trajectories followed by individual bodies.

[1]Fixing the center of the system at the center of mass of one of the bodies.



(a) Representation of the three-body problem.



(b) Interactions in the three-body problem.

Figure 1.6: Simplified representation of the three-body problem and the interactions between bodies.

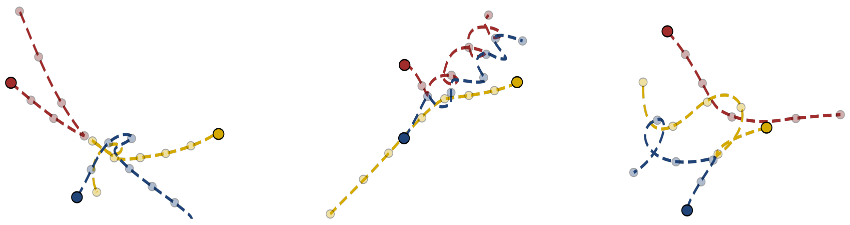


Figure 1.7: Examples of the trajectory of a system of three bodies.

In a two-body problem, the calculation of the gravitational force between the bodies has only one contribution. For a system of N bodies, Newton's equation can be rewritten as

$$\vec{F}_i = Gm_i \sum_{j \neq i} \frac{m_j}{|\vec{r}_{ij}|^2} \hat{r}_{ij}, \qquad i = 1, ..., N; \tag{1.8}$$

for each body. To calculate all the forces in a system of three particles, it would take 6 operations (or three times Equation 1.8). This number grows with the number of bodies in the system. For N bodies, the complexity -or number of calculations- scales quadratically with N. Knowing that the force exerted on body i by another body j is reciprocal to the one by body j on i, the number of calculations can be reduced.

Unlike in the case of the two-body problem, there is no analytical solution for the equations of motion. Thus, in order to know the future state of the system, we cannot apply a given equation, but we need to take small steps using numerical integrators.

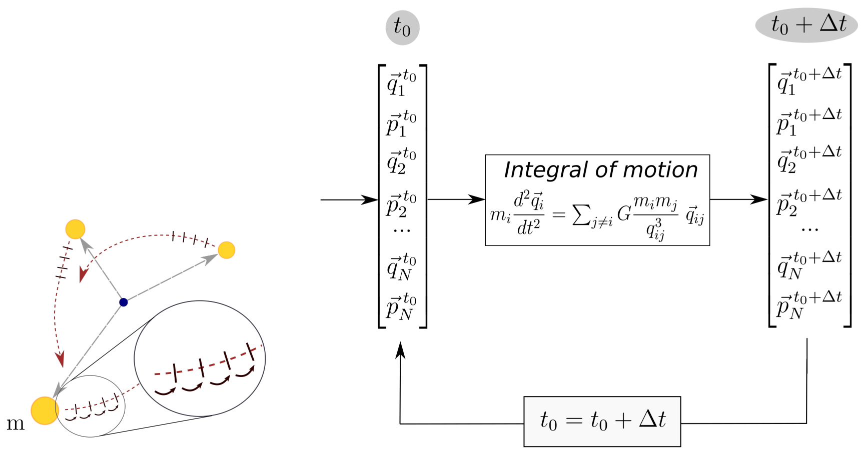
1.1.5 Numerical integration

Celestial bodies move in a continuous phase-space. At each moment in time their positions, velocities, and the forces acting on them vary. However, their trajectories cannot be computed analytically for $N \geq 3$, so we need to use numerical integration methods. Such methods propose approximate solutions for a given integral. For the problem of N

bodies moving under the influence of gravity, the differential equation to be solved can be written as a first-order differential equation

$$\begin{pmatrix} \dot{\vec{r}} \\ \dot{\vec{v}} \end{pmatrix} = \begin{pmatrix} \vec{v} \\ -G\sum_{i\neq j} \frac{m_j}{r_{ij}^2}\hat{r}_{ij} \end{pmatrix}. \tag{1.9}$$

In our case, numerical integrators divide the trajectory into discrete parts called steps and solve the equations of motion (Equation 1.9) for each of those consecutively. A representation of the three-body problem can be seen in Figure 1.8a. From the initial conditions or initial state $\vec{s}^{\,t_0}$, the state of the system after one time step Δt can be calculated using Equation 1.9. Once we have the new state at time t_i, the process can be repeated until a final time is reached (see Figure 1.8b).


m
t_0
$t_0 + \Delta t$
$\vec{q}_1^{\,t_0}$ $\vec{p}_1^{\,t_0}$ $\vec{q}_2^{\,t_0}$ $\vec{p}_2^{\,t_0}$ … $\vec{q}_N^{\,t_0}$ $\vec{p}_N^{\,t_0}$
Integral of motion
$m_i\frac{d^2\vec{q}_i}{dt^2} = \sum_{j\neq i} G\frac{m_i m_j}{q_{ij}^3}\vec{q}_{ij}$
$\vec{q}_1^{\,t_0+\Delta t}$ $\vec{p}_1^{\,t_0+\Delta t}$ $\vec{q}_2^{\,t_0+\Delta t}$ $\vec{p}_2^{\,t_0+\Delta t}$ … $\vec{q}_N^{\,t_0+\Delta t}$ $\vec{p}_N^{\,t_0+\Delta t}$
$t_0 = t_0 + \Delta t$

(a) **(b)**

Figure 1.8: Graphic (a) and Schematic (b) representations of a numerical integrator.

