Hybrid Modeling of the N-Body Problem with Applications to Astrophysics

by

Åsmund van Brussel Synnevåg*

Master Thesis in Applied and Computational Mathematics

Department of Mathematics

University of Bergen, Bergen, Norway

June 2022

* Bergen, Norway. Email address: asy012@uib.no

Abstract

Over the last years, the field of hybrid modeling, the concept of combining data-driven machine learning models and numerical solution methods to simulate a physical system, has seen an immense increase in research. This new paradigm within modeling uses its predictor capabilities from neural networks to uncover the unknown physics of the underlying system, and bridges these hidden physics with the strong mathematical foundation of numerical integrators and the governing equations of the physical system. Even though hybrid modeling is being introduced into many different fields of research, one field which so far has lacked a more detailed investigation is the field of n-body problems. This field also represents the class of non-linear systems of O.D.Es. with symplectic structure. The n-body problem has through the ages been a source of countless scientific discoveries and is still of great interest to this day. As an application, this thesis will look at the problem of n-body dynamics of planetary motion, more specifically, the simulation of the main celestial bodies of the Solar System. As the first to create a hybrid model for the n-body problem of such size ($n > 3$), this thesis will show, through a series of important observations and modeling approaches, that hybrid modeling of the n-body problem can be achieved. The results will also show that the subsequent model can improve the results of a standard physics-based model, the standardized modeling approach for the n-body problem. To the best of the author's knowledge, this thesis will also be the first to present a pure data-driven model for predicting the orbital motion of planets in our Solar System.

Acknowledgements

I would like to thank my supervisors Jakub Both, Kristian Gundersen, and Guttorm Alendal for following up on my work, helping me through obstacles, and giving constructive feedback and encouragement along the way. Firstly, I would like to thank Jakub for letting me explore fields of research in line with my own personal interests; the work has been exciting, challenging, and highly motivational. I would also like to thank him for introducing me to the field of Machine Learning, and always taking time off to help me in my work. Secondly, I would like to thank Kristian for giving his expert knowledge in the field of Machine Learning and helping me overcome difficult challenges. I would also like to thank Guttorm for his guidance through the research and writing process.

A special thanks goes out to Jan Martin Nordbotten for guidance in preliminary stages for the thesis and helping me discuss relevant fields of research and supervisors.

Finally, I would like to thank my fellow students and friends. Over the years at UiB they have created a wonderful study environment, and without their help and motivation through the periods of study, this thesis would not be possible.

Contents

1 Introduction

Over the last few decades, we have rapidly entered into the era of big data. With an exponential increase in available data within countless research fields, and the quality of data increasing (Cai, Zhu, 2015), it opens up more and more possibilities for all research to come. One field of research, which has greatly benefitted from big data, is the field of Machine Learning (M.L.) and data-driven models (D.D.M). Since D.D.Ms., as the name suggests, rely heavily on both quality and quantity of data, the introduction of big data has naturally made way for extensive research within this field. D.D.Ms. have already been shown to facilitate tasks within commercial application to the extreme and have shown to be forthcoming in obtaining new scientific knowledge and methods (Willard et al., 2020). Despite the persistent research within the field of D.D.Ms., we have only scratched the surface, and it is still viewed as a field of optimism and possibilities (Ford, 2018).

In general, D.D.Ms. are any M.L. models governed exclusively by empirical data, and thus, do not take into consideration the fundamental mathematical and physical principles determining any underlying problem. The model can be viewed as a self-constructing algorithm not directly bounded by human knowledge and perspectives. Also, these models do not require much processing power when first established. As remarkable as they may sound, these models do not come without restrictions. Firstly, as previously stated, these models can be immensely data hungry when being constructed and rely heavily on the quality of the labeled data. Thus, the model can in general only be as good as the data presented. For modeling complex physical, biological, and engineering systems, there is still a large data gap, which in many cases leaves us with a frail and divergent model (Raissi, Perdikaris, Karniadakis, 2017). Secondly, as these models omit the fundamental mathematics and physics of a given problem, the results of such a model are not as theoretically justifiable.

In contrast, Physics-Based models (P.B.M.), are models derived from the governing equations of the problem and appropriate numerical solution methods, and thus are theoretically well grounded. These models are used extensively within all field of science and engineering, where one seeks to explain and simulate a physical system. P.B.Ms. have also been fundamental in obtaining new scientific knowledge through the ages and are still a cornerstone for new scientific discoveries (Daw et al., 2017). Though its theoretic principles offer sound and intelligible results, this obligation towards the known physics is also one of its drawbacks. In the case of describing complex dynamical system (e.g., planetary motion, detailed

population growth, groundwater flow), especially, the P.B.M. is only a simplification of the real-world problem. As restrictions are made to the systems (e.g., number of variables, domain size), the unavoidable numerical errors (i.e., rounding errors and truncation errors), and the exclusion of yet unknown physics, results in the model not capturing the whole truth. Another flaw is also the slow and expensive computations that drives the model for complex problems.

A conclusion drawn from the previous statements finds that D.D.Ms. and P.B.Ms. individually will in many cases yield incomplete results. This gives rise to the idea a synergistic relationship between the presented models, a hybrid model. By combining the theoretically grounded P.B.M. with the unconventional algorithms of the D.D.M., one can arrive at a more accurate solution still abiding the law of physics. Also, by this synergy, a more inexpensive model can be created. This new paradigm within modeling has over the last years seen a rapid increase in research, and there exist already large amounts of published papers on this topic (Daw et al., 2017; Blakseth, Rasheed, Kvamsdal, San, 2022; Raissi, Perdikaris, Karniadakis, 2017; Willard et al., 2020). Early research, though limited to simplified and isolated problems, have already shown great promise for this modeling approach.

The goal of this thesis is to analyze the results of a P.B.M., D.D.M., and hybrid model, to see how the performances stand in comparison to each other for a specified problem. To obtain these results, each model must first be constructed and tuned to yield comparable results. The P.B.M. will be a simple model, i.e., using a lower order numerical integrator and solving a restricted system (see chapter 3). The D.D.M. will be constructed through a series of simplified scenarios to see which elements must be included to arrive at a model comparable to the P.B.M. (see chapter 4). The hybrid model, presented as a Hybrid-Physics-Data-Residual model (H.P.D.R.M., see chapter 5), will try to combine the previously mentioned P.B.M. and D.D.M. The P.B.M. will give an approximated solution to the problem, with the D.D.M. producing an additive residual term to compliment the P.B.M. solution. The conclusion of this work is based on extensive numerical implementations and investigations (see Appendix A.2). The main software development is built on the PyTorch framework (Paszke, et al. 2019).

The focus of this thesis will be the n-body problem for planetary motion, with the main problem being simulating the Solar System for all $n = 8$ planets (see chapter 2). The n-body problem (not only for the case of planetary motion) have been of key interest for scientist for centuries, and its research have has led to countless new theories and scientific discoveries, and is still studied to this day. The n-body problem for planetary motion describes a system fluctuating between

relativistic and classical mechanics. Implementing classical mechanics can result in a too simplified problem, yet, enforcing a relativistic approach may prove too convoluted and computationally expensive. The standard approach for simulating n-body problems is still by the use of P.B.Ms., and not much research has been done on the field in relations to D.D.Ms. and hybrid models. The most advanced work in D.D.M. for the n-body problem to date was completed by Breen, Foley, Boekholt, Zwart (2020), which gave evidence to support that D.D.Ms. can outperform P.B.Ms. for n-body problems Here, a D.D.M. for predicting a chaotic three-body system was constructed, and the system simulated was 2 dimensional with particles of equal mass and zero initial velocity. As this thesis tackles D.D.M. and hybrid model for the n-body problem for $n > 3$, to the best of the author's knowledge, it is the first thesis to construct and study such models for a system of this size. For the problem of thesis, a classical mechanics approach is used for describing the dynamical system. As mentioned, this approach may be deemed too simple for the true solution to transpire. This where the D.D.M. is implemented to uncover the unknown physics of the relativistic. For the pure D.D.M., the model should produce results describing both the known and unknown physics, though for the H.P.D.R.M., the D.D.M. should generate results expressing the unknown physics and a correction for the simplification and numerical errors of the P.B.M.

2 Mathematical modeling of the n-body problem

As the title suggests, this thesis will aim at hybrid modeling of the n-body problem. The application for the hybrid model in chapter 5 will be grounded in solutions to the n-body problem derived from Hamiltonian mechanics. Specifically, dynamical systems of planetary motions. The mathematical model presented in the following sections will be the basis for the numerical model in chapter 3 later used for solving the n-body problem. This numerical model will again be a basis for the main aspect of this thesis, the final hybrid model. Firstly, the general n-body problem will be presented from a physics-based point-of-view, together with some historical context. Later, a restricted n-body system of planetary motions will be described in general terms, which is the main system to be modeled in this thesis. Lastly, the n-body problem, both $n = 2$ and $n \geq 3$, for planetary motion will be derived mathematically with the use of Hamiltonian dynamics.

2.1 The n-body problem

In general, n-body problems are problems which describe the motion of n-many particles within a given dynamical system, where usually the particles interact under the influence of some type of physical force. These types of problems arise within systems of celestial bodies under the influence of gravitational forces, charged particles effected by electrostatic forces (Synge, 1940), amongst others. Its fields of application are vast, and by defining "particles" within the given system in specific ways, like viewing clusters of particles as a single particle, n-body problems can help simulate even more dynamical systems, e.g., describing star clusters (Heggie et al., 2003) and galaxy interactions (Renaud, Appleton, Xu, 2010). Due to its extensive applicability, the n-body problem has been of peak interest for astronomers, physicists, and mathematicians over the last several hundred years (Greenberg, 1990; Diacu, 1996). The theoretical study of the n-body problem was though limited to the coarse qualitative dynamics of the system, and to a lower number of bodies. This limitation has been somewhat lifted over the last decades due to the introduction of high-speed computers, and different solution and simulation methods (Greenberg, 1990). The study of the n-body problem has given way for much of the theoretical work within cosmology, solid state physics, differential equations, and potential theory (Greenberg, 1990).

Historically, the mention of the n-body problem dates back to the late 17th century, where it was first stated in Newton's *Principia* as an initial value problem of ordinary differential equations for celestial mechanics. This formulation of the problem did not include any gravitational forces acting upon planets, but Newton mentioned in the same publication that some interactive forces must be present (Diacu, 1996). Daniel Bernoulli completely solved the $n = 2$ problem in 1710, though a solution for the $n = 3$ problem would bogle the mind of mathematicians and physicist for the coming centuries. In 1885, in honor of King Oscar II of Norway and Sweden, a prize was established for whoever could find a convergent power series of the initial value problem stated in Newton's *Principia*, amongst three other questions. None of the papers submitted managed to find a solution, however, Henri Poincaré was awarded the prize for his exceptional contributions which bolstered our understanding of the dynamical system, known today as Hamiltonian systems (Diacu, 1996). This contribution later gave Poincaré the fundamentals for his world-renowned chaos theory and has inspired many other branches of mathematics.

2-body problem: For a closed 2-body system with constant magnitude of velocities and without perturbations, it is relatively easy to arrive at an analytical solution. As mentioned above, Bernoulli already solved the problem back in 1710. His version of the problem, a version which may be considered the easiest n-body problem, was constructed by assuming two point-masses with one mass being fixed and one free-moving (Winter, 1941; Diacu, Holmes, 1999). By this assumption, most 2-body problems can be reduced to what sometimes are referred to as multiple 1-body problems. These problems are somewhat more of a mathematical idealization than a real-world physics problem. This assumption leads to bodies moving towards an unmoving center, with only one force acting, an uncommon situation, except for special cases and controlled laboratory experiments.

Restricted 3-body problem: As in the 2-body case, one can reduce the problem by looking at negligible masses and fixed particles to find an analytical solution for a restricted 3-body system. Even for a 3-body problem without these restrictions a global solution can be found, in the form of a series expansion. This was done by Sundman, though it was restricted to the case of non-zero angular momentum (Wang, 1991) and could therefore not be generalized.

n $\geq$ 3-body problem: There has been a misunderstanding regarding the general n-body problem that is still spread today. As there exist multiple proofs for the non-integrability of the n-body problem (Bruns, 1887), these results have been interpreted by many as that the problem is unsolvable. These results only relate to one solution method, thus do not prove the problem unsolvable. As it turns out,

the n-body problem has actually been solved. Though Sundman failed to arrive at a generalized global solution, the problem of zero angular momentum was later solved when Wang (1991) published a paper with a global solution using series expansion for the general $n \geq 3$ - body problem. The problem with this solution, as mentioned in the paper, is that the convergence rate of the series is so slow that one does not obtain a useful solution (Wang, 1991). But even though the result of Wang makes the general n-body problem integrable, the slow convergence rate makes numerical methods the go-to method for solving $n \geq 3$ - body problems (for more information, see Diacu 1996).

2.2 n-body problem of planetary motions

Until now, the general n-body problem has been discussed, which is applicable within many fields. From here on out, even though many concepts are usable for other n-body problems, only the n-body problem from a celestial mechanics point of view (i.e., planetary motion) will be considered.

As mentioned in the previous section, it is hard to describe the motions for an arbitrary $n \geq 3$ -body problem and arriving at a useful solution. The difficulty is also increased when looking at bigger, unrestricted $n \geq 3$ - body systems. Here, from what looks like a stable system, from a mathematical point of view, chaos can arise, where small perturbations can propagate and accumulate to large, chaotic behaviors (Alligood, Sauer, Yorke, 1996). In specific applications, the chaotic behavior of a n-body system may be the core interest to examine; however, this is not the case for this thesis, as the system to be modeled, described later in this section, has a Lyapunov time (i.e., the timescale for which a dynamic system can be predicted) much larger ($T_L = 2 \cdot 10^6$) (Hayes, 2008) than the time simulated. By the restrictions applied to system throughout this section, the system will be kept stable and, as a secondary effect, the possibility of chaotic behaviors is prevented.

As previously stated, when describing a problem of planetary motion, as for all other physical systems, some simplification must be made. The first part to consider is the domain itself. The domain considered is reduced to roughly the size of orbit the outermost celestial body one wants to observe, and the influence from all bodies outside the domain is being neglected. At this point, a n-body system may look something like this (see figure 1):

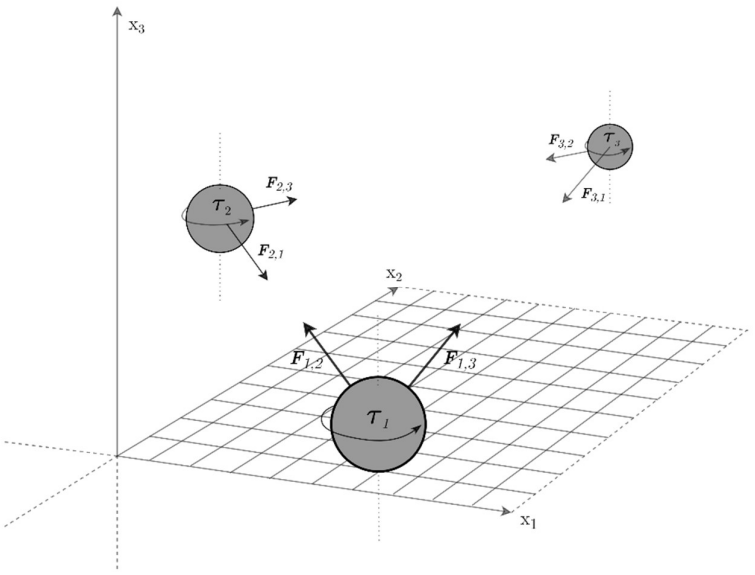

Figure 1: Example of 3-body system.

with $\boldsymbol{\tau}_i$ being torque of planet i, and $\boldsymbol{F}_{i,j}$ being the gravitational force planet i exert on planet j. For the case of this thesis, the domain will be restricted by the orbit of the outmost planet of our Solar System (i.e., Neptune), with the origin being located at the center of mass for the Sun. One could consider the solar system barycenter (i.e., center of mass of $n \geq 2$ bodies, which all bodies orbit) as the origin, but for simplification purposes, the Sun, as mentioned, is chosen.

The next simplification is the exclusion of all non-planet celestial objects (e.g., asteroids, dwarf planets, moons), but the Sun. This yields a $n = 9$ – body system (i.e., the Sun, Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune). For this problem, the inclusions of the smallest and outmost planets may not be necessary (e.g., Neptune, Mercury), as big contrasts between distances, masses, and orbital periods result in these bodies having a relatively small influence on the system. The magnitudes of the contributions by these bodies may also increase rounding errors in solutions of the total system. The inclusion of these planets, and the exclusion of larger collections of bodies (e.g., the main asteroid belt), may also contribute to imbalance in the conservation of energy of the system, as will later be shown, the dynamics of the system (i.e., a Hamiltonian system) should conserve the systems total energy (see eq. (13) section 2.3). Even though the exclusion or inclusion of different celestial objects will affect the results, the assumption is made that the restricted system constructed in this thesis is sufficient for describing its real-life counterpart.

Another simplification is to look at the celestial bodies as point particles, consequently the assumption is made that the celestial bodies have no volume. A result of this simplification of this is that one can neglect the spin of the celestial bodies, as point particles do not have spin. This leads the torque of each body to be omitted, and figure 1 is reduced to figure 2:

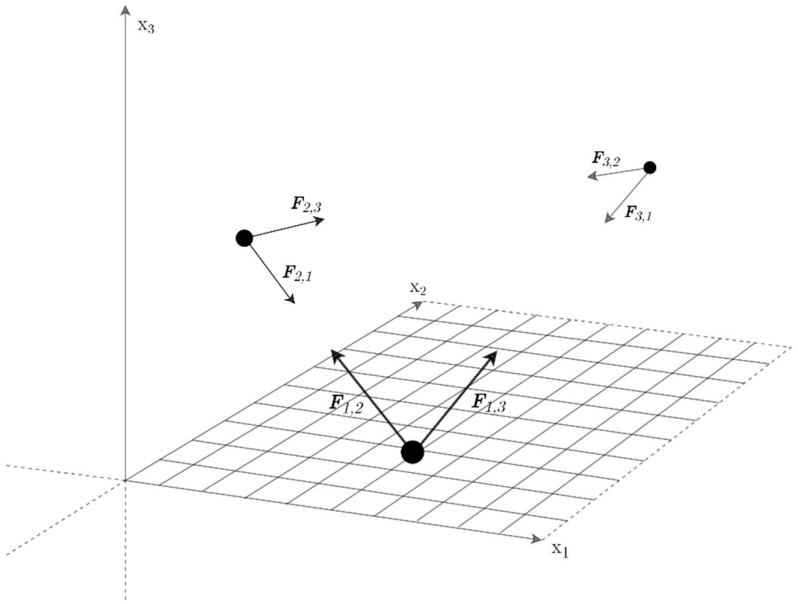

Figure 2: Example of simplified 3-body system.