Time step size and complexity

The size of the time step is a decision variable. Depending on the problem to be solved, a different value has to be chosen to satisfy certain conditions. A large value of the time step will lead to faster results, as the number of times that the equation of motion needs to be solved is smaller. However, a large time step size is a coarse approximation of the continuous phase space, and therefore the results will be less true to the physical world. In contrast, a small time step size will lead to more accurate results, as it represents a better approximation of the continuity of the trajectory, at the cost of computation time. Finding the right balance between accuracy and computation time is a problem-dependent issue. It will also be the main focus of Chapters 4 and 5 in this thesis. Whereas for simple experiments low accuracy results may suffice, for most cases in astrophysics a good accuracy is required to ensure that the obtained results are true to the actual behavior of celestial bodies.

Despite the time-step size being one of the main parameters that determines computation time, the number of bodies in the system is also a main contributor. In Subsection 1.1.4 we defined complexity as the number of operations required per time step to integrate the system. As N increases, so does the complexity, leading to radically more expensive computations.

Numerical errors

In a numerical simulation, there are many sources of errors. Those will drive our simulation away from the truth. Firstly, we have previously talked about discretization errors. Those appear from the assumption that the trajectory can be made into discrete pieces, whereas in reality, it is continuous. It is directly related to the time-step size. The larger the time-step size, the larger the error incurred.

Secondly, the simulation will suffer from round-off errors. Those refer to the error caused by the limited precision of the computer (Boekholt & Portegies Zwart (2015b)). Most algorithms are by default limited to 15 significant digits to store the solutions, leading to an accumulation of round-off errors at each time step. This error grows with the number of integration steps. Unfortunately, this means that reducing the discretization error (by reducing the time-step size) can lead to an increase in the round-off error.

Additional sources of error will appear depending on the specific configuration of the integrator. We will describe different types of integrators and their implementations. We adopt the denomination of unphysical solutions for those cases in which the errors have grown to a point in which conservation laws are no longer fulfilled and therefore the simulation does not adhere to the laws of physics. In those cases, the simulations are no longer useful for most scientific purposes.

Types of numerical integrators

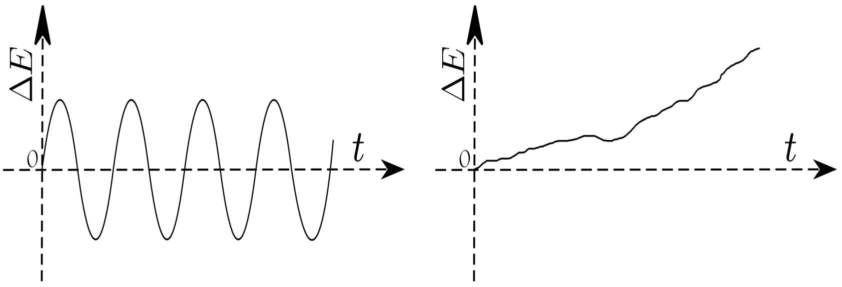
Numerical integrators have been developed and optimized for specific applications. The large range of astronomy problems has also led to a variety of integrators. Here we will review some of the most relevant types of codes used to solve the N-body problem.

The simplest type are pure codes. These do not include any physical parameters that need to be chosen except for the time-step criterion (Portegies Zwart & McMillan (2018)). Examples like `Ph4` and `Hermite` will be used in this thesis. Another type are direct N-body codes which include additional parameters to speed up the code or reduce numerical errors. The complexity of these codes scales with N^2. An example of this category is a code specially suitable for the long-term integration of the Solar System; Wisdom-Holmann integrator (Wisdom & Holman (1991)). This code belongs to a special type of integrators denominated *symplectic codes*. Because of their importance for this thesis, they deserve a more in-depth explanation. Finally, in order to reduce the computation time in certain cases, there are approximate methods. For example, tree codes such as the `Barnes-Hut` tree scheme (Barnes & Hut (1986)) assume that only particles close to a given one will contribute to its gravitational potential and the far away particles can be grouped into a single effect. This code saves computation time as its complexity scales with $N\log(N)$.

Symplectic integrators

In Hamiltonian systems, the solutions to the equations lie on a symplectic manifold in phase space. A symplectic integrator is one in which the solution resides on the symplectic manifold. A detailed mathematical description of the definition of symplecticity can be found in Sanz-Serna (1992).