By this assumption one also assumes that the torque of the celestial bodies has negligible effect on the total system. As volume is not a parameter included in the calculation of the Hamiltonian, the omission of spin and volume makes the overall mathematical setup much easier (see section 2.3 and eq. (3)). The effects of relativity, gravitational waves and other non-gravitational forces are also neglected to make the problem simpler to solve.

2.3 n-body problem described by Hamiltonian mechanics

To find a solution to the n-body problem, the governing system of the bodies must be described. For describing the system, Hamiltonian mechanics are introduced (for equations and derivations, see Goldstein, Poole, Safko, 2002). A Hamiltonian system is a dynamical system (i.e., the state of the system varies with time over a geometrical domain) where the dynamics of the system conserves the total energy, i.e., the Hamiltonian (see eq. (13)). In physics, one can think of a dynamical system as one that describes changes in planetary orbits, particles in an electromagnetic field etc. Together with Hamiltonian mechanics, the Universal Law

of Gravitation will be used to model and find an approximated solution for n-body the problem. General relativity can be applied to further bolster the accuracy of the approximation, though its setup is much more complicated to due additional factors like distortion in the space-time continuum (Chisari, Zaldarriaga, 2011), and therefore not included in this mathematical model.

First, let the Hamiltonian be defined as:

$$\mathcal{H}(\boldsymbol{x},\boldsymbol{p}) = \sum_{i=1}^{n} \dot{\boldsymbol{x}}_i \boldsymbol{p}_i - \mathcal{L}(\boldsymbol{x},\boldsymbol{p},t) \tag{1}$$

with $\mathcal{L}$ being the Lagrangian of the system, defined as $\mathcal{L}(\boldsymbol{x},\boldsymbol{p},t) = T(\boldsymbol{p}(t)) - V(\boldsymbol{x}(t))$, the difference between the total kinetic energy $T(\boldsymbol{p})$, and the potential energy $V(\boldsymbol{x})$, with $\boldsymbol{x}(t)$ the position and $\boldsymbol{p}(t)$ the momentum.
Letting:

$$\boldsymbol{p} = m\boldsymbol{v} \qquad (2) \qquad \text{and} \qquad T(\boldsymbol{p}) = \frac{1}{2} m\boldsymbol{v}^2 \tag{3}$$

with $\boldsymbol{v}(t)$ being the velocity. Due to the simplifications in section 2.2, the kinetic energy can be written as in eq. (3), with the rotational kinetic energy being omitted. The summation term of $\mathcal{H}$ can be written as:

$$\sum_{i=1}^{n} \dot{\boldsymbol{x}}_i \boldsymbol{p}_i = \sum_{i=1}^{n} T(\boldsymbol{p}_i) = \ T(\boldsymbol{p}) \tag{4}$$

the sum of the kinetic energy of all components in the system, i.e., the total kinetic energy. The Hamiltonian $\mathcal{H}$ can then be written as:

$$\mathcal{H}(\boldsymbol{x},\boldsymbol{p},t) = T(\boldsymbol{p}(t)) + \ V(\boldsymbol{x}(t)) \tag{5}$$

the sum of the total kinetic and potential energy of the system.

Two-body problem: To understand the general formulation of the n-body problem, it is easier to start by considering a two-body system, the easiest being the case of one stationary and one free-moving particle. By The Universal Law of Gravitation, the interactive force between two objects in a gravitational field is defined as:

$$\boldsymbol{F}(\boldsymbol{r}) = -\frac{GMm}{|\boldsymbol{r}|^2}\hat{\boldsymbol{r}} \tag{6}$$

G being the gravitational constant, M the mass of the stationary particle, m the mass of the free-moving particle, $|\boldsymbol{r}|$ the Euclidean distance (l_2 $norm$) between the particles, $\hat{\boldsymbol{r}}$ the outward unit vector in radial direction. Together with the definition of potential energy, the gravitational potential energy is given by:

$$V(\boldsymbol{x}) = \int_{\mathcal{C}} \boldsymbol{F}(\boldsymbol{r}) \cdot d\boldsymbol{r} = -\int_{\infty}^{|\boldsymbol{x}|} -\frac{GMm}{|\boldsymbol{r}|^2}\hat{\boldsymbol{r}}\ d\boldsymbol{r} = \frac{GMm}{|\boldsymbol{x}|} \tag{7}$$

with $|\boldsymbol{x}|$ being the Euclidean distance. The Hamiltonian for the two-body problem, together with eq. (2) and eq. (3), can then be written as:

$$\mathcal{H}(\boldsymbol{x}, \boldsymbol{p}) = \frac{\boldsymbol{p}^2}{\boldsymbol{2}m} - \frac{GMm}{|\boldsymbol{x}|} \tag{8}$$

The derivatives are then given as:

$$\frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} = \frac{\boldsymbol{p}}{m} \tag{9}$$

and

$$\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} = \frac{GMm}{|\boldsymbol{x}|^3}\boldsymbol{x} \tag{10}$$

and, assuming that the system is described by the Hamiltonian equations:

$$\frac{d\boldsymbol{x}}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} \tag{11}$$

and

$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} \tag{12}$$

the total energy of the system is conserved, as:

$$\frac{d\mathcal{H}}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{dt} + \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}\frac{d\boldsymbol{p}}{dt} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{x}}\frac{\partial \mathcal{H}}{\partial \boldsymbol{p}} - \frac{\partial \mathcal{H}}{\partial \boldsymbol{p}}\frac{\partial \mathcal{H}}{\partial \boldsymbol{x}} = 0 \tag{13}$$

As the Hamiltonian system constructed is a consequence of the simplification made to the physical system, it should be noted that the conservation of energy is a key property that will play a significant role when choosing a numerical integrator for solving the system and when analyzing results. By using eq. (9)-(12), the dynamic of the planetary two-body system is then given by:

$$\frac{d\boldsymbol{x}}{dt} = \frac{\boldsymbol{p}}{m} \tag{14}$$

and:

$$\frac{d\boldsymbol{p}}{dt} = -\frac{GMm}{|\boldsymbol{x}|^3}\boldsymbol{x} = -\frac{GMm}{|\boldsymbol{x}|^2}\hat{\boldsymbol{x}} \tag{15}$$

n-body problem: If one now considers $n+1$ particles, indexed $0 \to n$, the Hamiltonian in a "stationary" frame of reference will be given by:

$$\mathcal{H}(\boldsymbol{x}_1, \dots, \boldsymbol{x}_n, \boldsymbol{p_1}, \dots, \boldsymbol{p}_n) = \sum_{j=1}^{n} (\frac{\left|\boldsymbol{p}_j\right|^{\boldsymbol{2}}}{2m_j} - G \sum_{l=0}^{j-1} \frac{m_j m_l}{\left|\boldsymbol{x}_j - \boldsymbol{x}_l\right|}), \qquad l \neq j \tag{16}$$

where index j represents the planet whose total energy is being calculated, and index l represents the planet interacting with j. The outer sum collects the energy of every particle, while the inner sum gathers all the gravitational forces acted upon planet j. The $l = 0$ index is used for the Sun, and as the Sun is kept at the origin, $j = 0$ is omitted from the outer sum. The equations governing position and momentum are then:

$$\frac{d\boldsymbol{x}_j}{dt} = \frac{\boldsymbol{p}_j}{m_j} \tag{17}$$

and:

$$\frac{d\boldsymbol{p}_j}{dt} = -G \sum\nolimits_{l=0}^{n} \frac{m_j m_l}{\left|\boldsymbol{x}_j - \boldsymbol{x}_l\right|^3} (\boldsymbol{x}_j - \boldsymbol{x}_l), \qquad l \neq j \tag{18}$$

Closed-form ODE description: Finally, the solution vector of the system can be written on the form $y = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_n, \boldsymbol{p}_1, \dots \boldsymbol{p}_n]$, with motions of the Hamiltonian system described by the ODE-system of general form:

$$\frac{d\boldsymbol{y}}{dt} = F(\boldsymbol{y}), \qquad where\ \boldsymbol{y} = [\boldsymbol{x}, \boldsymbol{p}] \tag{19}$$

or rewritten in the two components:

$$\begin{cases} \dot{\boldsymbol{x}} = \nabla_p \mathcal{H}(\boldsymbol{p}, \boldsymbol{x}) \\ \dot{\boldsymbol{p}} = -\nabla_x \mathcal{H}(\boldsymbol{p}, \boldsymbol{x}) \end{cases} \tag{20}$$

where $\dot{\boldsymbol{x}}$, $\dot{\boldsymbol{p}}$ being the derivatives with respect to time of $\boldsymbol{x}$, $\boldsymbol{p}$, and ∇_p, ∇_x being the gradient with respect to moment and position, respectively. This is the system which will be solved later in chapter 3. As the solar system contains more than two bodies, when trying to describe it using the system above, one cannot find a closed form global solution for the system. Therefore, a numerical integrator will be used in chapter 3 to find a numerical solution.

The reason for working with Hamiltonian mechanics is grounded in two main properties of Hamiltonians systems. Given $\mathcal{H}(\boldsymbol{x}, \boldsymbol{p})$ is a smooth, real-valued function, the Hamiltonian system eq. (20) is (Harrier, 1999):

1) Energy conserving (see eq. (13))
2) Symplectic

The first property is something that comes naturally when wanting to simulate physical problems, as for classical mechanics, we want the total energy of the system conserved. The second property comes from the fact that the Hamiltonian lies on a symplectic manifold in phase-space, with natural splitting in the position and momentum component. A manifold can be loosely described as a topological space that local resembles Euclidean space. The symplectic structure (in $\mathbb{R}^2$ for simplicity) refers to the conservation of the area in phase-space ($dp \wedge dq$) of the flow $\varphi_t(y)$ for all t. The flow $\varphi_t(y)$ of the system is the trajectory of the solution y trough the vector field in phase-space. This symplectic property can then be described in a simplified manner as: the area constructed from the position $\boldsymbol{q}$ and momentum p components of the solution y is constant as the solution advances trough phase-space in time, given an initial condition (q_0, p_0); with phase-space being a space where each point represents a possible state of the system. All ODEs that satisfy the Hamilton's equation of motion have a symplectic flow φ_t, which is a result of the following theorem (Hairer, Wannar, Lubich, 2006):

Theorem 1
Let f: $\mathbb{R}^{2d} \to \mathbb{R}^{2d}$ be a continuously differentiable. Then, $\boldsymbol{y}' = f(\boldsymbol{y})$ is a Hamiltonian system, if and only if its flow $\varphi_t(\boldsymbol{y})$ is symplectic for all $\boldsymbol{y} \in \mathbb{R}^{2d}$ and for all sufficiently small t.

As a direct consequence, symplecticity becomes a characteristic property of the Hamiltonian system eq. (20), which will later be a key-element when choosing a numerical integrator for solving the problem.

Transformed Hamiltonian: The general Hamiltonian, as shown in the equations in this chapter, is dependent on the momentum $\boldsymbol{p}$ and position $\boldsymbol{x}$. Later, in chapter 3, when solving the Hamiltonian system numerically, a transformed system is used where the equations are dependent on the velocity $\boldsymbol{v}$ instead of the momentum $\boldsymbol{p}$. This is merely done for convenience when handling data. From system eq. (20), by eq. (2), (17) and (18), let:

$$
\begin{aligned}
-\nabla_x \mathcal{H}(\boldsymbol{p}_j, \boldsymbol{x}_j) &= \frac{d\boldsymbol{p}_j}{dt} = m_j \frac{d\boldsymbol{v}_j}{dt} \\
= -G \sum_{l=0}^{n} \frac{m_j m_l}{|\boldsymbol{x}_j - \boldsymbol{x}_l|^3} (\boldsymbol{x}_j - \boldsymbol{x}_l) \;\; &= \;\; -\mathcal{F}(\boldsymbol{v}_j, \boldsymbol{x}_j) = -f(\boldsymbol{x}_j)
\end{aligned}
\tag{21}
$$

and

$$
\nabla_p \mathcal{H}(\boldsymbol{p}_j, \boldsymbol{x}_j) = \frac{\boldsymbol{p}_j}{m_j} = \boldsymbol{v}_j = \mathcal{G}(\boldsymbol{v}_j, \boldsymbol{x}_j) = g(\boldsymbol{v}_j) \tag{22}
$$

By this transformation, the assumption is made that the system conserves the Hamiltonian properties **I** and **II** as mentioned above. The transformed closed-form ODE-system then becomes:

$$
\frac{d\tilde{\boldsymbol{y}}}{dt} = F(\tilde{\boldsymbol{y}}), \qquad where\ \tilde{\boldsymbol{y}} = [\boldsymbol{x}, \boldsymbol{v}] \tag{23}
$$

or, as in system eq. (20), rewritten in the two components:

$$
\begin{cases}
\dot{\boldsymbol{x}} = g(\boldsymbol{v}) \\
\dot{\boldsymbol{v}} = -f(\boldsymbol{x})
\end{cases}
\tag{24}
$$

Let it be emphasized that f and g are independent of velocity v and position x, respectively, as this is a key property when relating system eq. (24) to the numerical integrator in section 3.1.

3 Physics-based modeling of the n-body problem

As we try to obtain a deeper understanding of the complexity of the physical world around us, we have often reached out to physics-based models (P.B.M.) to learn relationships between variables and get insight into yet unobserved situations. These models have helped us discover knowledge within many fields of physics, mathematics, and engineering (Daw et al., 2017) and are still the go-to models for many applications (Blakseth, Rasheed, Kvamsdal, San, 2022). Especially for the application in this thesis, the n-body problem, physics-based models are frequently used (Heggie, 2005), where their use has made space-travel possible, amongst many other breakthroughs (Diacu, 1996).

The term physics-based model may not be as familiar as its building blocks; the mathematical equations based on physical foundations for which the system is described, and the numerical solution method used to approximate solutions and simulate the system. This term P.B.M. has recently been more frequently used and is sometimes used in the context of machine learning (M.L.), where one wants to separate P.B.M. from the model derived from M.L. (e.g., Blakseth, Rasheed, Kvamsdal, San, 2022; Daw et al., 2017; Willard et al., 2020).

In the following chapter, a numerical solution method (the symplectic Euler method) for solving the n-body problem will be presented together with some specific properties shared by the Hamiltonian structure of system eq. (20) and the numerical integrator. At the end, numerical results will be presented, where these results will be the baseline for which this thesis seeks improvement.

3.1 Time integrator:

As mentioned earlier, system eq. (20) must be solved numerically to arrive at a useful solution for the $n \geq 3$ - body problem. To arrive at such a solution, Symplectic Euler, also known by names such as Euler-Cromer and Semi-Implicit Euler, is implemented. This numerical method is a one-step, first-order symplectic method based on the Explicit and Implicit Euler method. This method is derived by approximating a discrete solution of $\boldsymbol{p}(t_{i+1})$ and $\boldsymbol{x}(t_{i+1})$ at a given timestep $t_{i+1} = t_i + \Delta t$ with the use of the known solution $\boldsymbol{p}(t_i)$ and $\boldsymbol{x}(t_i)$, where Δt is the chosen timestep size for every update of the system. As one of its names suggests, the method is semi-implicit due to the use of both an implicit step:

$$\boldsymbol{p}_{i+1} = \boldsymbol{p}_i - \Delta t \nabla_x \mathcal{H}(\boldsymbol{p}_{i+1}, \boldsymbol{x}_i) \tag{25}$$

and an explicit step:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta t \nabla_p \mathcal{H}(\boldsymbol{p}_{i+1}, \boldsymbol{x}_i) \tag{26}$$

as the solution method for approximating the system's next state. This yields, by use of the transformed Hamiltonian system eq. (24) above:

$$\boldsymbol{v}_{i+1} m = \boldsymbol{v}_i m - \Delta t f(\boldsymbol{x}_i) \tag{27}$$

$$\boldsymbol{v}_{i+1,j} = \boldsymbol{v}_{i,j} - \Delta t G \sum_{l=0}^{n} \frac{m_l}{\left|\boldsymbol{x}_{i,j} - \boldsymbol{x}_{i,l}\right|^3} (\boldsymbol{x}_{i,j} - \boldsymbol{x}_{i,l}) \tag{28}$$

an explicit one-step discretization of $\boldsymbol{v}$, only dependent on $\boldsymbol{x}$, where idices j and l represents the planets as used in eq. (21) and eq. (22).
And:

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta t g(\boldsymbol{v}_{i+1}) \tag{29}$$

$$\boldsymbol{x}_{i+1,j} = \boldsymbol{x}_{i,j} + \Delta t \boldsymbol{v}_{i+1,j} \tag{30}$$

also, an explicit one-step discretization, this time for $\boldsymbol{x}$ though dependent on both $\boldsymbol{x}$ and $\boldsymbol{v}$.

3.1.1. Properties of Symplectic Euler:

There are many choices for a time integrator. Some standard choices are methods such as the Explicit or Implicit Euler, but for solving a Hamiltonian system, Symplectic Euler is a superior choice. One of the main reasons being connected to the energy preserving properties of the Hamiltonian. Though similar to the Explicit Euler, by adding an implicit term to the method one arrives at a method with almost energy preserving properties, unlike Explicit Euler. The reason is that the Symplectic Euler is, as the name suggests, a symplectic integrator by **theorem 2** (DeVogelaere, 1956), which implies that the method has a discrete flow $\hat{\varphi}_t$ with symplectic structure, thus having the properties of symplecticity as for the Hamiltonian system eq. (20).

Theorem 2
The so − called Symplectic Euler methods

$$\text{I)} \begin{cases} \boldsymbol{p}_{i+1} = \boldsymbol{p}_i - \Delta t \nabla_x \mathcal{H}(\boldsymbol{p}_{i+1}, \boldsymbol{x}_i) \\ \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta t \nabla_p \mathcal{H}(\boldsymbol{p}_{i+1}, \boldsymbol{x}_i) \end{cases} \quad or \quad \text{II)} \begin{cases} \boldsymbol{p}_{i+1} = \boldsymbol{p}_i - \Delta t \nabla_x \mathcal{H}(\boldsymbol{p}_i, \boldsymbol{x}_{i+1}) \\ \boldsymbol{x}_{i+1} = \boldsymbol{x}_i + \Delta t \nabla_p \mathcal{H}(\boldsymbol{p}_i, \boldsymbol{x}_{i+1}) \end{cases}$$

are symplectic $B-$*series methods of order* 1.