We have talked about the numerical errors that appear in a simulation. Instead of the energy being perfectly conserved, discretization error leads to a linear drift in the energy error over time. In symplectic integrators, the energy error is different than zero, but instead of drifting (see Figure 1.9b), it oscillates around the zero value as in Figure 1.9a.



(a) Symplectic integrator **(b)** Not symplectic integrator

Figure 1.9: Difference in the evolution of the energy error for a symplectic integrator (a) and a regular integrator (b).

Because of this property, they are specially suitable for the long-term integration of planetary systems, as in Rein et al. (2019). In this case, the integrators divide the Hamiltonian into two different parts: one that contains the planetary motions around the central star, and a second one describing the interactions planet-planet.

1.1.6 Chaos

A fundamental characteristic of the N-body problem is its chaotic nature. In a chaotic system, a small perturbation to the initial condition grows exponentially with time. In Figure 1.10, we show how a small change in the position of one particle at the beginning of the simulation can lead to a completely different output. Similarly, numerical errors accumulate during a simulation and can cause the final solution to diverge from the real one. For that reason, for chaotic systems, the necessary precision increases exponentially with time (Srivastava et al. (1990)). In order to prevent the accumulation of numerical errors, Boekholt & Portegies Zwart (2015b) design an integrator with arbitrary precision. This integrator allows for minimizing round-off errors and in converged solutions, discretization errors. Then, it can be compared to other commonly used integrators. They tested their integrator (`Brutus`) against other commonly used ones on the Pythagorean problem (a special case of the democratic three-body problem). Their findings show that only about half of the solutions give accurate results compared to the ones obtained with the accurate integrator.

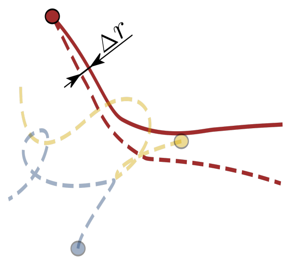


Figure 1.10: Representation of chaos in a three-body problem.

Chaos represents an important challenge for the simulation of N-body systems. Its effect on our methods will be inevitably present in the next chapters.

1.1.7 General assumptions

We have mentioned at the beginning of this chapter that we will not consider General Relativity (GR) in our experiments. Our assumption remains valid as long as we ensure that classical dynamics dominate in our system. For example, when talking about planetary systems, Mercury should not be included as its dynamics are dominated by the general relativity effects caused by its proximity to the Sun. In fact, its anomalous rate of precession was discovered in 1859 but could only be explained by Einstein's general relativity in 1915 (Yahalom (2022)). For other bodies orbiting further away from the Sun, the GR effects become more subtle and can in many cases be considered negligible.

In this work, we consider bodies as point masses. By doing so, we ignore the radius of the body and assume that all the mass is concentrated on its center of mass. Additionally, when looking at planetary systems and star clusters, we ignore stellar evolution and assume that the mass of the star remains constant. Similarly, we ignore effects such as radiation pressure which is mostly relevant for small objects, as shown in examples by Mignard (1982) and Belkin & Kuznetsov (2021).

1.2 Part 2. Machine Learning in Astronomy

The field of computational astrophysics is in constant need of faster and more efficient methods to adapt to the increasing complexity of the simulations performed. Numerical integrators have been developed for decades and are currently optimized for different problems. Despite the work put on perfecting these methods, there is a limit to what they can achieve.

With the fast growth of the field, many new Machine Learning (ML) methods have been created in the past years. Those methods cover a large range of applications from classification tasks to optimal control problems. In Figure 1.11, we show a simplified classification of ML methods. Supervised learning encompasses those methods in which labels are available in the training dataset. This means that for the training data, we know the "real" solutions and the networks will train on learning those. There are two main types of supervised learning algorithms: classification tasks, in which the output is one of the different classes available, and regression, where the output is a rational number or an array of rationals. In contrast, unsupervised learning does not involve the correct solution, but instead focuses on finding patterns in the data. An example is clustering methods, in which the data samples are grouped according to certain characteristics. Another example is that of dimensionality reduction algorithms, in which the goal is to reduce the size of the data while preserving qualities of the data. Finally, the third type of ML is Reinforcement Learning (RL). In this case, an agent is used to learn to make decisions given a certain situation. We will dive deeper into these types of methods in Subsection 1.2.2.

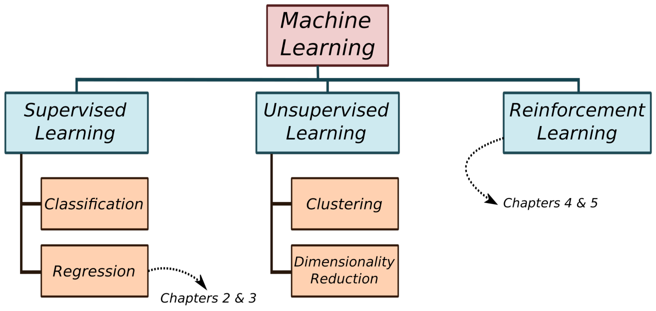


Figure 1.11: General classification of the different types of Machine Learning methods.

Despite the exceptional interest that the field has generated, most of those new methods are usually tested for the solution of simple equations or for overly simplified test cases. There are many possible applications of ML methods to scientific fields, but identifying the transferability of those methods from simple applications to complex cases such as the ones found in computational astrophysics is not trivial. We have explored many such topics and will show a brief description of many interesting applications in Subsection 1.2.3. However, this thesis focuses on two specific topics: physics-aware neural networks (Chapters 2 and 3) and Reinforcement Learning (Chapters 4 and 5). In the next subsections, we will aim to get a fundamental understanding of those two topics.

1.2.1 Physics-aware neural networks

Artificial Neural Networks (ANNs) are a type of supervised Machine Learning algorithms in which a set of inputs is used to obtain a desired output. ANNs are inspired by the function of biological neural networks. Its structure is formed by neurons, generally organized in layers, and connected to one another by nodes. The layers are usually divided into an input layer, an output layer, and hidden layers (as shown in Figure 1.12). The simplest type of ANNs are called Multi-Layer Perceptrons.

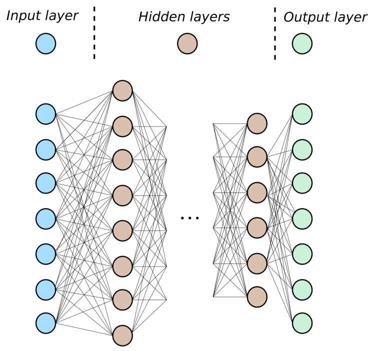


Figure 1.12: Simplified schematic of a basic Multi-Layer Perceptron.

The nodes connecting neurons carry a value denominated weight (W) that multiplies the value from the neuron in the previous layer. Additionally, a bias (b) is added to each layer. It is common to use an activation function f in the neurons to add non-linearities and expand the capabilities of the network. Starting from a set of inputs x, the values at the next layer (z) are calculated as

$$z = f\left(b + \sum_{i=1}^{n} x_i W_i\right) \tag{1.10}$$

for a layer with n neurons.

Both weights and bias are values that are chosen to make the network achieve a certain task. This process is not done manually but through a process denominated training. During training, the outputs achieved by the neural network with a certain set of trainable parameters (weights and biases) are compared to the desired output associated with an input (label). The difference between the desired output and the obtained one is denominated loss function. The number of layers and the number of neurons per layer are part of what is called *hyperparameters* and are values to be chosen in advance.

Neural networks have become extremely popular and many studies are being done to improve the training, activation functions, and hyperparameter optimization. Additionally, there are many studies working on their application to scientific fields. When applying neural networks to physical systems, it is important that the results adhere to the laws of physics. Since neural networks are mere statistical algorithms, physics are not enforced. Therefore, Raissi et al. (2019a) designed the first type of neural networks that incorporated physical knowledge; Physics-informed Neural Networks. Since then, the field has grown at a rapid pace leading to a wide range of algorithms that in one way or another include some physical knowledge.

We can currently divide these algorithms into two different types depending on whether the physics are added as a soft or hard constraint. An example of the former is physics-informed neural networks (PiNNs) as the physical knowledge is incorporated into the loss function. This means that the physics act as a regularization term during the training, but during inference there is no assurance that the output will adhere to the given constraint. In contrast, some algorithms incorporate physical knowledge into their structure. In this case, the physics represent a hard constraint. Examples of this are Hamiltonian Neural Networks (Greydanus et al. (2019a)), SympNets (Jin et al. (2020a)), but a more detailed study about these networks is performed in Chapter 3.

Each type presents advantages and challenges of its own. In what concerns PiNNs, they showed that they could achieve better performance than regular neural networks with a smaller database. Since generating datasets is generally a computationally expensive task, it resulted in more efficient studies. Additionally, they managed to achieve better extrapolation capabilities in some cases. However, as mentioned before, they did not ensure that the results adhered to physics laws, which in some other cases led to unphysical results. After their initial popularity and many applications (Farea et al. (2024)), many studies have arisen to solve some of the initial challenges and create more robust algorithms. However, there is still progress needed for them to be able to learn intricate physical phenomena such as multi-scale and chaotic behaviors (Antonion et al. (2024)).

Neural networks that incorporate physics into their architecture have not experienced the same popularity as PiNNs, but their results are extremely promising. There are multiple types and each implementation is different. However, a common challenge is the complexity added by creating network structures that represent some physical phenomenon. By creating these structures, the problem becomes more rigid and their use to different problems can be challenging. These challenges will be addressed in Chapter 3. The main advantage of this method is that their results show a better adherence to the physical laws acting upon a system, and therefore better extrapolation to unseen data.