The reason for not conserving the energy exact is that one cannot have symplecticity and exact energy conservation at the same time, as shown by Hairer and summarized in the theorem below (see Hairer, 2006 for more details):

Theorem 3
Consider

- *a Hamiltonian system with analytic* $\mathcal{H}: U \to \boldsymbol{R}$, *and*
- *the symplectic Euler method by theorem* 2

As long as $\{\boldsymbol{y}_n = (\boldsymbol{x}_n, \boldsymbol{p}_n)\}$ *stays in a compact set, we have for* $t_n = nh$ *and* $h \to 0$,

$$\mathcal{H}(\boldsymbol{y}_n) = \mathcal{H}(\boldsymbol{y}_0) + \ \mathcal{O}(h^1) + \ \mathcal{O}(t_n e^{-\gamma/\omega h}), \tag{31}$$

where $\gamma > 0$ *only depends on the method, and* ω *is related to the Lipschitz* $-$ *constant (or highest frequency) of the differential equation.*

As pointed out by Hairer; as h becomes sufficiently small, the second error term in eq. (31) becomes exponentially small on exponentially large timescales, and we are left with a method that conserves the Hamiltonian up to a bounded error $\mathcal{O}(h^r)$ on such a timescale. The approximated solution on the symplectic manifold by the integrator may be perturbed from the real solution, but the perturbation is of a lower order than non-symplectic integrators, and it turns out to be a linear drift in the perturbation (Benettin, Giorgilli, 1994), which gives good approximation for long-term solutions.

As this method is still first order, its global truncation error is directly proportional to the step-size Δt (see figure 6 section 3.3); but as this method is a symplectic integrator, there is an almost guarantee for periodicity and the error is given an artificial bounded oscillation, unlike Explicit Euler, where the error increases with time (Hairer, 2006).

Another benefit to the Semi-Implicit Euler is its simplicity. Though formulated as containing an implicit step, the way it is implemented, by first calculation eq.

(28) then eq. (30), lead to a purely explicit scheme. This is due to the variable independence of system eq. (24), though the method will still preserve the properties of the general Symplectic Euler by **theorem 2.** As the outline of the method shows resemblance to a two-step Explicit Euler, its implementation is simple, and it has relatively low computational cost compared to other symplectic integrator and the Implicit Euler.

3.2 Data by NASA for initial state and comparable solution of the planetary system

As mentioned at the start of this chapter, the goal for this P.B.M. is to simulate the Solar System by solving eq. (24) with use of the numerical integrator described by eq. (28) and (30). For solving the problem there is the need for initial data. As the n-body problem for the solar system does not have a useful analytical solution, the initial data, and comparable exact solutions for error-estimate at later timesteps, cannot be calculated. This gives the need to obtain comparable data by observation. In this thesis, data for the state vectors $\boldsymbol{s} = (\boldsymbol{x},\ \boldsymbol{v}) =\ (x,\ y,\ z,\ v_x,\ v_y,\ v_z)$ and different parameters of each planet in the Solar System are obtained through NASAs JPL Horizons data system. As this data system does not state errors in their values (except for a few parameter), and one can assume that NASA has managed to obtain data more precise than what this P.B.M. can achieve, the data from NASA are treated as the exact solution to the system eq. (24) when considering initial condition and error-estimates. The data from NASA is obtained using a discretization with step-size $\Delta t = 10\text{min}$. As eq. (28) shows, there is also the need of values for the parameters m (i.e., mass of the celestial bodies). These are retrieved directly from the JPL Horizons website[1]. State vectors are retrieved through jplhorizons, a module of the Python package Astroquery, which queries information from the JPL Horizons data system (see Appendix A.1) The state vectors queried through jplhorizons are given with a position and velocity component. Due to this fact, as mentioned in section 2.3, the system eq. (24) to be solved is given by $(\boldsymbol{x}, \boldsymbol{v})$, and not $(\boldsymbol{x}, \boldsymbol{p})$, for easier handling of data.

Units: The parameters m are stated in SI-units (International System of Units) due to the magnitude, though the state vectors are stated in AB-units (Astronomical System of Units) due to astronomical quantities tending to have large magnitudes impractical to express in SI-units. This leads to the necessity of rewriting constants and parameters in units which gives values within the same scale. To obtain this, the following conversions have been made (see table 1):

[1] https://ssd.jpl.nasa.gov/horizons/app.html#/

Table 1: SI-units to AB-units conversions.

Symbol in Equations	Quantity/ Constant	Value	Astronomical Base Unit (AU)	Value	SI Base Unit
x	Distance	1	au	$1.495979707 \cdot 10^{11}$	m
v	Velocity	1	$au \cdot day^{-1}$	$1.73146 \cdot 10^{1}$	$m \cdot s^{-1}$
m	Mass	1	$M_{\odot}$	$1.98850 \cdot 10^{30}$	kg
t	Time	1	day	$8.6400 \cdot 10^{4}$	s
G	U.G. constant[2]	$6.67430 \cdot 10^{-11}$	$au^3 M_{\odot}^{-1} day^{-2}$	$2.95926 \cdot 10^{-11}$	$m^3 kg^{-1} s^{-2}$

3.3 Numerical results:

The first numerical results in this section are obtained by applying the Symplectic Euler with initial conditions from NASA (see table 17, Appendix A.1) over a period of approximately $T \approx 225$ years, with a fixed timestep of **1** day (i.e., $\Delta t = 1$ as time unit is in astronomical units; see table 1). The period T is an arbitrary choice for the P.B.M., but not so much for the final data-driven model of section 4.3. T is therefore kept constant over the two models for convenience when comparing results. The timestep is fixed to $\Delta t = 1$ to make a coarse discretization and lower the accuracy of the P.M.B. This timestep is clearly large, as the innermost planet has an orbital period of approximately 88 days. The large timestep comes though with some benefits: the low computational cost. This low computational cost is a key aspect of the hybrid model in chapter 5.

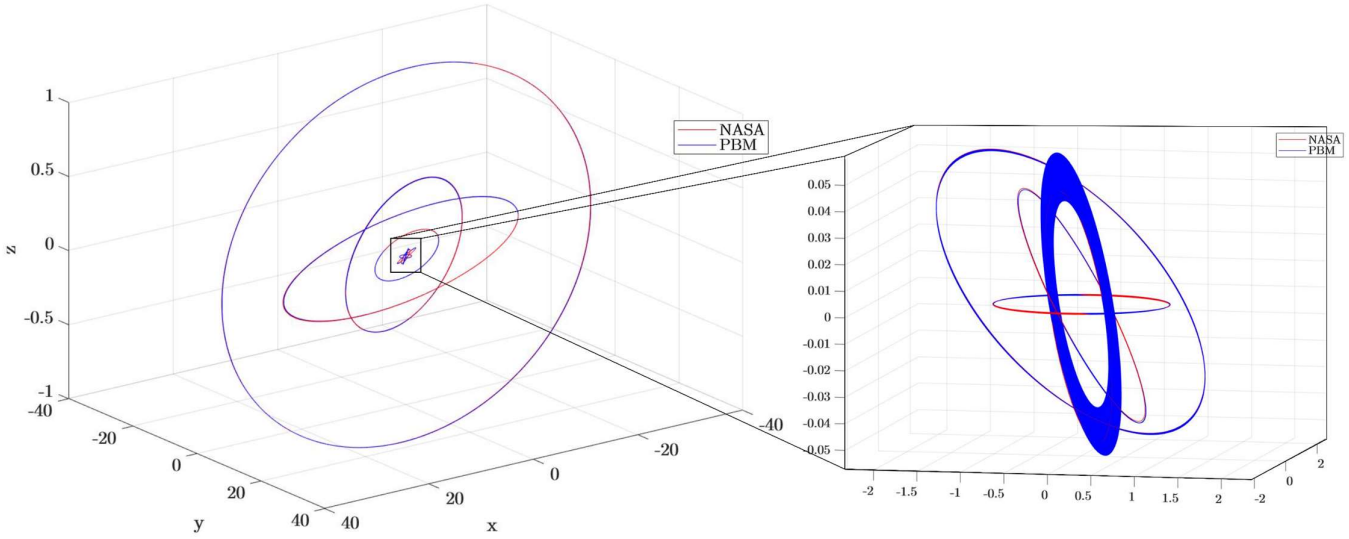

Figure 3: 3d plot of PBM results vs. NASA data over 225 years.

[2] Universal Gravitational constant

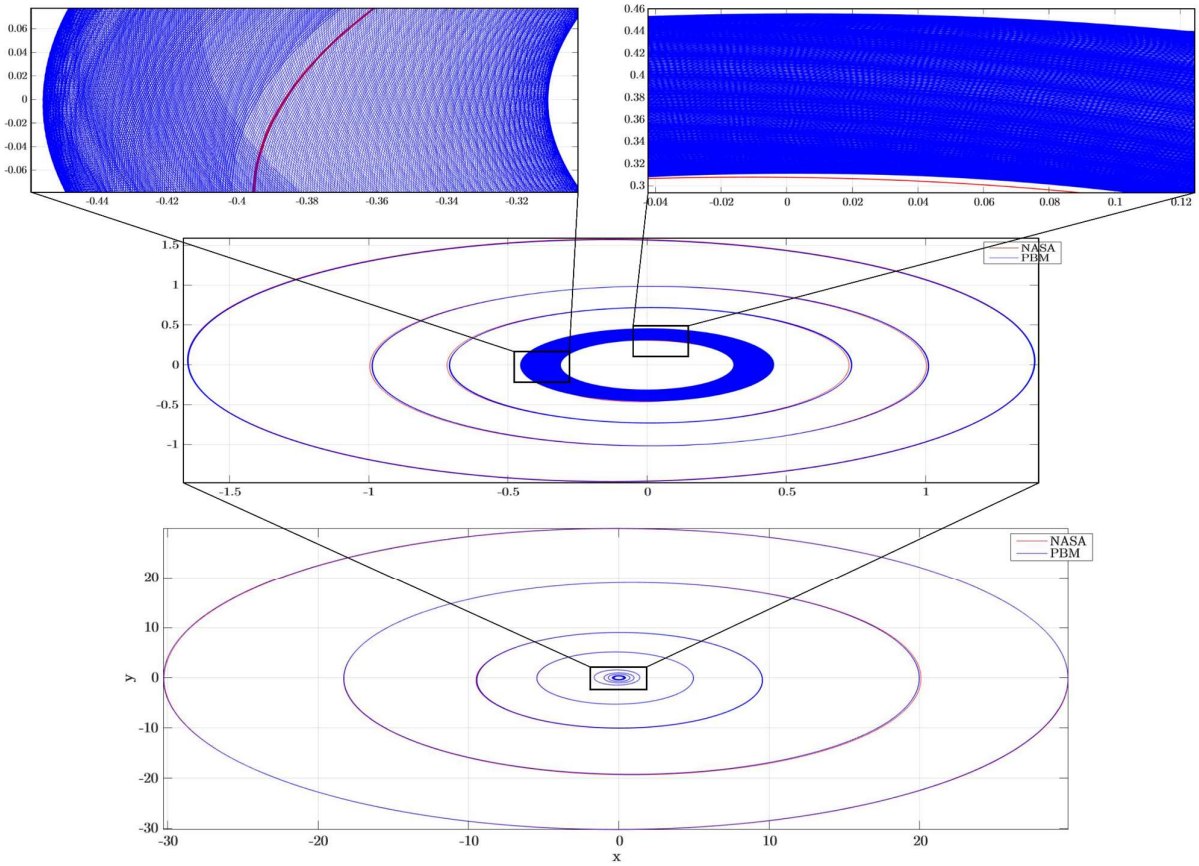

Figure 4: 2d plot of PBM results vs. NASA data over 225 years.

Figure 3 and 4 show a 3d and 2d plot, respectively, of the results from the P.B.M. against the data from NASA. The plots show a clear overlap between the numerical and exact solution for all the planets, except for the innermost planet. For the innermost planet one can see that at the P.B.M. increases the semi-minor axis in comparison to the exact solution, and that there is a large fluctuation in the semi-major axis of the innermost planets orbit. The periodic instability of the orbit is most likely due to the large timestep used in comparison to the orbital period of the planet. The actual motion of the orbit of the innermost planet may also be affected by gravitational forces not correctly described by the Hamiltonian system eq. 20 and may need a more relativistic approach.

Table 2 shows the average absolute and relative error of both position and velocity for each planet. The Sun is not included in the table as it stays fixed at the origin, and thus the errors are zero. The absolute errors are calculated using the Frobenius norm eq. (32), a matrix norm defined in the same manner as the standard L^2-vector norm eq. (36), for the Euclidean distance between the P.B.M. solution matrix $\hat{A}$ and the solution matrix A. The relative error is calculated in the

same manner, just normalized using the Frobenius norm of the solution matrix A, eq. (33).

$$\left\|\hat{A} - A\right\|_F = \sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n}|\hat{a}_{ij} - a_{ij}|^2} \tag{32}$$

$$\frac{\left\|\hat{A} - A\right\|_F}{\|A\|_F} = \frac{\sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n}|\hat{a}_{ij} - a_{ij}|^2}}{\sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n}|a_{ij}|^2}} \tag{33}$$

The errors are then scaled by the square root of number of steps, $\sqrt{m}$, and averaged of the time-interval $\boldsymbol{T}$, such that the errors becomes comparable for different methods.

$$\frac{\left\|\hat{A} - A\right\|_F}{\sqrt{m}\cdot T} = \frac{\sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n}|\hat{a}_{ij} - a_{ij}|^2}}{\sqrt{m}\cdot T} \tag{34}$$

$$\frac{\left\|\hat{A} - A\right\|_F}{\|A\|_F}\frac{1}{\sqrt{m}\cdot T} = \frac{\sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n}|\hat{a}_{ij} - a_{ij}|^2}}{\sqrt{\sum_{i=0}^{m}\sum_{j=0}^{n}|a_{ij}|^2\cdot m}\cdot T} \tag{35}$$

Table 2: Average absolute and relative error by PBM for position and velocity of each planet.

Celestial object	Avg. absolute error (position)	Avg. absolute error (velocity)	Avg. relative error (position)	Avg. relative error (velocity)
Mercury	$6.8067 \cdot 10^{-6}$	$4.7571 \cdot 10^{-7}$	$5.9479 \cdot 10^{-8}$	$6.0021 \cdot 10^{-8}$
Venus	$6.9013 \cdot 10^{-6}$	$1.9551 \cdot 10^{-7}$	$3.3281 \cdot 10^{-8}$	$3.3719 \cdot 10^{-8}$
Earth	$5.1918 \cdot 10^{-6}$	$8.7784 \cdot 10^{-8}$	$1.8107 \cdot 10^{-8}$	$1.7801 \cdot 10^{-8}$
Mars	$7.1478 \cdot 10^{-6}$	$6.5433 \cdot 10^{-8}$	$1.6256 \cdot 10^{-8}$	$1.6382 \cdot 10^{-8}$
Jupiter	$1.0662 \cdot 10^{-5}$	$1.5531 \cdot 10^{-8}$	$7.1360 \cdot 10^{-9}$	$7.1807 \cdot 10^{-9}$
Saturn	$1.1771 \cdot 10^{-5}$	$6.8659 \cdot 10^{-9}$	$4.2948 \cdot 10^{-9}$	$4.2964 \cdot 10^{-9}$
Uranus	$1.1812 \cdot 10^{-5}$	$2.3975 \cdot 10^{-9}$	$2.1405 \cdot 10^{-9}$	$2.1311 \cdot 10^{-9}$
Neptune	$1.4212 \cdot 10^{-6}$	$1.4697 \cdot 10^{-10}$	$1.6469 \cdot 10^{-10}$	$1.6348 \cdot 10^{-10}$

By table 2, one can see that the average absolute error for the planets position is in general increasing for larger orbits, as expected when calculating absolute error for increasing values. For average relative error in positions, the results show a

decrease in error for larger orbits. This is most likely due to the large timesteps in comparison to the orbital periods, as larger timesteps will cover a greater arc length for short orbital periods. The exception is the outermost planet, where the absolute value is the lowest of all the planets, though its relative error follows the previous statement. This is most likely due to the overall error for the outermost being very low. For the average absolute error in velocities, the errors are decreasing with increase in orbit size. This fits well with the system simulated, as for the planets in the Solar System, larger orbits yields lower orbital velocities. The average relative error in velocities also supports this.

The next result is a plot which shows the Hamiltonian calculated with respect to the numerical solutions by the P.B.M. against the Hamiltonian for the NASA data. The Hamiltonian by the P.B.M. appears to be bounded, with an oscillation in the 7^{th} significant figure. There also appears to be a small perturbation in the P.B.M Hamiltonian as it gets delayed over time compared to the Hamiltonian for the NASA data. This result fits well with **theorem 3** and the subsequent statements in section 3.1.1, as the theory states that one should expect an almost conserved Hamiltonian and a small perturbation from the real solution.

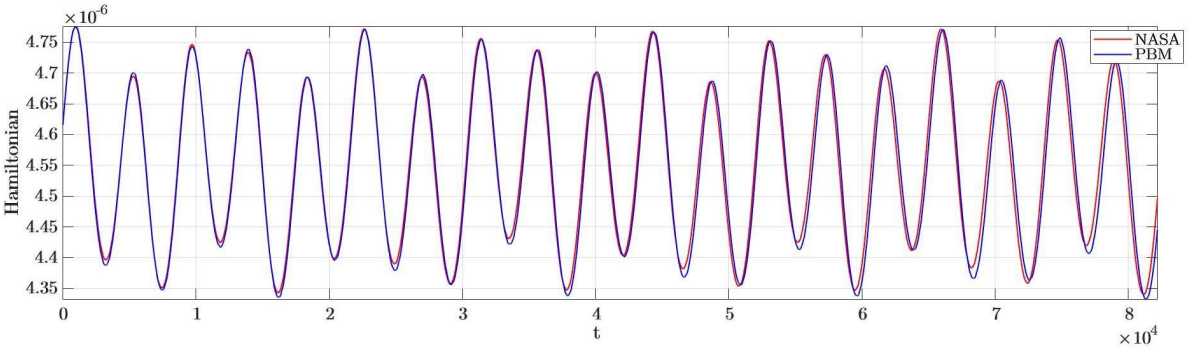

Figure 5: plot of Hamiltonian of PBM results vs. NASA data over 225 years.

Table 3: Average absolute and relative error by PBM for the Hamiltonian.