1.2.2 Reinforcement Learning

The second part of this thesis deals with the use of Reinforcement Learning (RL) algorithms. In RL, an agent is trained to learn how to choose between different actions in order to fulfill a desired task in a dynamic environment.

The agent and the environment interact with each other via the action, the state, and the reward function. The action represents the choice made by the agent and that will be implemented in the environment. The state is a representation of the environment that is fed into the agent for it to make a choice. Finally, the reward is the function that quantifies how well the action has worked to achieve the desired goal. A visual representation is presented in Figure 1.13. The agent makes the decision about the action to take based on a policy, which is a decision-making function to determine the control strategy of the agent.

There are multiple RL algorithms available, and they can be generally classified into

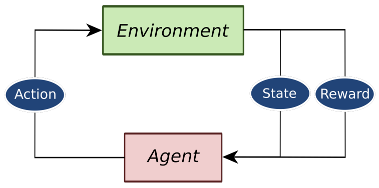


Figure 1.13: Interaction between the agent and the environment in RL algorithms .

model-based or model-free RL. The first use experience to construct an internal model of the transitions and outcomes of the environment (Dayan & Niv (2008)). The latter uses experience to learn directly the state and action values or policies without estimating or using a world model. The latter ones represent simpler algorithms, which becomes useful for preliminary experiments like the ones in this work.

RL algorithms have become widely used for optimal control problems such as self-driving cars, industry automation (Kiran et al. (2022)), finance and trading (Hambly et al. (2023)), neurobiology (Dayan & Niv (2008)), and healthcare (Coronato et al. (2020)), among others. Many RL applications for science problems focus on their use to create engineering tools that can automatically choose some parameters to create better-performing systems. Many examples can be found in the field of fluid mechanics and heat control (Novati et al. (2021); Viquerat et al. (2022)). In astronomy, most applications of RL focus on the optimal control of telescopes to improve the observation strategy (Jia et al. (2023)). However, some techniques have also been applied to the prediction of solar flares. In this thesis, we will focus on the use of RL techniques for the improvement of computational tools.

1.2.3 Applications and challenges

We have identified different applications for Machine Learning in the context of the integration of the gravitational N-body problem. As we have only fully developed two main possibilities (physics-aware NNs and Reinforcement Learning), we dedicate this Subsection to elaborate on some of the relevant ideas identified, with our view on their applications and challenges.

Surrogate for integration

The most straight forward application for ML in the integration of a system of N bodies is the replacement of the integrator with a neural network that predicts the state of the system at time t_{i+1} using the state at time t_i as an input. This idea is being applied to multiple fields but with a special focus on fluid mechanics and weather forecasting.

The recent popularity of ideas such as Physics-informed Neural Networks (see Subsection 1.2.1) has led to a variety of neural networks that aim to incorporate physical knowledge into their structure. These neural networks, but also those without physical constraints, can be applied to many problems in astrophysics that deal with time evolution such as stellar evolution, origin, and evolution of planetary systems, star cluster dynamics, etc.

Each of these networks presents advantages and disadvantages of their own, but additional challenges appear when applying them to the gravitational *N*-body problem instead of the standard test cases. Chapter 2 presents a detailed explanation of the advantages and challenges of a variation of PINNs (Hamiltonian Neural Networks), when applied to a planetary system. Although the introduction of physical constraints into the neural network leads to a better adhesion to the dynamics and better extrapolation capabilities, the training of the networks becomes more challenging. New types of neural networks that include physical knowledge are being developed to overcome those challenges. Chapter 3 shows a new type of neural networks with a structure that resembles that of an integrator. Thanks to their structure, they lead to better extrapolation capabilities and adhesion to the physics.

There are two main challenges that appear when applying neural networks to the *N*-body problem. The first one is the accumulation of errors made by the neural network. In a chaotic system, this accumulation of errors can lead to large deviations from the ground truth on short time scales. Currently, the accuracy achieved by neural networks cannot compete with that of traditional integrators, making them inadequate for their use in chaotic systems. The second challenge refers to the fixed size of the input. The state of the system is used as an input to the neural network. This size remains fixed for a trained network, which means that it is not possible to change the number of bodies in the system without retraining. This limits the generalization possibilities of a trained network in astronomy.

Predictor for the secular evolution of a system

Similarly to the previous case, neural networks can be used to predict the future state of the system knowing some initial one. In this case, instead of predicting each integration step, the neural network can be used in a planetary system scenario to predict the secular evolution of the keplerian elements of the planets. In a planetary system, most of the evolution of a planet is an elliptical movement around the central body, but close encounters with other planets can lead to changes in the orbit.