	Avg. absolute error	Avg. relative error
Hamiltonian	$2.4024 \cdot 10^{-13}$	$1.8411 \cdot 10^{-10}$

Table 3 shows the average absolute and relative errors for the P.B.M. Hamiltonian plotted in figure 5. The absolute error is calcutated using standard L^2-vector norm eq. (36) for the Euclidean distance between the P.B.M. Hamiltonian vector $\hat{\boldsymbol{y}}$ and the solution Hamiltonian vector $\boldsymbol{y}$. The relative error is calculated in

the same manner, just normalized using the L^2-vector norm of the solution vector $\boldsymbol{y}$, eq. (37).

$$\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2 = \sqrt[2]{\sum_{i=0}^{n} |\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i|^2} \tag{36}$$

$$\frac{\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2}{\|\boldsymbol{y}\|_2} = \frac{\sqrt[2]{\sum_{i=0}^{n} |\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i|^2}}{\sqrt[2]{\sum_{i=0}^{n} |\boldsymbol{y}_i|^2}} \tag{37}$$

The errors are then scaled by the square root of number of steps, $\sqrt{n}$, such that the errors becomes comparable for methods with different timesteps. The errors are also scaled by the length of the time interval, T, which yields an average error per timestep. This way, results by methods on different time intervals can be compared.

$$\frac{\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2}{\sqrt{n} \cdot T} = \frac{\sqrt[2]{\sum_{i=0}^{n} |\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i|^2}}{\sqrt{n} \cdot T} \tag{38}$$

$$\frac{\|\hat{\boldsymbol{y}} - \boldsymbol{y}\|_2}{\|\boldsymbol{y}\|_2} \frac{1}{\sqrt{n} \cdot T} = \frac{\sqrt[2]{\sum_{i=0}^{n} |\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i|^2}}{\sqrt[2]{\sum_{i=0}^{n} |\boldsymbol{y}_i|^2 \cdot n} \cdot T} \tag{39}$$

The last numerical result is a plot which shows the L^2-error of the P.B.M against NASA data by eq. (36). The plot shows the local truncation error against a quadratic decrease in timesteps, and as the plot shows, the local truncation error of method is of order $\mathcal{O}(h^2)$. Its global truncation error thus $\mathcal{O}(h^1)$, which agrees with the **theorem 2**. The plotted values can also be found in table 4.

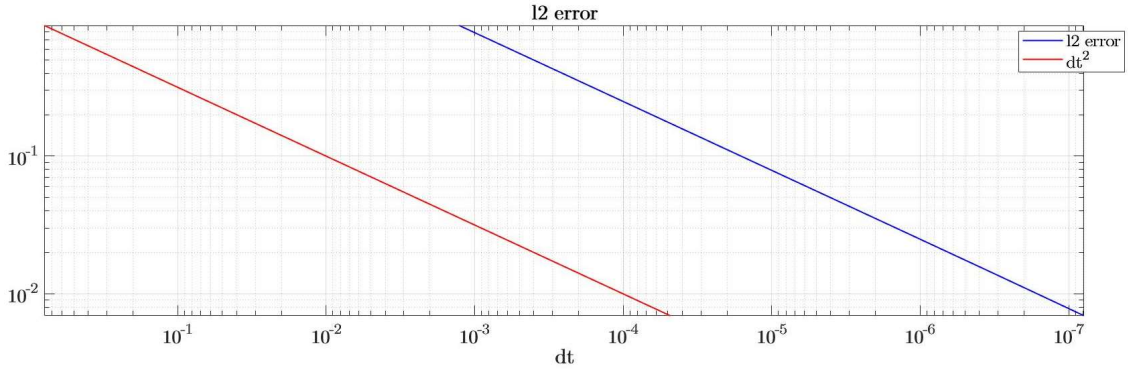

Figure 6: plot of the local L^2-error of the PBM.

Table 4: Local truncation error by PBM for different timesteps.

Δt	$\|\hat{y} - y\|$	$\frac{\|\hat{y} - y\|(\frac{\Delta t}{2})}{\|\hat{y} - y\|(\Delta t)}$
$^8/_9$	$1.2722 \cdot 10^{-3}$	–
$^4/_9$	$3.1759 \cdot 10^{-4}$	$2.4964 \cdot 10^{-1}$
$^2/_9$	$7.9364 \cdot 10^{-5}$	$2.4989 \cdot 10^{-1}$
$^1/_9$	$1.9839 \cdot 10^{-5}$	$2.4997 \cdot 10^{-1}$
$^1/_{18}$	$4.9606 \cdot 10^{-6}$	$2.5004 \cdot 10^{-1}$
$^1/_{36}$	$1.2412 \cdot 10^{-6}$	$2.5021 \cdot 10^{-1}$
$^1/_{72}$	$3.1140 \cdot 10^{-7}$	$2.5089 \cdot 10^{-1}$
$^1/_{144}$	$7.8937 \cdot 10^{-8}$	$2.5349 \cdot 10^{-1}$

4 Data-driven modeling of the n-body problem

The world of Artificial Intelligence (A.I.) has come forth into the light of the general population over the last decades and have built huge expectations for what humans can achieve the decades to come. Some may say there is an overhype in the general population for the subject, though it is still a wide research field with many hidden possibilities (Ford, 2018).

In the following chapter, a general introduction into the world of Artificial Intelligence, Machine Learning and Deep Learning will be given. Following, the basics of Neural Networks will be presented and accompanied by an example (section 4.2), to give the (particularly unexperienced) reader a general insight into Neural Networks – the conceptual setup procedure will remain the same for all Neural Network based models in this thesis; additionally, this simple example allows for a clear discussion of network optimization techniques. Later, in section 4.3, a pure data-driven model (D.D.M.), based on Neural Networks, will be presented as an alternative to the physics-based model (P.B.M.) in chapter 3 for modeling the n-body problem for planetary motion. The motivation for creating a pure D.D.M. for solving the n-body problem is that, after the model has been correctly set up and trained, the computational cost is very low in comparison to many highly accurate numerical integrator (Breen, Foley, Boekholt, Zwart, 2020). This data-driven model will be one of the main components of the final hybrid-model presented in chapter 5 and will be responsible for an additive residual term which is used to improve the results for the physics-based model.

4.1 Artificial Intelligence vs. Machine Learning vs. Deep Learning

Some concepts which are often intertwined are the concepts of Artificial Intelligence, Machine Learning and Deep Learning. In the following section, a brief introduction to these concepts will be given, and how they are connected, so it is clear which concepts are used in this thesis.

Through the ages, humans have searched for ways to efficiently execute tasks and improve performance. One of the ways we have managed to do this is through the means of Artificial Intelligence (A.I.). A.I. is not just another tool to improve our daily lives, it also helps us to further understand human beings and answer intricated questions about the world around us. The use of A.I. is nothing new, it dates back to the 1940s, and with its first conceptual mentioning as early as the 1840s (Boden, 2016). A.I. enables us to perform tasks which normally would be

physically and/or temporally impossible by the human hand and mind. These tasks can come in the form of overwhelmingly long mathematical calculations, which by the human mind would take over a lifetime to complete, or in the form observational oriented tasks, where humans' physical attributes makes certain observations impossible.

To get a much clearer understanding of A.I., we need to look at its focal point: intelligence. From the Oxford Languages dictionary[3] intelligence is defined as "the ability to acquire and apply knowledge and skills". From this general definition one can see there are many opportunities for debate. Knowledge can be split up in different categories, same goes for skills, and there are many ways to apply skills and knowledge. This leads to different definitions of intelligence; some splitting the term into different subtypes, some based in biology, some based in psychology (for more in-dept information, see Boden 2016).

Artificial intelligence is mostly known as the concept of replicating human behavior to be able to perform tasks which normally would require human intelligence, though A.I. is not confined to these biological mechanics found in humans. There is the possibility to create A.I. which completes these tasks based on methods very unlike those of the human mind. So, what is the difference, or rather the similarity, between Machine Learning (M.L.) and A.I.? A.I. is just an umbrella term that includes Machine Learning, amongst other things. A.I. can be as simple as a digital calculator, where one could execute basic arithmetic. This would be an explicit A.I., where we define every possible scenario and the desired outcome. Here, the algorithm, or the "skill" of the calculator is defined by us. For M.L., this would not be the case. To create a program which is based on M.L., we would design a program where the computer itself creates the "skills". We can think of this as a Blackbox; we give the computer some input, and it produces some output. We do not know what happens inside the box, but by giving the program some correct solutions to compare its outputs to, the program itself can adjust parameters inside this Blackbox, so that in the end the output comes relatively close to the correct solutions. By doing this over and over for many different input-output pairs, which can be referred to as the programs "learning process", the program "learns" how to solve the given problem for different inputs.

The kind of problems which can be solved by M.L. are generally divided into two categories: classification- and regression-problems. For classification-problems, a given input is sorted and produces a specific output within a limited amount of

[3] https://languages.oup.com/

output labels, whereas for regression-problems, the output is a continuous number. Even though the computer itself is in charge of the "learning process", we can define features of the inputs for which the program should priorities when learning. A classic example of this comes in the form of a classification problem:

If you are designing a program to differentiate between cats and dogs, the program would look at all the features of dogs and all the features of cats and then try to compare these features to a given input, let's say a picture of a cat or a dog. One could argue that cats and dogs have an infinite number of features, so this task would prove impossible for the program. What we can do then is give the program some features of the cat and dog to priorities, e.g., the shape of the face, tail etc., such that the program in many ways only looks at those features when classifying the picture. As there would still be some similarities between features of cats and dogs of specific breeds, the program is not guaranteed to successfully classify the picture every time, but it could still come close.

Another way to approach M.L. is to exclude these features and let the program itself figure out which features to be prioritized. This is the concept of Deep Learning (D.L.) and is most known for its Artificial Neural Network (A.N.N.) approach, or just Neural Network (N.N.) for short. Its structure stems from the idea of replicating the human brain, with its billions of neurons layered up in a specific way.

4.2 Neural Network implementation with example

To further understand Deep Learning and especially how its implemented in this thesis, the following section will present a fairly "easy to implement" D.L. program. This D.L. program is a data driven model, i.e., the model is derived solely by considering empirical data, not by any knowledge of the physical process by which the output is obtained (Souza, Araújo, Mendes, J., 2016). The D.D.M. uses a fixed pipeline, which is the same for every D.D.M. in this thesis; though, some parts within the pipeline have small alterations from model to model. The foundational pipeline for the D.D.M. (see figure 7) is based on three components; the N.N. with its architecture, a loss function to be minimized, and an optimization step to optimize parameters. The goal is then to train the N.N. by going through the pipeline, such that in the end its output (the predictor) is close to the solution.

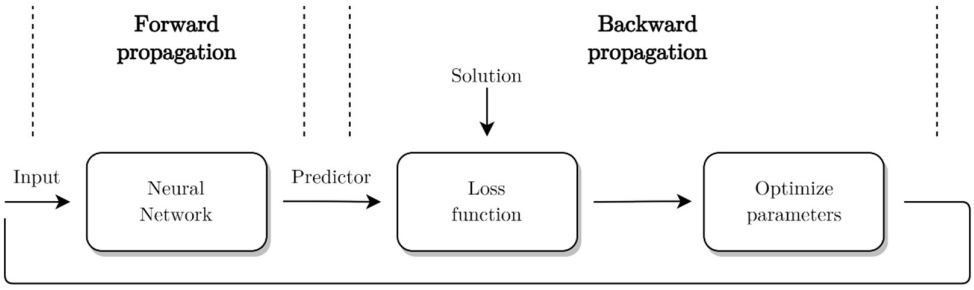

Figure 7: Basic N.N. pipeline.

Guiding example: Mapping polar coordinates to Cartesian coordinates: Throughout this section, an example will be used to associate the D.D.M. with a simple problem and see how different elements affect the training and prediction quality of the D.D.M. The D.D.M. will try to learn eq. (40), where it will try to map polar coordinates to Cartesian coordinates.

$$\begin{cases} x = r\ cos\ (\theta) \\ y = r\ sin\ (\theta) \end{cases} \tag{40}$$

Data: After the model's purpose has been established conceptually, e.g., learn eq. (40), data associated with the task can be constructed. The data's objective is to present the N.N. with enough general examples for it to be able to learn relationships between input and output data. This is one of the first modeling choices one has to consider. The finished D.D.M. is in most part defined by the data, and one has to carefully choose the N.N. input and output data in correlation to the overall idea of what the D.D.M. should resemble. This is why a D.D.M. can predict results not bound by mathematical and physical laws, as it only takes into account the data it is presented with. It should be stressed that the quality of the data can be paramount in the predictive capability of the model, as will be made clear in the results to come. In this example, to train the N.N., an input dataset of 100 000 points $\boldsymbol{x}^{input} = (\theta, r)$ with $\theta \in [0, 2\pi]$ and $r \in [0, 100]$ will be used, together with an output-dataset with corresponding points $\boldsymbol{y}^{sol} = (x, y)$ by eq. (40). This matches each end of the mathematical algorithm the D.D.M. should reproduce, and it is up to the N.N. to find relationships between these datapoints.

Forward Propagation: First is to feed forward the input $\boldsymbol{x}^{input}$ through the N.N. $\mathcal{N}$, where the N.N. preforms a computational algorithm defined by the modeler, and then outputs a predictor, $\boldsymbol{y}^{pred}$, that is:

$$\mathcal{N}(\boldsymbol{x}^{input}) = \boldsymbol{y}^{pred} \tag{41}$$

This process is commonly referred to as Forward Propagation. The predictor, $\boldsymbol{y}^{pred}$, is whatever the N.N. produces by sending the input through the network. The goal is to get $\boldsymbol{y}^{pred} = \boldsymbol{y}^{sol}$, which is done by training the N.N. (see the following paragraphs). For this example, the input for the N.N. is the point in polar coordinates, $\boldsymbol{x}^{input} = (\theta, r)$, and the output predictor is the corresponding Cartesian coordinate $\boldsymbol{y}^{pred} = (x, y)^{pred}$. This yields:

$$\mathcal{N}(\theta, r) = (x, y)^{pred} \tag{42}$$

Backward Propagation: The next step is to compare the predictor $\boldsymbol{y}^{pred}$ to the output $\boldsymbol{y}^{sol}$; this is normally done through a loss function – this can be a crucial modeling step as there exist many different possibilities for different problems. The one used in this thesis, eq. (43), is the Mean Square Error (MSE), as its one of the most commonly used loss functions. The next step is an optimization step with the goal to minimize the loss function, i.e., the distance between predictions and output data:

$$f_{loss} = \frac{1}{N}\sum_{j=1}^{N} \|\boldsymbol{y}_j^{pred} - \boldsymbol{y}_j^{sol}\|^2 \rightarrow min \tag{43}$$

Here an optimization algorithm is used to calculate the gradients for the loss function, $\nabla f_{loss}(\boldsymbol{y}_k)$, and to minimize the loss function with the use of these gradients, where $\boldsymbol{k}$ is the iteration index of the optimization routine. The algorithm minimizes the loss function by eq. (44) (i.e., some form of gradient decent), where the basic idea is to follow the gradient backwards to minimize the objective function (i.e., the loss function) and updates the parameters (w, b) (i.e., weights and biases, see neuron-to-neuron connection later in this section) in the N.N. respectively, where α_n is the step-size.

$$\boldsymbol{y}_{k+1} \approx \boldsymbol{y}_k - \alpha_k \nabla f_{loss}(\boldsymbol{y}_k) \tag{44}$$

This step, together with the loss function step, is what is commonly referred to as Backward Propagation, which is due to the fact that the steps revolve around sending and tracking information backward through the N.N. This Backward Propagation is what makes the N.N. able to adjust to data and learn. It is important to note that the N.N. can adjust too much to the data and end up overfitting, that is, the N.N. does not generalize well and mostly only works for the specific case it is trained for. This can happen when the N.N. is too large in comparison to the data (there is in general no direct way of telling how big the network should be for a given problem, it is a process of trial and error), or by doing to many Backward

Propagations. A solution to overfitting is to implement different regularization methods, which will be introduced later in this section.

As networks can be quite large and complicated, the optimization algorithm can be quite complicated as well. To stay concise and not go into too many details, the optimization algorithm used in the D.D.Ms. in this thesis, as in the case for the guiding example, is call ADAM. This algorithm is based on Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp) to minimize the loss function, which is one of the most commonly used optimization algorithms (for more in-dept details, see Kingma, Ba, 2014).

Train/Validation/Test - sequence: To track the learning process of the N.N., one can repeat the same process as in figure (7) but removing the optimization step, as shown in figure (8). This implementation is referred to as the validation sequence, and the former, the training sequence. This way, one gives the N.N. an input and checks the loss function to see if the difference between the predictor and the solution is small. This process would represent a genuine application of the N.N. and lets us know in some sense how the N.N. would perform if one would stop the learning process at that stage. When the training is complete, the test sequence is initiated. This is where one uses the N.N. in a real application with the use of inputs which have not been used for training and validations (see figure 8). If the N.N. performs as wanted at this sequence, the learning process is generally complete.

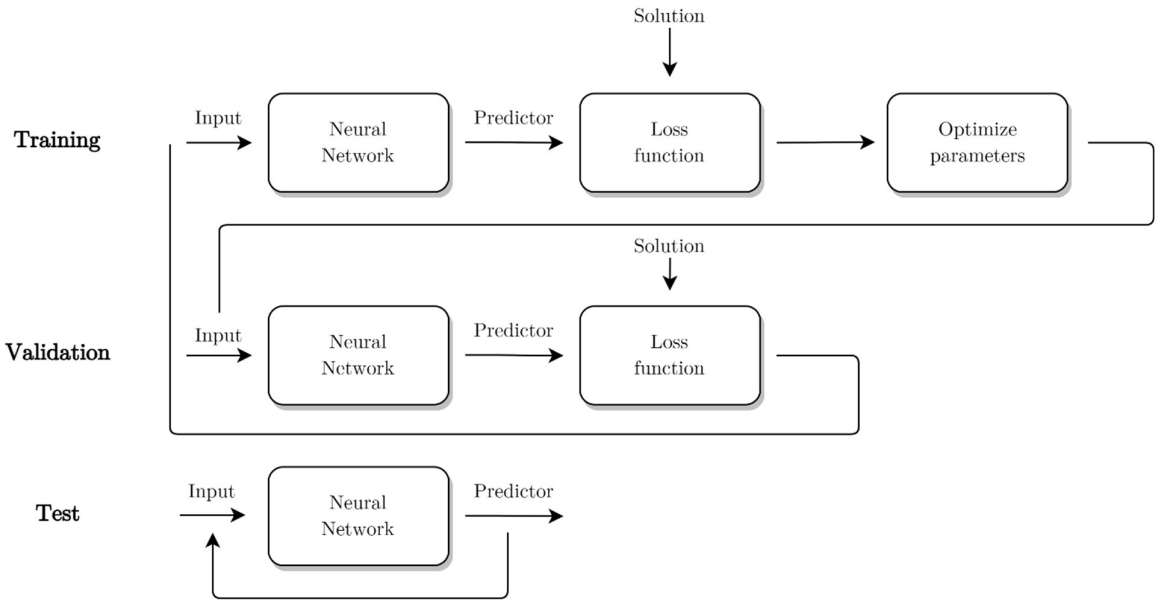

Figure 8: Full D.D.M. pipeline.

N.N. Architecture: The N.N. step in the pipeline is as mentioned the part in which an input is passed through a machine code to produce a predictor. The way the N.N. arrives at the predictor is determined by how the N.N. architecture is defined. Defining the architecture is part of the modeling process; in general, the architecture is up to the modeler and can be constructed in countless ways, but there are some standardized approaches.

For this example, a type of "Feedforward Deep Neural Network" (F.D.N.N) architecture is used. "Feedforward" meaning that the information travels along one direction; an input is given, and the N.N. passes the input through a series of various transformations and gives an output, the predictor. "Deep" refers the N.N. having multiple "hidden" layers of neurons it passes the input trough. These types of N.N. architectures are the quintessential deep learning models (Goodfellow, Bengio, Courville, 2016), though there are other types of architectures which can be used (e.g., "Recurrent" N.N., which are similar to "Feedforward" networks, but includes feedback connections).

F.D.N.N. architectures consist in general on three types of layers: one input layer, one output layer, and "hidden" layers. The input layer is provided the input and the output layer returns the predictor. Each layer consists of neurons, each connected in-between layers. The "hidden" layers are layers where there is no clear interpretation of the role of each single neuron. This is where the N.N. updates its parameters during the Backward Propagation, where the modeler has no direct influence over what the parameter are set to and why. This can be considered the main "Blackbox" of the N.N. The number of "hidden" layers are chosen by the modeler, as well as the number of neurons in each "hidden" layer. For this example, the network consists of three "hidden" layers, each containing 50 neurons, as shown in figure 9. The input and output layers have both two neurons, one neuron for each element in (θ, r) in the input layer and for (x, y) in the output layer.

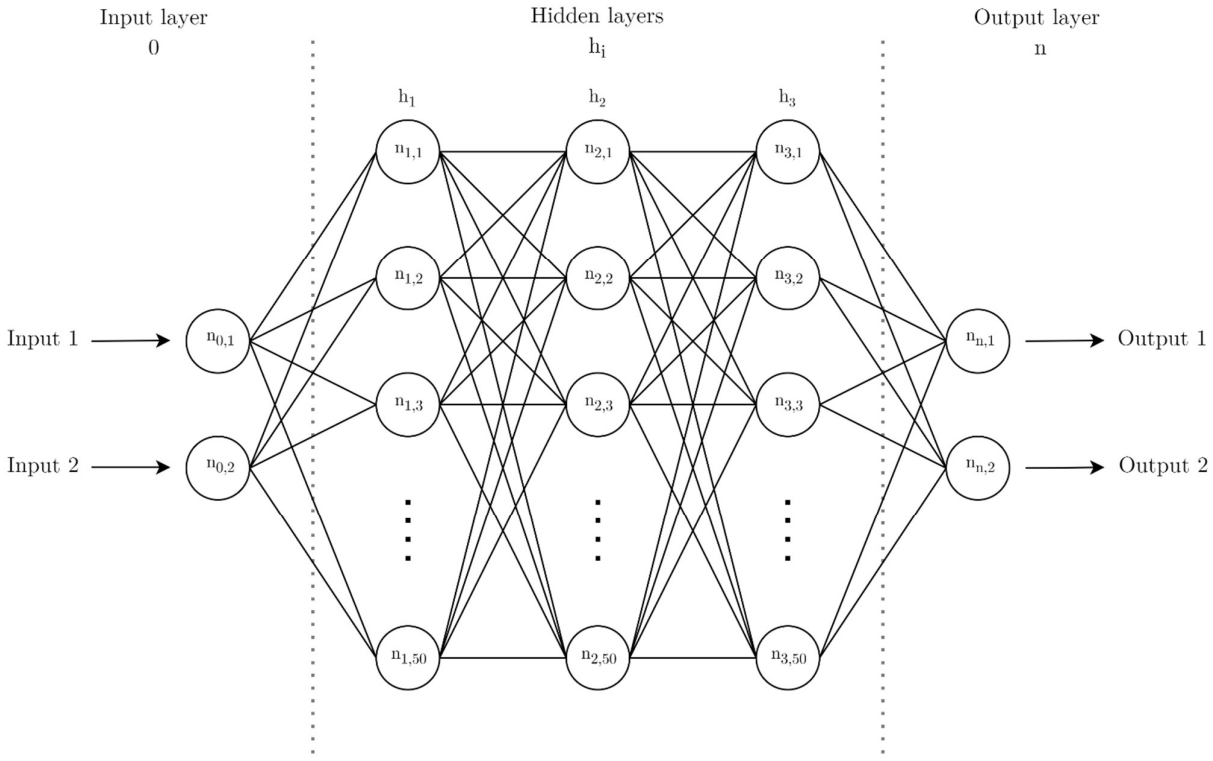

Figure 9: Example of F.D.N.N. architecture.

Neuron-to-neuron connection: After defining each layer in the N.N., there is the need for defining how each layer is connected. The connections are between the neurons, where each neuron in one layer in the N.N. is connected to each neuron in the next layer. Each neuron in one layer defines an output, with its input given by all neurons the previous layer. The mapping between these inputs and outputs defines the neuron-to-neuron connection and is chosen by the modeler. The connections in this example (and for the remaining models of this thesis), as shown in figure 10, consists of three steps: one linear transformation, a probabilistic nullifying, and one non-linear transformation (an activation function). As a result of the linear transformation, each connection has its own parameters called weight and bias associated with it. These are the fitting parameters updated in the optimization step in the training sequence mentioned with figure 7.

The linear transformation (eq. (45)) is carried out by calculating the weighted sum of the value a_k^i of each neuron n_k^i with respect to the associated weight $w_{k,j}^{i,i+1}$ and biases $b_{k,j}^{i,i+1}$ in connection with neuron n_j^{i+1}. Index i represents layer number, k the neuron in layer i, and j the neuron in layer $i+1$.

$$\tilde{a}_j^{i+1} = \sum_{k=1}^{n}(a_k^i w_{k,j}^{i,i+1} + b_{k,j}^{i,i+1}) = \sum_{k=1}^{n}(a_k^i w_{k,j}^{i,i+1}) + b_j^{i,i+1} \tag{45}$$

The value of the weights $w_{k,j}^{i,i+1}$ can be view as indicating "how strong" the connection is between each neuron, that is: if the connection between neuron $n_{k=j}^{i}$ in layer i and neuron n_j^{i+1} in layer $i+1$ has a weight $w_{k=j,j}^{i,i+1}$ with relatively large magnitude compared to the weights $w_{k\neq j,j}^{i,i+1}$ with respect to the surrounding neurons $n_{k\neq j}^{i}$, neuron $n_{k=j}^{i}$ will affect the value of neuron n_j^{i+1} the most.

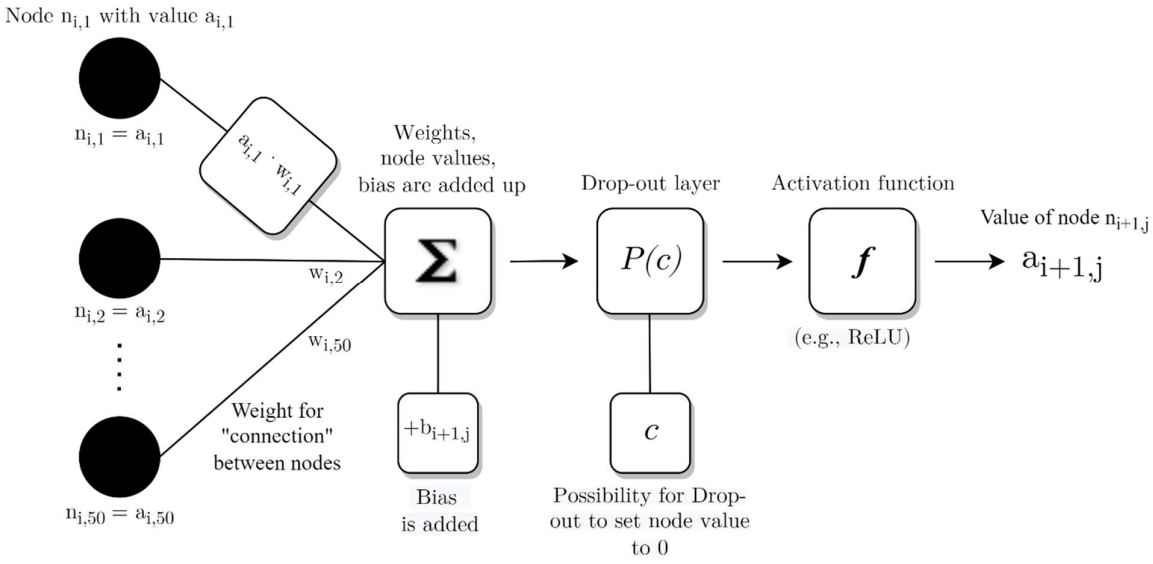

Figure 10: Neuron-to-neuron connection.

The next step is a probabilistic nullifying, a "dropout", as described in eq. (46), where $P(c)$ read 'with probability c'. This step takes in the value $\tilde{a}_j^{i+1}$ of neuron n_j^{i+1} and sets $\tilde{a}_j^{i+1} = 0$ by a given possibility c. This "dropout" is a regularization technique which helps the N.N. during the training and is a way to reduce the number of neurons in the network, which can help with overfitting to the training data.

$$\hat{a}_j^{i+1} = \begin{cases} \tilde{a}_j^{i+1}, & with\ P(c) \\ 0, & otherwise \end{cases} \tag{46}$$

The last part of the connection is a non-linear transformation, which is referred to as an activation function within N.Ns. These are introduced to give the network the possibility to mimic non-linear functions, as a sequence of linear transformations would only result in one big linear transformation due to the principle of superposition. The non-linear transformation used in this example (and throughout the remainder of this thesis) is known as the Rectified Linear Unit (ReLU) activation function, as shown in eq. (47) and visualized in figure 11 below.

$$\varrho(\hat{a}_j^{i+1}) = max(0, \hat{a}_j^{i+1}) = \begin{cases} \hat{a}_j^{i+1}, & \hat{a}_j^{i+1} > 0 \\ 0, & \hat{a}_j^{i+1} \leq 0 \end{cases} \tag{47}$$

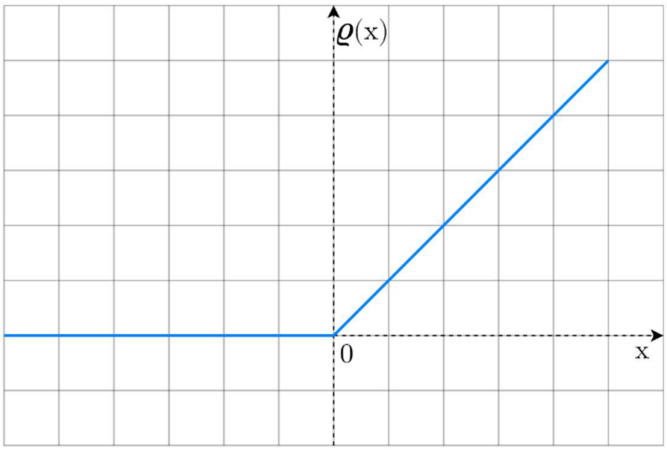

Figure 11: ReLU activation function graph.

This yields the equation for the value of each neuron:

$$a_j^{i+1} = \begin{cases} \varrho\left(\sum_{k=1}^{n}(a_k^i w_{k,j}^{i,i+1}) + b_j^{i,i+1}\right), & with\ P(c) \\ 0, & otherwise \end{cases} \tag{48}$$

The only exception to this equation is the connection between the last "hidden" layer and the output layer. Here, the linear transformation eq. (45) is the only step in the connection, though this can vary from problem to problem. This type of final connection is common in the case of regression problems, as in this thesis, though for classification problems one might use a SoftMax or Sigmoid function (Sharma, Sharma, Athaiya, 2017).

4.2.1 Basic N.N. results:

The following section shows the results of the plain D.D.M. presented in section 4.2 applied to the example problem of mapping polar coordinates to Cartesian coordinates. As a reminder, the N.N. was trained with data of polar coordinates for $\theta \in [0, 2\pi]$ and $r \in [0, 100]$. For testing the D.D.M., multiple datasets of polar coordinates along the boundary of different circles were used as inputs to see whether the D.D.M. managed to reproduce the same circles in Cartesian coordinates. The test data also contained out-of-distribution data (O.O.D.), data which lie outside the initial training data, used for evaluating the model's generalization. Table 5 shows the different angles and radii for which the circles were represented

in polar coordinates and accuracy of the results from the D.D.M in comparison to the real solution. Figure 12 shows the plots of the same circles as predicted by the D.D.M.

Table 5: Accuracy of DDM for unscaled data for mapping polar to Cartesian coordinates.

Radian	$[\mathbf{0}, \mathbf{2\pi}]$					$[-\mathbf{2\pi}, \mathbf{0}]$					$[\mathbf{2\pi}, \mathbf{4\pi}]$				
Radius	0.5	1	10	50	100	0.5	1	10	50	100	0.5	1	10	50	100
Accuracy	4%	18%	66%	87%	93%	0%	0%	1%	5%	0%	0%	0%	3%	8%	6%

As table 5 shows, the D.D.M. in its current state only manages to produce valid results for circles with the largest radii, and for angles within the same period as the training data, $[0, 2\pi]$. For the smaller radii, figure 12 shows that the D.D.M. manages to predict something resembling a circle, but the results are far from satisfying. For angles within $[-2\pi, 0]$ and $[2\pi, 4\pi]$, table 5 shows that the D.D.M. is not able to predict the circles, and from figure 12 f), one can see that predictions are closer to a straight line. Every prediction for circles within the latter periods produces a similar plot to figure 12 f), therefore, these plots are omitted from this thesis. Figure 12 f) also shows another important results, which will also appear for other D.D.Ms. in this thesis: As the periods lie outside the training data (i.e., a O.O.D.), the D.D.M. does not manage to predict these circles due to the D.D.Ms. poor ability to extrapolate, which is a well-known but nevertheless important conclusion of this exercise.

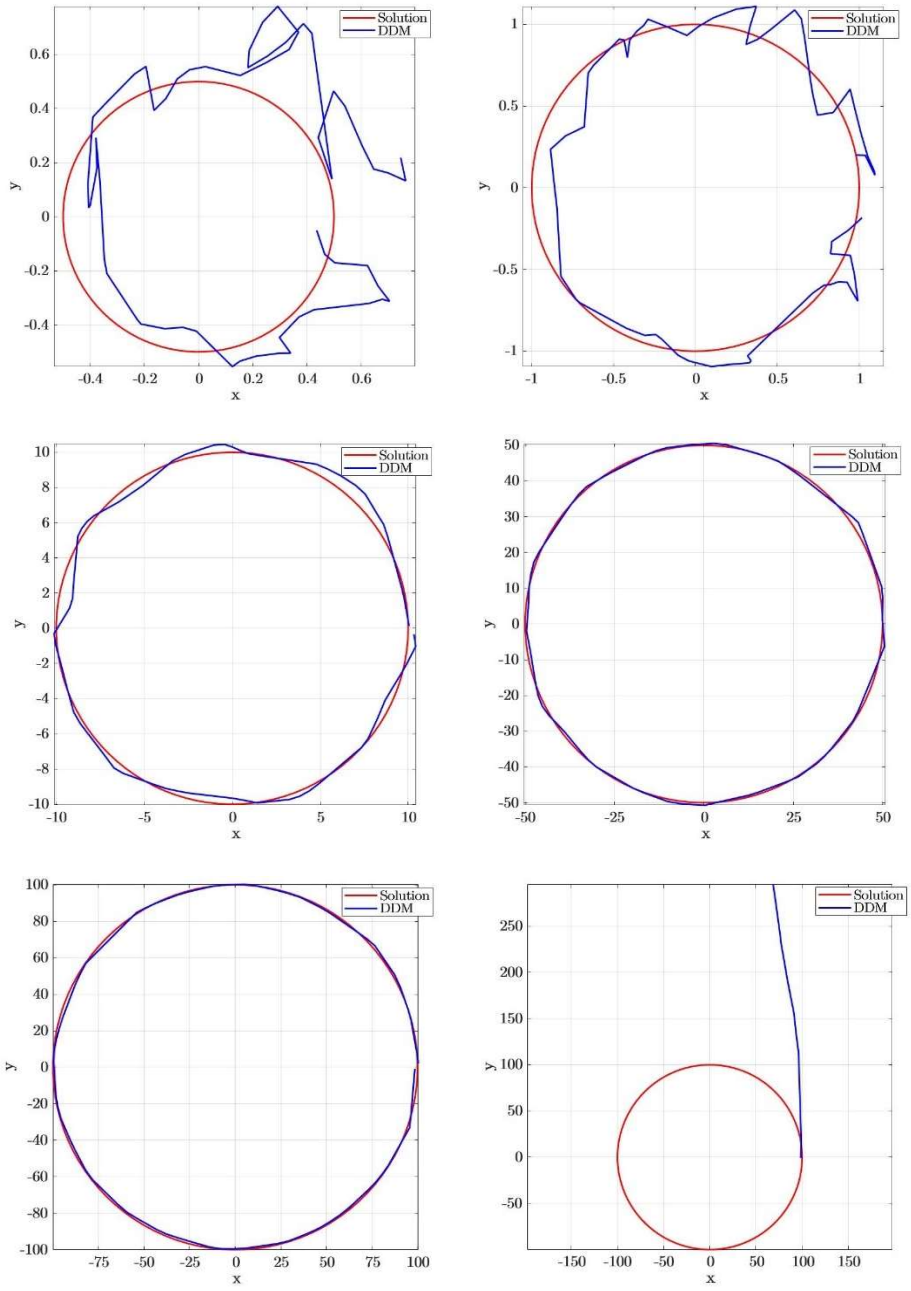

Figure 12: plot of DDM predictors vs. exact solutions for $\theta \in [0, 2\pi]$:

$a) r = 0.5, \quad b) r = 1, \quad c) r = 10, \quad d) r = 50, \quad e) r = 100, \quad f) \theta \in [2\pi, 4\pi], r = 100$

4.2.2 Network optimization:

Even though a N.N. can be constructed in countless ways and thereby yield multitudinous outcomes, there is still no guarantee that there exists a given configuration that will give satisfying results. As seen in section 4.2.1, a plain N.N. may not be sufficient. This is where different optimization techniques come to hand. Optimization techniques are direct implementations in the N.N. itself or in data pre-processing which can improve the results in numerous ways. One problem, overfitting as mentioned in section 4.2.1, can occur when the N.N. is too large or the N.N. is overtrained. *Regularizations*, some of the following optimization techniques, can help overcome this overfitting problem by either reducing the size of the N.N., by adding a penalty to the loss function, or by halting the training process of the N.N. Another problem can occur when the data given to the N.N. is not optimized. Here, *data scaling*, either as pre-processing or during training, can be implemented. This can help optimize the learning process of the N.N. by improving convergence of the loss function. One can also apply *downsampling* to the data to allow for more coarse data, which can allow the N.N. to learn correlations in data more easily. Another technique is *pooling*, where one gathers the data and only extracts the most prominent features, which can reduce the complexity of the data. The last optimization technique is *hyperparameter optimization,* where one fine-tunes the hyperparameters of the D.D.M. to improve the results in various ways. The mentioned optimization techniques only account for a handful of the optimization techniques available in the literature out there, but this thesis will focus on mentioned. In this following section, the different optimization techniques used in the D.D.Ms. throughout this thesis will be presented in more details.