By using the neural network only once per orbit, we prevent the accumulation of errors at each integration step. However, setting up the problem in a way that the search space is covered is not straight-forward. In order for the algorithm to be generalizable, the training samples need to be representative of the different scenarios that may appear. This becomes challenging due to the large range of possible configurations of planetary systems. Additionally, since each planet has a different orbital period, it is not clear how to choose a time at which the secular changes are calculated.

Chaos predictor

Understanding how chaotic a system is before running the simulation can be beneficial to choose the right time step size or, in the case of integrators such as `Brutus` (Boekholt & Portegies Zwart (2015a)), the size of the mantissa. The Lyapunov exponent can be used as a measurement of chaos.

A simple neural network can be used to predict the Lyapunov exponent before the simulation. The challenge arises from the training. In chaotic problems, similar initial conditions can lead to radically different outputs. Solving this problem may imply a different choice of inputs other than cartesian coordinates, training a network that is tuned to highly sensitive solutions, and having a large enough dataset to cover a broad range of cases.

Generation of initial conditions

In astronomical simulations, it is often challenging to find a set of initial conditions that is representative of real systems. For example, there are currently some assumptions that can be applied for the initialization of star clusters such as fractal cluster models (Goodwin & Whitworth (2004)) and several density models to choose their masses (see Plummer (1911)). However, some of those models are decades old and make simplifying assumptions. Additionally, in systems in which there is gas present, the choice of initialization becomes even more challenging.

There are multiple ML methods that are growing in popularity for their use as generative systems. Generative Adversarial Networks are a common example. Those could be applied to the problem of finding a set of initial conditions that fit observations but contain all the relevant information to run the simulation.

This problem comes with the challenge that the system must comply with physical laws such as conservation of energy and angular momentum. The use of neural networks for this problem requires the introduction of physical constraints during the training procedure.

System representation

Cartesian coordinates may not be the optimal choice for the representation of the state of a system when using neural networks. Their large variability may make the training more challenging depending on the problem to be solved. On the other hand, keplerian coordinates do not combine well with neural networks. This is due to the combination of linear parameters, such as the semi-major axis and the eccentricity, and periodic ones, such as the angles (see Subsection 1.1.2). The use of sines and cosines in the inputs many times benefits from specific activation functions, which might be incompatible with the linear inputs.

Both Cartesian and Keplerian coordinates have advantages and disadvantages depending on the problem. However, there is currently no state representation designed specifically for its use in combination with neural networks. Finding a state representation that is optimal for a specific problem could make a substantial difference in the final performance of an algorithm.

Autoencoders have been designed to compress information while retaining representative features. This compression can be combined with any of the aforementioned applications to improve the performance of the trained networks.

Choice of integration parameters

The choice of integration parameters can be thought of as a control problem. Although some of these parameters can be chosen a priori, others should be adapted dynamically during the simulation to adapt to fast-changing conditions.

Reinforcement Learning can be used during the simulation to choose parameters such as the time-step or time-step parameter (depending on the chosen integrator). Additionally, it could be used to make other choices such as the integrator at each step, or even individual choices for each particle. We study two applications of reinforcement learning for the choice of time step in Chapters 4 and 5.

The case of `Brutus`

A specially interesting case is that of `Brutus` integrator (Boekholt & Portegies Zwart (2015a)). This method allows a free choice of accuracy and mantissa to ensure that the final solution is free of numerical errors to a given accuracy, i.e., the simulation is converged. In order to find a converged solution, it is necessary to run the code multiple times with different choices of accuracy and mantissa until the results do not change significantly. This leads to extremely computationally expensive runs, specially for chaotic problems.

A RL algorithm that can choose those dynamically would be extremely useful to avoid the need for the repetition of the simulation multiple times, therefore leading to a substantial speed-up of each simulation. The main challenge in this case arises from the fact that once the mantissa has been decreased to a certain number of decimal places, it cannot be increased without incurring round-off errors. An alternative option to speed up this convergence process using ML would be to train a neural network to predict a priori what the simulation parameters should be, even thought in this case they would not be changed dynamically during the simulation.

Grouping algorithms

The integration of star clusters can be computationally expensive due to the large number of bodies present. A speed-up of this integration is sometimes achieved by using integrators that group stars depending on the strength of the effect on each other. This is the example of tree codes (see the different types of numerical methods in Subsection 1.1.5). This grouping can be done by taking into account the distance between them, their relative velocity... but this process is expensive and might lead to sub-optimal groupings due to the assumptions made.

Clustering algorithms could be used to find optimal groupings at different moments of the simulation.

Graph Neural Networks

A type of neural networks that has not yet been mentioned is Graph Neural Networks (GNNs). It is specialized for tasks in which the inputs are graphs. In these networks, the graph nodes exchange information with their neighbors. It is specially well-suited for chemical networks where atoms form the edges of the network. Analogously, these networks could be used for the simulation of a system of N-bodies, where each node represents one of the particles in the system.