Data scaling: One particularly important step to D.D.Ms. is the data itself. A big problem with data lies in the optimization step in the N.N. pipeline, where it updates the parameters in the N.N. with respect to the minimization of the loss function. If the dataset contains data of different scales, or data with large dispersion, the loss calculated from data of largest magnitudes will be prioritized by the optimization step. This is due to the fact that a change in the parameters associated with the path of the loss function backwards through the network would result in a greater minimization of the loss function, compared to the loss from data of smaller magnitude. This gives the need for data scaling, such that all the data in the dataset are set within the same magnitude, and thus, the network will not prioritize some data over other. This also helps improve the convergence of the loss function eq. (43). In this thesis standardization (or z-score normalization), eq. (49), is used to scale the data to get a mean of 0 and standard deviation of 1.

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x} - \mu}{\sigma} \tag{49}$$

where μ is the mean and σ is the standard deviation of the given distribution, $\boldsymbol{x}$ is the datapoint, and $\hat{\boldsymbol{x}}$ is the standardized datapoint.

Downsampling: If one is looking at dataset with e.g., timeseries (as will be the case for the main D.D.M. of this thesis in section 4.3) or other values from continuous functions with fine discretization, there may be too little change between datapoints, which can result in the N.N. not being able to learn the relationship between the points. In this case, downsampling may be an option. Downsampling in this case uses the same approach as downsampling in signal processing. The way downsampling is implemented in this thesis, as shown in example form in figure 13, is that the entire dataset is still used for training the N.N., but the solution the N.N. is comparing its predictor to is m steps from the input datapoint (i.e., $x_i \rightarrow x_{i+m}^{pred}$). From the N.N. point-of-view, the dataset is thereby downsampled by a factor 3. The discretization in the predictions of the N.N. will become coarser, but in return, the dataset will be larger and more detailed in comparison to training with an already coarse discretized dataset. Downsampling will not be implemented with this example but will be a key element of the main D.D.M. in section 4.3.

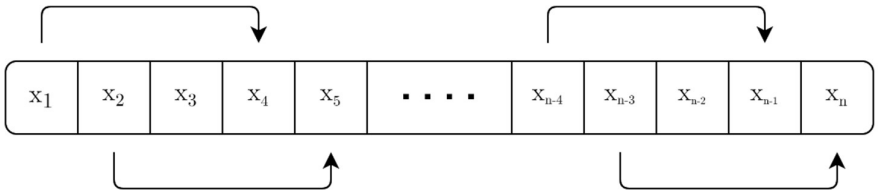

Figure 13: Example of downsampling with m=3; arrows shows which value DDM(x_i) should compare its predictor with (x_{i+3}).

Pooling: When constructing a N.N., one may end up with some part that contains too much information during a stage in training. The N.N. may correspondingly end up overfitting (i.e., adapting to the specific training-data, no generalizing) or may not be able to learn anything at all. It can be due to too large hidden layers, input-data with too many dimensions, or data too large along one dimension etc. One solution to this problem is the use of pooling. This can be done by taking all the data at a certain point in the Feedforward process, pooling them together, and returning some downsized version. This downsizing can be done in many different ways, with respect to different operations. In this thesis, the MaxPooling method is used. Here, the max value of a certain enclosed part of the data is used to shape a new representation of the data, as seen in figure 14. As with

downsampling, pooling will not be implemented in this example but will be implemented within the main D.D.M. in section 4.3.

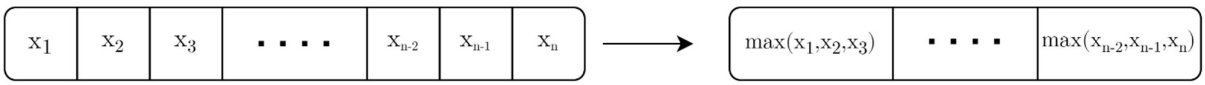

Figure 14: Example of MaxPooling.

Regularization: Regularization techniques are implementations one can make that reduces the chance of overfitting. There are many such techniques, but three are in focus in this thesis. The first technique is "*drop-out*"-regularization, as mentioned in more detail previously in section 4.2. In short, "drop-out" reduces the complexity (the size) of the N.N. and can help with overfitting.

The next is *L2-regularization*, which is a penalty term added to the loss function eq. (43):

$$\hat{f}_{loss} = \frac{1}{N}\sum_{j=1}^{N}\|\boldsymbol{y}_j^{pred} - \boldsymbol{y}_j^{sol}\|^2 + \lambda\sum_{k=1}^{M} {w_k}^2 \rightarrow min \tag{50}$$

with w being the weights and λ the regularization parameter; a scalar to weight the penalty term. In broad, it works the same way as "drop-out", but instead of setting whole neurons to zero, it acts on the loss function such that larger weights get a larger penalty and gets reduces toward zero. Smaller weights, on the other hand, will be viewed as more optimal and not given such a large penalty. All the weights in the network will thereby be reduced towards zero and get a more even distribution. One can also use *L1-regularization*, which is the same concept but with the use of absolute values of the weights instead of the square.

The last regularization technique is "*early stop*", a straight-forward concept to implemented in the training-loop which stops the N.N. when no learning is taking place. It can be done in different ways; in this thesis, it is a monitoring of the loss function, and works such that if the loss function during the validation step of the N.N. does not decrease over a given number of iterations (epochs), the training stops. By looking at which iteration the global minimum of the loss function appeared, one can get a good idea of when to stop the training during a new training session, though this iteration number may vary from session to session due to the N.N.s random initialization of parameters (initial parameters are not actually random, but for the simplicity of this thesis, they will be viewed as random, see Yam, Chow (2000) for examples).

Hyperparameter optimization:

As mentioned earlier in chapter 4, there are many different choices modelers have to make when designing a D.D.M. Except for the overall architecture, loss function and optimization algorithm, there are a handful of parameters one can set up and fine-tune to optimize the model, known as hyperparameters (Goodfellow, Bengio, Courville, 2016). The optimization of these hyperparameters is one of the most basic tasks in machine learning and there are several ways to do so (Hutter, Kotthoff, Vanschoren, 2019). This thesis uses a basic, more hands-on hyperparameters tuning-technique, which consists of training the N.N. for a certain set of hyperparameter values and reviewing the results to find the optimal values of the given hyperparameters (Claesen, De Moor, 2015). The following hyperparameters were considered when tuning the model:

- **Learning rate**: the step-size α_k for eq. (44).
- **Epochs**: number of times to iterate over the whole training dataset.
- **Mini-batch size**: size of the subsets of the training dataset during one epoch, for which gradients and losses are calculated and optimized.
- **Drop-out**: the probability c, see section above.

Table 6 shows the results for different hyperparameters in comparison to accuracy and loss of both training and validation for the example problem given throughout section 4.2. The table only shows the 10 best results, as showing the rest does not serve any purpose for the rest of the thesis. One can see that there is no direct correlation between the different hyperparameters and the results, except the exclusion of dropout, which for this specific N.N. gave the best results. A learning rate of $\alpha = 0.0005$, mini-batch size of 64, and drop-out $P(0)$ coincidentally gave the best results for the other D.D.Ms. in this thesis, so these hyperparameters will be kept at those values throughout the thesis.

Table 6: Result of loss and accuracy with respect to hyperparameters of different values.

Hyperparameters			Learning results			
Learning rate	Mini-batch size	Drop-out	Training accuracy	Training loss	Validation accuracy	Validation loss
0.0005	64	0.0	0.7840	0.0004325	0.8444	0.0001737
0.0001	32	0.0	0.7773	0.0005654	0.8433	0.0002178
0.0001	64	0.0	0.7892	0.0005493	0.8243	0.0002386
0.0005	32	0.0	0.7518	0.0005998	0.8255	0.0002654
0.0005	256	0.0	0.7630	0.0005605	0.7923	0.0003016
0.0005	128	0.0	0.7669	0.0004923	0.7981	0.0003170
0.005	128	0.0	0.6827	0.0009614	0.7926	0.0003539
0.0005	128	0.01	0.6169	0.001477	0.7966	0.0003931
0.0005	64	0.01	0.6305	0.001321	0.7977	0.0004037
0.0001	32	0.01	0.5545	0.002325	0.7871	0.0004419

4.2.3 Optimized N.N. results:

The following results use the same setup as in section 4.2.1; the only exception is the implementation of optimization techniques, more specifically hyperparameter optimization, early stop, and data scaling. The data scaling in this D.D.M. is implemented a little different from the standardization used in the rest of the thesis. Here, $\boldsymbol{y}^{pred}$ and $\boldsymbol{y}^{sol}$ are both scaled by the L^2-norm of the solution $\boldsymbol{y}^{sol}$ before the loss function is calculated, such that there is an in-training data scaling. This prohibits the loss function from prioritizing large values when minimizing the loss function (i.e., for this examples, points at large radii). As the problem for the D.D.M. in section 4.2.1 was the bias toward learning points at large radii, this implementation should reduce the error for points at small radii.

Table 7: Accuracy of DDM for scaled data for mapping polar to Cartesian coordinates.

Radian	$[\mathbf{0}, \mathbf{2\pi}]$					$[-\mathbf{2\pi}, \mathbf{0}]$					$[\mathbf{2\pi}, \mathbf{4\pi}]$				
Radius	0.5	1	10	50	100	0.5	1	10	50	100	0.5	1	10	50	100
Accuracy	90%	87%	86%	76%	74%	4%	5%	5%	9%	0%	6%	6%	2%	2%	0%

Table 7 shows results for the same datasets as in section 4.2.1, though for this D.D.M. the predictions for the smaller circles have a much higher accuracy than for the D.D.M. in section 4.2.1. This implies that the data scaling implementation worked as expected. One interesting fact though, is that the accuracy for the larger

circles has dropped significantly. A more evenly spread accuracy for the different radii was expected. This reduction in accuracy for large radii can also be clearly seen in figure 15. The reason behind this reduction is unfortunately hard to point out. Table 7 and figure 15 f) also show, as in 4.2.1, that the D.D.M. do not handle extrapolation. This could be solved by including a larger dataset with angles over multiple periods or introducing a totally different N.N. architecture. This is not implemented as this problem only serves as an example and an introduction to the final D.D.M. of this thesis, and therefore these implementations would not be of interest.

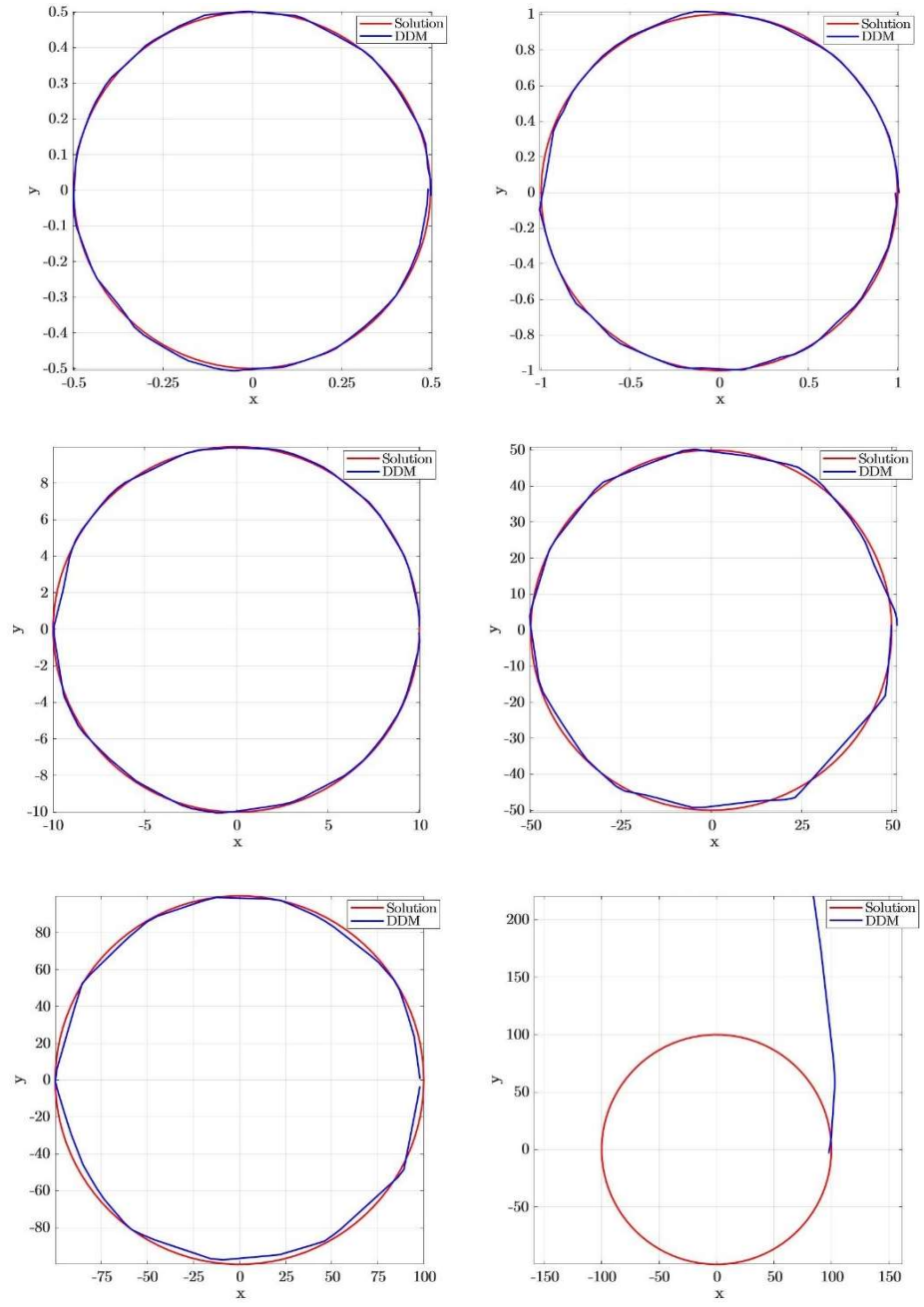

Figure 15: plot of DDM with normalization predictors vs. exact solutions for $\theta \in [0, 2\pi]$:

a) $r = 0.5$, *b)* $r = 1$, *c)* $r = 10$, *d)* $r = 50$, *e)* $r = 100$, *f)* $\theta \in [2\pi, 4\pi], r = 100$

4.3 D.D.M. for predicting planetary motion

The main goal of this section is to see how well a D.D.M. based on the basic F.D.N.N. in section 4.2 can perform in comparison to the P.B.M. in chapter 3 for describing planetary motion. The dataset from NASA, presented in section 3.2, will also be used in this D.D.M. as training/validation data, as well as data for testing the D.D.M.

The D.D.M. in this section is heavily based on the D.D.M. from 4.2, though there are some changes that will be established along the way. The way the D.D.M. and the results are presented in this section is by slowly increasing the complexity of the problem and do one implementation for improvement at a time, to isolate and clearly understand the different implementations and consequently improvements in the results. One substantial change from the previous D.D.M., which is added right from the start, is the input to the N.N. As the D.D.M. is trying to predict planetary motion closely resembling the P.B.M., a natural approach is to give the N.N. a state vector $\boldsymbol{s} = (\boldsymbol{x},\ \boldsymbol{v}) = (x,\ y,\ z,\ v_x,\ v_y,\ v_z)$ at a time t_i, and let it predict the state vector at time t_{i+1}, that is:

$$\mathcal{N}(\boldsymbol{s}_t) = \boldsymbol{s}_{t+1}^{pred} \tag{51}$$

Since the position and the velocity of one body is of different magnitude, and to hopefully make it easier for the N.N. to see connection between values, a vector $\boldsymbol{x}^{input} = (\boldsymbol{x}_{i-1},\ \boldsymbol{x}_i) = (x_{i-1},\ y_{i-1},\ z_{i-1},\ x_i,\ y_i,\ z_i)$ is used instead of the general state vector $\boldsymbol{s}$, where index i represent timestep**.** This results in:

$$\mathcal{N}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) = \boldsymbol{x}_{i+1}^{pred} \tag{52}$$

for each body. This approach removes the velocity component as a whole, and one would not be able to predict the velocity with help of the D.D.M. directly. This is not seen as an issue though, as velocity is a derivative of position, the motions of the planetary system can be described by predicting only positions for two successive timesteps. It is therefore hypothesized that the D.D.M. can make valid predictions without velocities.

While training, the D.D.M. is given one input by NASA and predicts only one step with a step-size of $\Delta t = 10$min. The D.D.M. cannot be trained to predict positions at smaller timesteps, as the training data from NASA has, as stated in section 3.2, a discretization with step-size of $\Delta t = 10$min. Also, the loss function to be minimized during training is now given as:

$$f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\|\boldsymbol{x}_{j,i+1}^{pred} - \boldsymbol{x}_{j,i+1}^{sol}\|^2 \rightarrow min \tag{53}$$

When testing the D.D.M., two consecutive initial values are given from the NASA data, and the D.D.M. is tasked with predicting full orbits by using its previous prediction as the next input, that is:

$$\mathcal{N}(\boldsymbol{x}_{i-1}^{pred}, \boldsymbol{x}_{i}^{pred}) = \boldsymbol{x}_{i+1}^{pred} \tag{54}$$

The initial test data also lies toward the end of the training dataset, such that when the D.D.M. makes its prediction, it makes predictions both within and outside the training dataset in time.

4.3.1 One-body prediction

Basic F.D.N.N.: The first case is a simplified one, where only one body's orbit is being predicted. As mentioned above, the N.N. takes in the position at two successive timesteps, $\boldsymbol{x}^{input} = (\boldsymbol{x}_{i-1},\ \boldsymbol{x}_{i})$, and returns the position at the following timestep, $\boldsymbol{x}_{i+1}^{pred}$. The data from NASA, which provides $\boldsymbol{x}^{input}$ and $\boldsymbol{x}_{i+1}^{sol}$, is pre-processed by standardization as presented in section 4.2.2. The data used for training, validation and test is inside a timeframe of 10 periods for the specific body. By the use of the basic D.D.M. presented in 4.2, the model produces the following results:

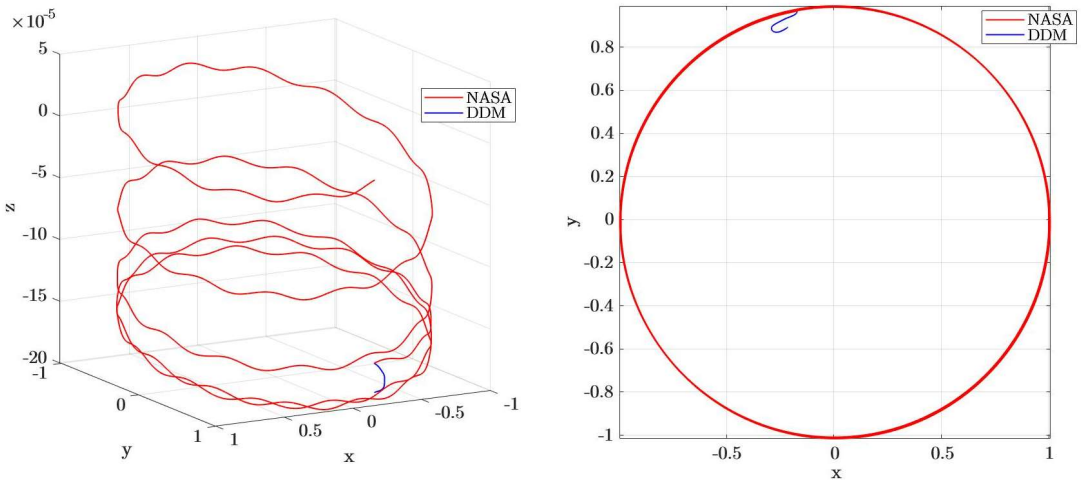

Figure 16: plots of DDM predictors vs. NASA data for one planet over 5 periods.

Table 8: Average absolute and relative error by DDM for one planet.

Avg. absolute error	Avg. relative error
1.372	0.001893

As figure 16 shows, the D.D.M. did not manage to predict anything close to resembling an orbit. The reason for these predictions may be a result of the data having to fine discretization, which can result in the N.N. not being able to differentiate between points as well as it should. A solution to this problem may be to use a larger timestep when producing the dataset for training and validation. This, however, limits the amount of data for the N.N. to learn from. Another approach, which will be introduced next, is to include downsampling.

Downsampled data: As mentioned as an optimization technique in section 4.2.2, one can apply downsampling to the input data to increase apparent coarseness of the dataset used in training and validation. By the method in figure 13, the amount of data is almost preserved, but the N.N. may be able to see connections between datapoints more efficiently. The following results are obtained by introduction this downsampling, with a downsampling of $m = 288$ (as $\Delta t = 10\text{min}$ for the dataset, $m = 288$ results in the D.D.M. being able to predict a position every 2nd day):

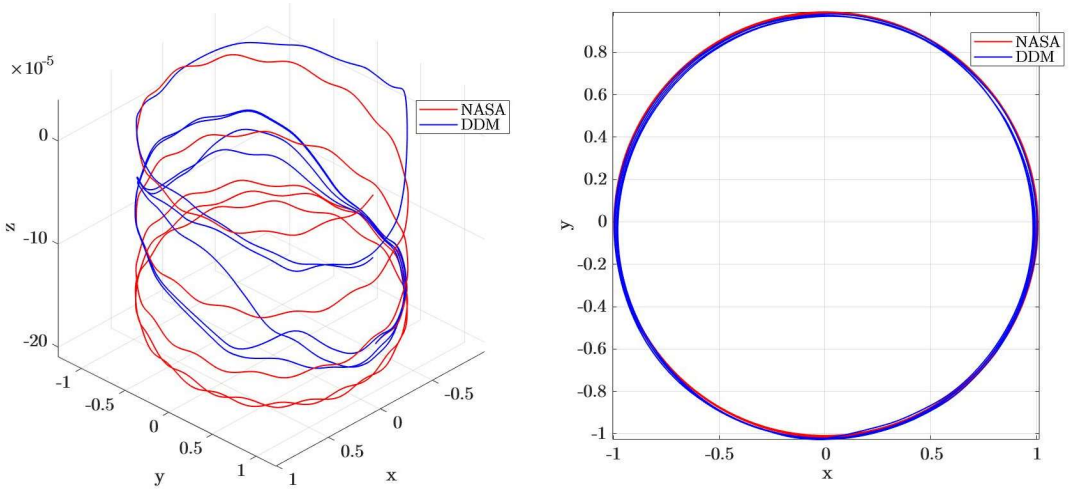

Figure 17: plots of DDM with sampling predictors vs. NASA data for one planet over 5 periods.

Table 9: Average absolute and relative error by DDM with sampling for one planet.

Avg. absolute error	Avg. relative error
0.02266	0.0007500

As figure 17 and table 9 show, the N.N. is satisfyingly able to predict the orbits of one body. As figure 17 also shows, the D.D.M. was able to partially capture the oscillations in the z-axis and the shift in the z-axis for the full orbits in the x, y-plane. The reason for not capturing these shifts and oscillations better, may be due to the fact that these orbits lie outside of the training data. Another reason may be the downsampling, which may result in much of the information for the changes in the z-axis disappearing. This fits well with the results of example problem in section 4.2, as the results indicated that the D.D.M. did manage to learn well within the domain of the given dataset and performs interpolation well, though the model do no handle extrapolation tasks well. This also fits well with the results of Wang et al. (2021), where it is shown that D.D.Ms. struggles with extrapolation of dynamical systems. It should be emphasized the smallness of the amplitude in the z-direction compared to the x- and y-axes; the deviation in the z-axis is therefore acceptable.

4.3.2 Multi-body prediction

Single N.N: The next step is to extend the problem to multiple bodies. This D.D.M., as in the D.D.M. above, has implemented the optimization techniques: downsampling, data scaling, hyperparameter optimization, and early stop, as described in section 4.2.2; however, there are a couple of differences in comparison to the simplified case in section 4.3.1:

One of the main differences here is the input to the N.N. As there are 8 celestial bodies considered (as the Sun is fixed at the origin and thus not considered), the input to the N.N. is a 6×8 tensor, that is, $\boldsymbol{x}^{input} = (\boldsymbol{x}_{i-1},\ \boldsymbol{x}_i)$ for each body. Another difference is the change of training dataset. As the D.D.M. above only considered one body, it was not necessary to include a large dataset over a large timeframe, only around ten periods were sufficient. As the D.D.M. now considers 8 bodies, the training of the N.N. requires data sufficient to learn from multiple periods for each body. This results in a large dataset which spans at least 225 years. As this dataset is very large, it was reduced to a discretization with $\Delta t = 1$h. This is believed to be sufficiently small timesteps to still maintain the essential information in the data. Yet, another consideration one has to make when considering multiple bodies is the difference in orbital periods T (see table 17 in Appendix A.1 for orbital periods for each body). As the difference in the innermost and outermost bodies orbital periods is quite large, one has to make some compromises. From section 4.3.1, the results showed that a downsampling of $m = 288$ was sufficient for a planet with that specific orbital period. An assumption is made that optimal results are obtained with:

$$n^{sampling} = \frac{T}{\Delta t \cdot m} = \frac{n}{m} \tag{55}$$

where n is the number of points along an orbit for one orbital period, and $n^{sampling}$ is the number of points along an orbit for one orbital period after sampling is applied, and the latter is constant for each body. This assumption yields that the outermost planet must have such a coarse discretization along its orbit for the N.N. to be able to differentiate between points. As a compromise between the orbits for the two outermost planets, the downsampling is set to $m = 1500$, which results in the D.D.M. being able to predict positions every 62.5 days. This timestep should be a problem for the innermost planets, as this timestep accounts for most of the bodies orbital period, but as the later results show, this is not directly the case.

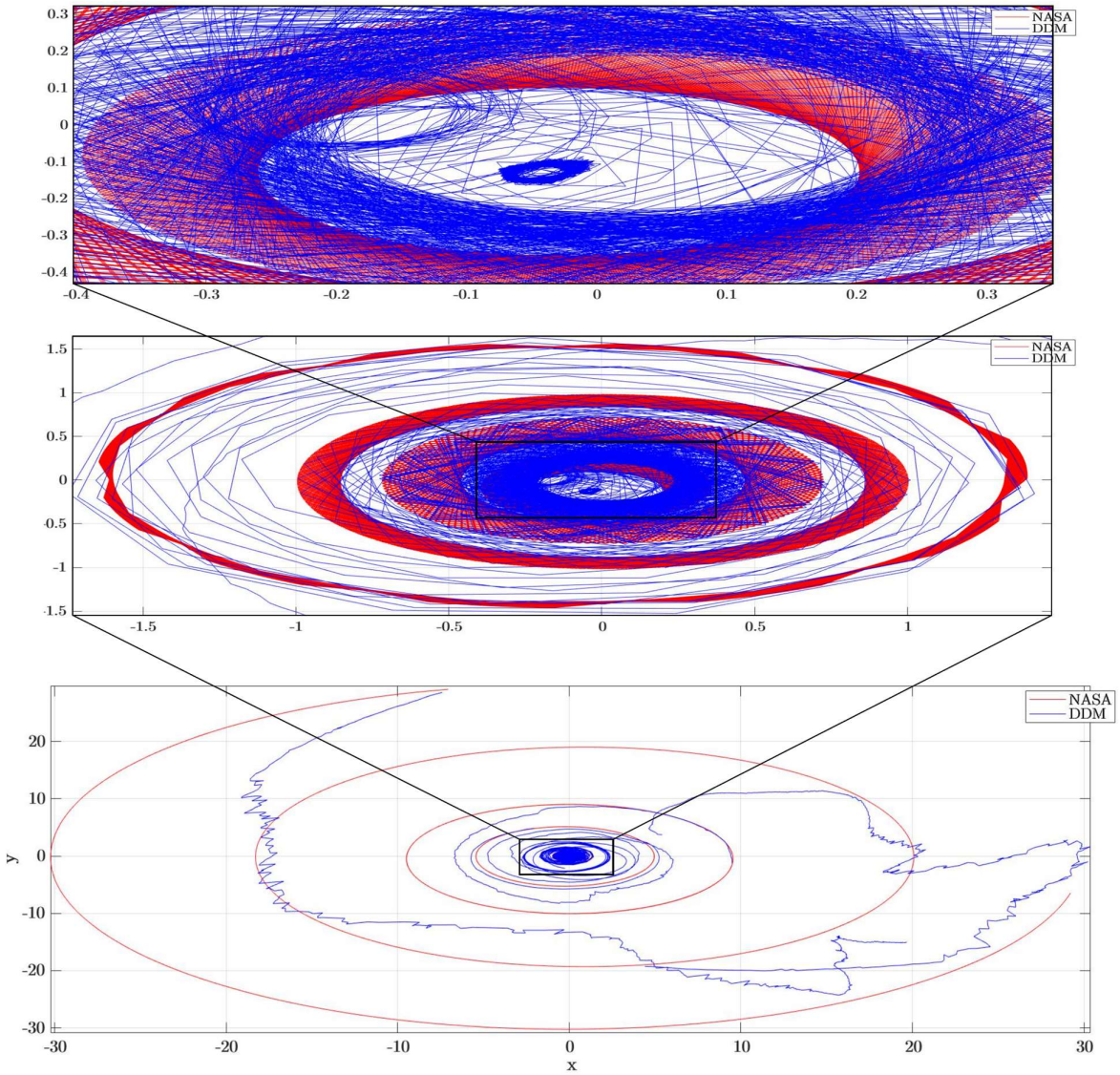

Figure 18: 2d plot of DDM predictors vs. NASA data.

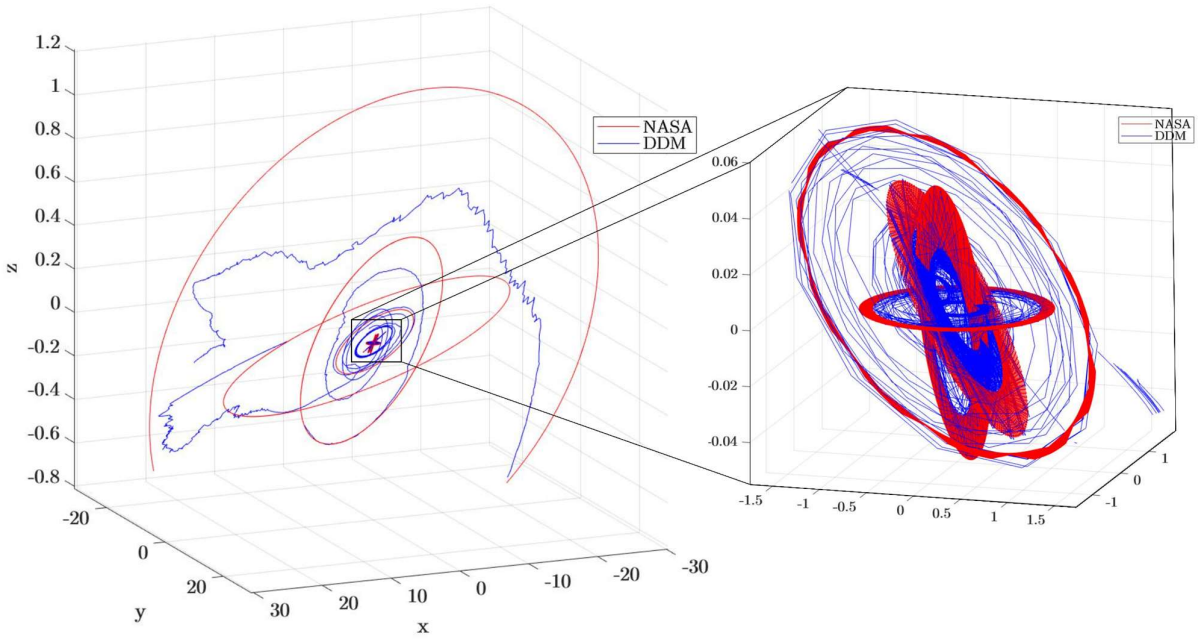

Figure 19: 3d plot of DDM predictors vs. NASA data.

Figure 18 and 19 shows the orbital prediction for each planet by the D.D.M. against the exact solution by the data form NASA. As the figures shows, the N.N. is unable to predict valid orbits for any of the planets. For the outermost planets, the D.D.M. did not come close to anything periodically. The reason may be a lack of data and thus the N.N. is not able to train these planets for enough orbital periods. This would explain the bad results of the outmost planets, but not the innermost, as these have multiple orbital periods contained within the dataset. It appears that the D.D.M. predicts some form of orbits for the innermost planets, though they seem unstable and chaotic. One could argue that the large timestep is the problem for the instability of the orbits of the innermost planets, but as later results will show, this is not the case.

Another reason for the bad results over all planets may be that, after data scaling, the N.N. received data within a unit circle, which for the N.N. the position of these datapoints may seem random. The N.N. has no direct way to differentiate between points on one orbit compared to another orbit. The data is almost like a low-density point-cloud. As the results for one-body predictions were satisfying, one could presume that the N.N., with some alterations, would be able to predict multiple bodies. Or is this like the case of the analytical solution of the n-body problem, that with more bodies comes a greater difficulty (and at a certain point, an almost impossibility) to arrive at a solution. As this thesis will show in the following case, there is a way around this problem, and this is where the concept of N.N. really shines.

Combined N.Ns.: As mentioned in section 4.1, one of N.Ns. main concepts is that N.Ns. are allowed to learn by themselves, as we humans are not in control of how the N.N. learns and how the finally trained N.N. arrives at a prediction. In our understanding and calculations of physical problems, we are constrained by our formulations, thinking methods and the problems connection to other problems and concepts; this is also the case for the n-body problem. This is, however, not the case for the N.N; it can find connections through its hidden states that we as human normally would not arrive at (Willard et al., 2020) (Blakseth, Rasheed, Kvamsdal, San, 2022). Therefore, a N.N. can be constructed in many different ways and still solve the problem, though the accuracy would be connected to how the network is set up.

This concept gives rise to the idea of setting up a global N.N. made up of many smaller local N.Ns., in comparison to the previous N.N., which was only one single large N.N. As one N.N. can predict the orbit of one body, given enough data, n N.Ns. could potentially predict orbits for n bodies. To justify this setup, a connection between the smaller N.Ns. will be applied in the form of a pooling-layer, as mentioned in section 4.2.2. This gives the possibility for interaction between the local N.Ns. such that the global N.N. can make predictions for a coupled n-body system, instead of considering each body as its own closed system. The use of a pooling-layer is inspired by Alahi et al. (2016), amongst others, where the implementation of pooling-layers has been seen to yield satisfying results. The pooling-layer in this thesis, as shown in figure 20, takes in three hidden states from each local N.N., and returns the maximum from each of the three hidden states of each local N.N. All the maximum from all the local N.Ns. are then gathered into one tensor and used as input for the next hidden layer in each local N.N, together with the previous hidden state from each respective local N.N. This means that at the middle of each local N.N., the networks are fed a value from each other network, which in some way represents the interaction between each body. As these values are hidden states of the local N.Ns., they do directly have a physical interpretation, so the number of hidden states the local N.Ns. need to share is ambiguous. This final D.D.M. is, except for this new N.N. architecture including a pooling-layer, the same as in the previous D.D.M., with same sampling, data scaling, early stop, hyperparameter optimalization and dataset.

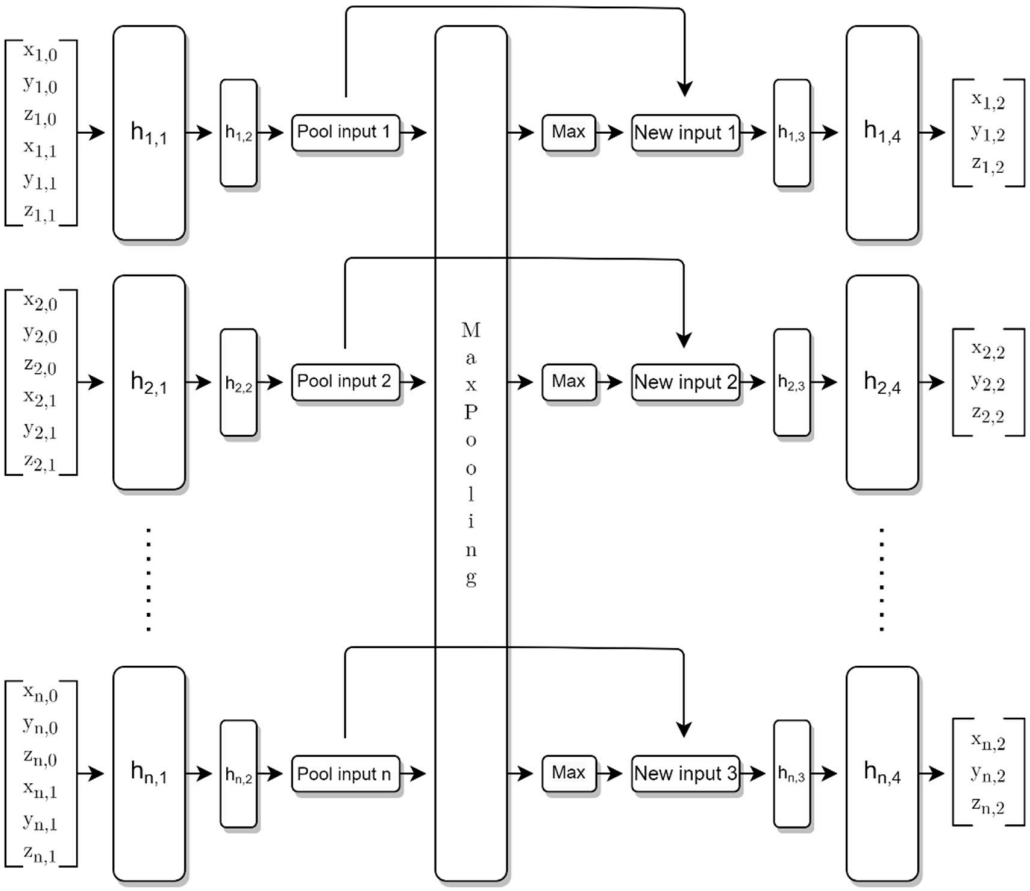

Figure 18: Global N.N. with n local N.Ns. and MaxPooling as interaction layer.