1.3 Thesis summary

This thesis compiles the experiments performed on the use of Machine Learning techniques for the simulation of N-body systems. For this work, we have simulated different astrophysical systems and performed the training of machine learning methods on two different categories: physics-aware Neural Networks and Reinforcement Learning.

In Chapter 2, we take the example of a planetary system with a large number of small bodies and apply neural networks to replace parts of the integration with the goal of speeding up the simulations. We use a specific integrator - Wisdom Holmann integrator- that separates the contributions to the acceleration of a body in two terms: the ones by the central body and the perturbing ones by the other bodies in the system. We adapt two types of neural networks to this case: a deep neural network and a Hamiltonian neural network. We identify the challenges of using neural networks in a complex case in astrophysics. While the standard neural networks are easy to set up, they do not lead to results that adhere to the laws of physics. They also do not extrapolate well to unseen data. We observed that small prediction errors accumulated and grew, leading to a drift in the energy error. In contrast, Hamiltonian Neural Networks managed to find solutions that adhere better to the physics and are capable of extrapolating for longer periods of time. However, there were also major challenges. For example, we observe that the large orders of magnitude difference in the masses of the bodies makes the application of Hamiltonian networks extremely challenging. As the calculation of the accelerations is done using automatic differentiation, the large differences in mass created equally large differences in the gradients. As the physics are embedded in the architecture, normalization of the inputs is not possible without breaking the physical relations. These facts contributed to making the training problematic. Additionally, with both networks, we encounter other challenges due to the chaotic nature of the problem. In a chaotic system, prediction errors by the neural networks accumulate and grow exponentially, leading to unphysical solutions. To address this problem, we create a hybrid method that evaluates the network prediction at each step. If the prediction does not satisfy a requirement of accuracy, the calculation is repeated using the analytical equations. Thanks to this method, the integrator can make use of neural networks to speed up the simulation while ensuring robustness in its accuracy.

With the results in Chapter 2, it became clear that the simulation of the N-body problem required more sophisticated machine learning methods to ensure the adherence to the conservation laws. Therefore, in Chapter 3, together with Philipp Horn, we compare different implementations of neural networks with structure-preserving architectures and develop a new type of neural network that represents a generalization of many of those types. SRNNs, SympNets, and HénonNets are three common cases of neural networks that implement hard physical constraints into their network structure. Those networks can be thought of as specific cases of a new type of structure-preserving neural networks: Generalized Hamiltonian Neural Networks (GHNNs). The performance achieved with those neural networks in solving Hamiltonian systems is compared for different experiments, of which the 3-body problem is shown in Chapter 3. The comparison between these networks is not straight-forward as those rely on different implementations. Therefore, their hyperparameters are chosen for their prediction time to be comparable. The results of the experiments show that the performance of the networks depends on the experiment. For simple cases such as the single and double pendulum, networks that included a larger number of simple updates dominated over those with fewer updates. In contrast, in the 3-body problem, networks with more complex updates (such as Deep HénonNets) achieved a better performance. In any case, MultiLayer Perceptrons, a simple type of physics-unaware neural network, showed the worst extrapolation capabilities compared to the physics-aware ones. It is concluded that the added symplecticity in the structure-preserving neural networks resulted in better performances, both inside and outside the training data.

After studying the advantages and disadvantages of replacing parts of the integration with neural networks, we moved our focus to the use of Machine Learning techniques to choose simulation parameters. When setting up a simulation, lack of expertise makes it common to use the default parameters of the integrator. This can lead to the solutions not being accurate enough or being too computationally expensive. Thus, in Chapter 4 we explore the idea of using Reinforcement Learning techniques that will eliminate the need for expert knowledge by choosing an important simulation parameter for us: the time-step parameter. We apply this method to the chaotic 3-body problem. In this case, there are moments in which the three bodies are close to each other, and the interactions will heavily determine the outcome of the simulation. In these moments, we want the integrator to choose smaller time-step sizes to capture those interactions in detail. In other moments in which the bodies are far from each other, we want the time-step size to increase to save computation time. The time-step parameter directly scales the time-step size chosen at each moment. By allowing this parameter to change dynamically to adapt to the needs of the simulation, we obtain an optimal choice that balances accuracy and computational effort at each time step. We explore the extrapolation of this trained algorithm to other integrators and determine that it can be used without retraining for similar algorithms. Additionally, the method setup can be easily extrapolated without major development changes.

In order to prove the generalization capabilities of the method created in Chapter 4, in Chapter 5 we apply a similar algorithm to a more complex astrophysics problem: the evolution of a star cluster in which some stars have planetary systems orbiting around them. In this case, we need to use different integrators for the star cluster and the planetary system to adapt to the orders of magnitude difference in their scales. Then, those two different subsystems are linked using the `Bridge` method from `AMUSE`. The interaction time between both integrators is a fixed parameter that has to be chosen manually before the simulation. We denominate it the bridge time-step size. Instead of choosing a value and keeping it fixed throughout the simulation, we apply our RL algorithm to allow it to change dynamically to find the optimal choice that balances accuracy and computation time. We find that our algorithm outperforms all of the current options and can adapt to different initializations. Due to the chaotic nature of our method, finding robust baselines proves to be problematic. Therefore, we base our comparisons on the energy error incurred by the simulations. Knowing that our system is highly chaotic, we want to create a method that is robust against suboptimal choices of the RL algorithm. Therefore, we create a hybrid method, similar to the one in Chapter 2, that evaluates the prediction and reduces the time-step size if considered inadequate. We find that this method leads to improvements of even orders of magnitude in the energy error without a major increase in computational effort.

1.3.1 Future work

We have identified in Subsection 1.2.3 additional ideas of implementations of ML methods to the gravitational *N*-body problem that we considered potentially interesting. Additionally, despite the promising results achieved with the new methods shown in this thesis, throughout our work, we have found some common challenges that should be addressed in the future. Some of those were common to many of the methods tested.

First of all, we found that it is a common practice to use the state of the system as an input to neural networks. In the case of physics-aware neural networks, this is even a strict requirement. However, this leads to a critical generalization problem: the number of inputs changes with the number of particles present in the system. A network trained for a system of 3 particles cannot be directly applied to one with 4 or more particles present. Finding a solution to this problem is not straightforward, specially as the evolution of physics-aware neural networks relies on hard constraints imposed on their architecture. This problem becomes specially serious for systems with a large number of particles, such as star clusters. In this case, it is impractical to train a neural network for each possible number of bodies. We propose a potential solution in Subsection 1.2.3 that could be further studied. In Chapter 5, we find a different input representation that works for that specific problem and is independent of the number of bodies in the system.

Continuing with the choice of inputs, Cartesian coordinates are the preferred choice in most studies involving the evolution of celestial bodies with neural networks. This option might not always be optimal. In certain cases, Keplerian coordinates are the preferred choice for the interpretability of a problem. However, those result problematic if used as inputs to the neural networks due to their mix between linear variables of different orders of magnitude and periodic ones. Currently, those are the main two choices to represent the problem and provide the ML algorithm with the required information of the system. Finding different system representations that are specially well suited for their combination with neural networks should be a priority for those studying this problem.

Moving to a different challenge that has been present in most of this work, we find that chaos is a major obstacle to obtaining accurate solutions. Prediction errors in the case of neural networks and suboptimal choices of actions in the case of Reinforcement Learning are specially problematic in chaotic systems as those errors accumulate and grow exponentially in time. The current performance of these algorithms does not generally allow for long-term simulations (with the exception of the method shown in Chapter 5). We create a hybrid algorithm to increase the robustness of the methods in Chapters 2 and 5, but in order to achieve truly long-term integrations within a reasonable energy error, the performance capabilities of the ML methods need to drastically improve. With the current popularity of these topics and the potential shown in this, and many other studies, it is reasonable to believe that sooner rather than later these methods will reach the required precisions to be used in chaotic problems.

Regarding problem-specific challenges, structure-preserving neural networks are still not specifically designed for astrophysics problems in which the masses of the bodies range orders of magnitude. Normalization is a common technique that can minimize this problem, but it is in many cases incompatible with the physics-based structure of those types of networks. By adding hard constraints into the networks, it becomes harder to adapt to certain complex problems. These should be taken into account when designing new types of neural networks for physical problems.

Finally, there is work to be done to continue the projects on this thesis. The methods explored are applied to simple cases found in astrophysics. Extending their capabilities and implementations to a broader range of applications is the most interesting future direction. Including the ML methods as a part of the frameworks used for astrophysics simulations can help to better understand the benefits and limitations of using Machine Learning in computational Astrophysics on a regular basis.

1.3.2 Computational tools

Most of the code has been created for the purpose of this thesis and can be found for each individual chapter. The astrophysics simulations have been developed using `Python`. I have made extensive use of libraries like `AMUSE` (Portegies Zwart et al. (2013)) and `ABIE` (Roa et al. (2020)) for the initialization of the planetary systems, triples, and star clusters, and for their integration in time.

For the Machine Learning algorithms, I have made use of `TensorFlow` (Abadi et al. (2015)), `PyTorch` (Paszke et al. (2019)), and additional machine learning packages such as `Scikit Learn` (Pedregosa et al. (2011)), `Gym` (Brockman et al. (2016a)), and `pyDOE`.

All simulations and experiments for Chapters 2, 4, and 5 have been run on an AMD Ryzen 9 5900hs computer, and the computation times incurred can be found for each chapter.