The following figure 21 and 22 shows the orbital prediction for each planet by the final D.D.M. against the exact solution by the data form NASA. This time, the predictions of the D.D.M. appears periodic and for all planets, except for the two outermost, follows the orbits produced by the NASA data well. These orbits, for both NASA and the D.D.M., are not smooth, though this is most likely a result of the large timesteps. The predictions for the two outermost planets do not coincide with the data from NASA, as in the case of the previous D.D.M. This is still most likely a result of the lack of data for multiple orbits, though for this D.D.M. the predictions more closely resemble orbits. The sudden stop in the prediction of these orbits is an accumulation of points at the point of extrapolation. This again agrees with the statement that D.D.Ms. do not extrapolate well. The outermost planet also has a large region of outlying points in the middle of the orbit. This is likely due to the same challenge as occurred with the one-body prediction in section 4.3.1. The rate of change in the position is very low for the outermost planets in comparison to the rest, which may result in the N.N. not being able to differentiate between values of two consecutive timesteps. As the downsampling was set to

hopefully accommodate both the two outermost planets, the sampling may still be too high for the outermost planet. One could be tempted to increase the sample rate, though this would further increase the time between each prediction, which is already viewed as large.

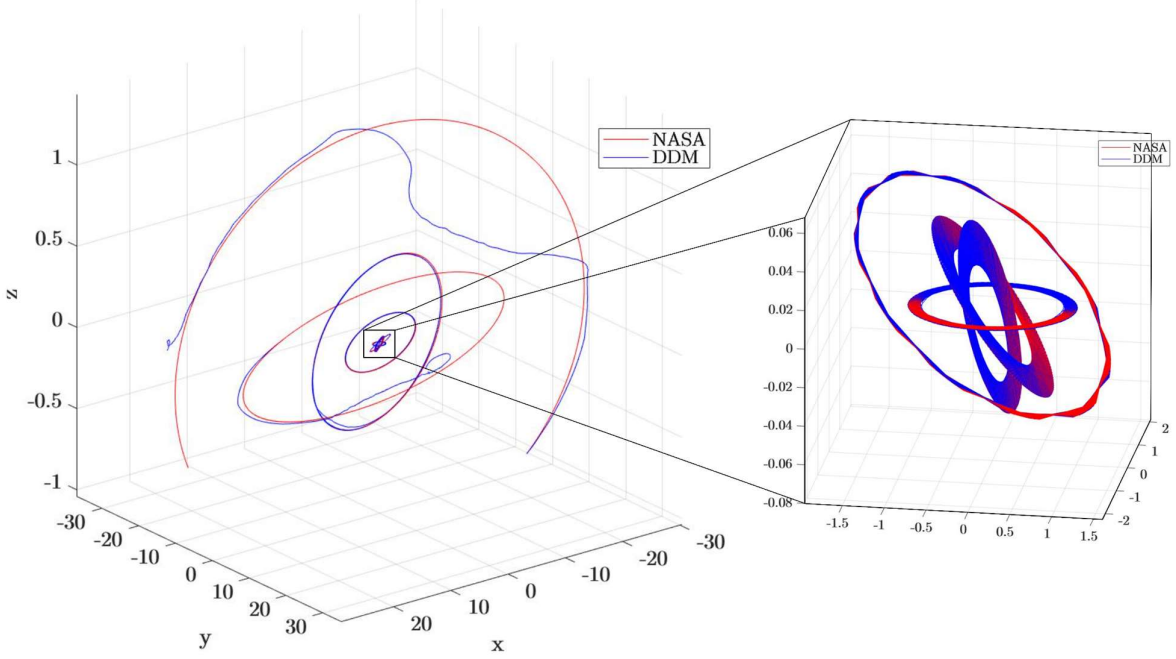

Figure 19: 3d plot of DDM with pooling predictors vs. NASA data.

The predictions for innermost planets appear stable and periodic. As the downsampling was set so large, one would hypothesis that the N.N. would not be able to predict a satisfying orbit with such coarse data, especially from a numerical point of view. As the timesteps increases drastically, one cannot expect a numerical method to produce a satisfying approximated solution. This is not the case for D.D.Ms., as the results show that the D.D.M. is not restricted in the same way by steps-sizes as standard numerical methods. By setting the steps-size such that none of the innermost planets do complete an exact period, the N.N. will be presented with the outline for the total period. As the dataset being large enough to contain multiple orbital periods of the innermost planets, after a small number of epochs, the training sequence has most likely already run through multiple periods of these planets, and thereby, the N.N. has managed to learn the orbits the N.N. should predict. This nicely demonstrate the fact that D.D.M. are only dependent on the data presented, not by the underlying physics and mathematics of the problem. The drawback to this implementation is that the practicality of the model would be highly reduced, as previously mentioned, it would only be able to predict positions for the planets every 62.5 days.

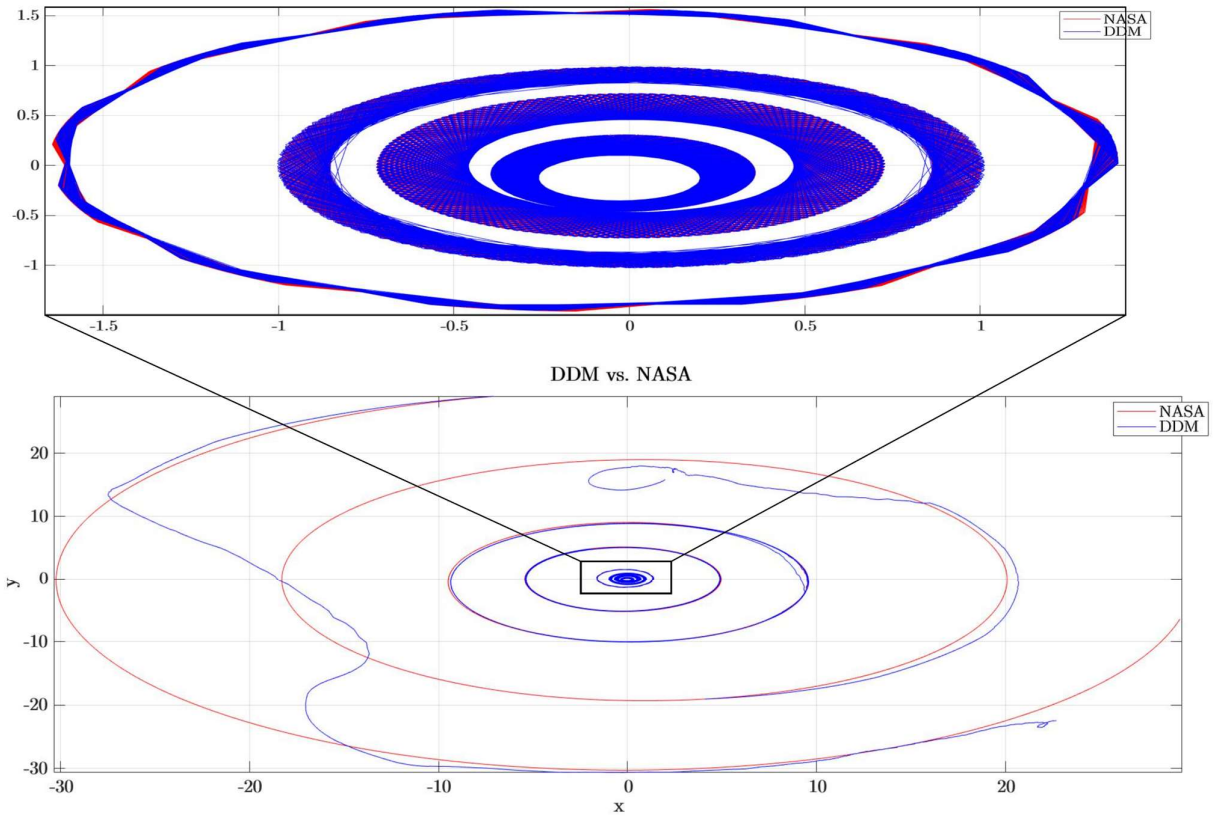

Figure 20: 2d plot of DDM with pooling predictors vs. NASA data.

Table 10 below shows the average absolute and relative error for each planet by both the pooling and non-pooling D.D.M. The table shows a reduction in the errors for every planet in the pooling D.D.M. in comparison to the non-pooling D.D.M, as was hypothesized. For two outermost planets, the absolute error in both cases is significantly greater than for the other planets, which is to be expected for orbits of such magnitude, and with a lack of data. For the relative error, the non-pooling D.D.M. produces similar errors for every planet, which is probably due to the overall incompleteness of the model. The pooling D.D.M. yields, as previously stated, overall improved results. The results also shows that the D.D.M. handles the exclusion of velocities.

Table 10: Average absolute and relative error by DDM with and without pooling for each planet.

	DDM without pooling		DDM with pooling	
Celestial object	Avg. absolute error (position)	Avg. relative error (position)	Avg. absolute error (position)	Avg. relative error (position)
Mercury	$8.4995 \cdot 10^{-6}$	$8.2992 \cdot 10^{-7}$	$1.0290 \cdot 10^{-6}$	$1.0048 \cdot 10^{-7}$
Venus	$1.0766 \cdot 10^{-5}$	$5.8023 \cdot 10^{-7}$	$2.8340 \cdot 10^{-7}$	$1.5273 \cdot 10^{-8}$
Earth	$1.5307 \cdot 10^{-5}$	$5.9658 \cdot 10^{-7}$	$2.7049 \cdot 10^{-6}$	$1.0543 \cdot 10^{-7}$
Mars	$3.6689 \cdot 10^{-5}$	$9.3260 \cdot 10^{-7}$	$8.4157 \cdot 10^{-7}$	$2.1392 \cdot 10^{-8}$
Jupiter	$7.3813 \cdot 10^{-5}$	$5.5119 \cdot 10^{-7}$	$6.5533 \cdot 10^{-6}$	$4.8936 \cdot 10^{-8}$
Saturn	$1.7349 \cdot 10^{-4}$	$7.0840 \cdot 10^{-7}$	$1.2111 \cdot 10^{-5}$	$4.9452 \cdot 10^{-8}$
Uranus	$5.6264 \cdot 10^{-4}$	$1.1303 \cdot 10^{-6}$	$5.5237 \cdot 10^{-4}$	$1.1097 \cdot 10^{-6}$
Neptune	$6.7602 \cdot 10^{-4}$	$8.7341 \cdot 10^{-7}$	$4.1414 \cdot 10^{-4}$	$5.3506 \cdot 10^{-7}$

5 Hybrid modeling of the n-body problem

As we try to model real-life physical systems, there will always be limitations, both by the human mind and tools we use. Like in the case of chapter 2, the mathematical formulation of the n-body system is limited by our knowledge of physics and mathematics, and the resulting model is ignorant of any unknown and unobserved physics (Blakseth, Rasheed, Kvamsdal, San, 2022).

When one considers complex systems with no or limited useful solutions, like in the case of n-body systems, one will most likely turn to numerical solution methods to arrive at an approximate solution. This will in turn result in even more loss of similarity between the real system and the simulated one. In the end, P.B.Ms. will have limitations derived from many different aspects and only show part of the whole governing system (Blakseth, Rasheed, Kvamsdal, San, 2022). Another aspect is the computationally demanding process through which P.B.Ms. arrive at a solution. On the other hand, P.B.Ms. gives stable and predictable results, relatively easy to implement, easily explained and can be applied to a wide range of systems and scenarios.

D.D.Ms., on the other hand, gives rise to some interesting ideas and application outside the reach of P.B.Ms. D.D.Ms. While limited by the availability of data for training and validation and thus not being as well suited for generalization as P.B.Ms., D.D.Ms. are not bounded by known physics and can develop their own relationships and patterns based on the data, and thus replicate the full physics of a system (Willard et al., 2020). The goal is therefore to combine elements of the P.B.M. and the D.D.M. to arrive at a solid model that encapsulates the strength of both models, hence the hybrid model. By having the basic physics described by the P.B.M., one can expect stable and justifiable results by the solid mathematical foundation of the numerical integrator and system equations. The D.D.M. will thereafter introduce the unknown physics to the results of the P.B.M., such that the final results yield a much closer approximation to the exact solution of the problem. The D.D.M. could also correct any numerical errors which appears. Also, by introducing the D.D.M., one will likely have the possibility for implementing a simpler P.B.M. such that the overall computational cost is lowered. These hybrid models are still restricted by availability of data, but early attempts have already shown promising results on improved accuracy of such models (Willard et al., 2020).

In this chapter, a hybrid model will be suggested and discussed for improving the results of the P.B.M. from section 3.3. This improvement will be accomplished by predicting residuals, in the form of an additive term, using the D.D.M. from section

4.3.2. As in the case of chapter 4, important observations and improvements will be introduced along the way, where in 5.2.2 the final hybrid model will be presented with results.

5.1 Constructing the hybrid model

When constructing a hybrid model, there are three main aspects to consider: (1) the P.B.M., (2) the D.D.M., and (3) how the latter parts are combined. (1) is usually straight forward as the P.B.M. generally only serves the purpose of obtaining an approximation to the real solution. Though, there is a choice to be made, which in essence is the complexity of the P.B.M. More complex P.B.Ms. are usually more accurate but also computationally heavy; thus, one may want to choose a simpler model to lower the computational cost. (2) is very ambiguous and relies heavily on how the D.D.M. and P.B.M. should work together; thus, (3) must first be defined. This thesis uses a hybrid model based on two simpler hybrid models: a basic hybrid-physics-data model (H.P.D.M.), and a residual model (R.M.). Both concepts are introduced in the following.

Remark: The equations following in this section uses **s** as input and output variables instead of $\boldsymbol{x}$, as used throughout chapter 4. The motivation for this change of variables is that the model now considers state vectors $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{v})$, not only positions $\boldsymbol{x}$**.** This do not change the fundamentals of the equations, only simplifies notation.

Hybrid-Physics-Data model: The H.P.D.M. (Daw et al., 2017). is a model which uses the P.B.M. as mentioned above, but the D.D.M is designed and used in such a way that it takes in the output from the P.B.M. as its input:

$$PBM(\boldsymbol{s}_i) = \hat{\boldsymbol{s}}_{i+1} \tag{56}$$

where in the context of planetary motions is the state vector of the system, and outputs an improved approximation to the real solution, that is:

$$DDM(\hat{\boldsymbol{s}}_{i+1}) = \boldsymbol{s}_{i+1}^{pred} \tag{57}$$

Alternatively, the input for the D.D.M. can be altered such that the D.D.M. used both $\boldsymbol{s}_i$ and $PBM(\boldsymbol{s}_i)$, that is:

$$DDM(\boldsymbol{s}_i, \hat{\boldsymbol{s}}_{i+1}) = \boldsymbol{s}_{i+1}^{pred} \tag{58}$$

This model closely resembles the pure D.D.M. from section 4.3.2, though by introducing $PBM(\boldsymbol{s}_i)$ as input, instead of only using predictions from the D.D.M. as inputs, the D.D.M. is more bounded by the underlying physics of the problem (Rai, Sahu, 2020).

Residual model: Another hybrid model is the R.M. It is one of the simplest hybrid models and most commonly used ones (Daw et al., 2017). This model uses the P.B.M. in the same manner as the H.P.D.M., though the D.D.M.'s prediction differs. Here, the D.D.M. uses the same input as the P.B.M. to predict a residual, $\boldsymbol{res}_{i+1}^{pred}$, which is then added to the P.B.M. output to produce the final prediction, that is:

$$PBM(\boldsymbol{s}_i) + DDM(\boldsymbol{s}_i) = \hat{\boldsymbol{s}}_{i+1} + \boldsymbol{res}_{i+1}^{pred} = \boldsymbol{s}_{i+1}^{pred} \tag{59}$$

This approach is viewed as a simpler task than improving $\hat{\boldsymbol{s}}_{i+1}$ as in the H.P.D.M. above. In R.Ms., the D.D.M. is tasked with learning the uncaptured physics of the P.B.M. and the systematic errors of the numerical method itself. There is, however, no direct synergy between the P.B.M. and the D.D.M., and even though the D.D.M. is entrusted in capturing systematic biases of the P.B.M., the D.D.M. is not given any direct information regarding the P.B.M. This can be solved by the following hybrid model.

Hybrid-Physics-Data-Residual model: The innovative hybrid model, termed hybrid-physics-data-residual model (H.P.D.R.M., or sometimes H.P.D.-Res.) (Daw et al., 2017), is the combination of the previously mentioned hybrid models, H.P.D.M. and R.M. This model uses the same structure as the R.M., only the D.D.M. uses the same inputs as the H.P.D.M., that is:

$$DDM(\boldsymbol{s}_i, PBM(\boldsymbol{s}_i)\ ) = \boldsymbol{res}_{i+1}^{pred} \tag{60}$$

as shown in figure 23. This gives the final hybrid model:

$$PBM(\boldsymbol{s}_i) + DDM(\boldsymbol{s}_i, PBM(\boldsymbol{s}_i)\ ) = \hat{\boldsymbol{s}}_{i+1} + \boldsymbol{res}_{i+1}^{pred} = \boldsymbol{s}_{i+1}^{pred} \tag{61}$$

The way this model is constructed, it inherits the advantages from both its parent models. As H.P.D.R.M. uses the same inputs as H.P.D.M. for its D.D.M., i.e., ($\boldsymbol{s}_i$, $\hat{\boldsymbol{s}}_{i+1}$), it simplifies the process of learning residuals; the D.D.M. now receives information regarding the results of the P.B.M. (Daw et al., 2017). Additionally, by using these inputs, the model is also restricted by the underlying physics, as in the case of the H.P.D.M. The D.D.M. of this model is also tasked with the simpler

process of predicting residuals suchlike the R.M, in contrast with the H.P.D.M., whose task is to predict the optimal approximation $\boldsymbol{s}_{i+1}^{pred}$ directly.

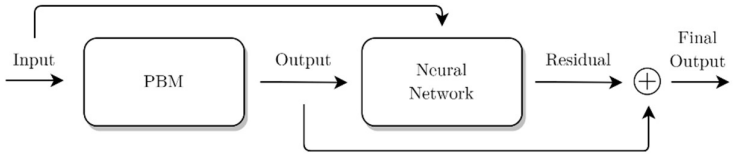

Figure 21: Pipeline for the H.P.D.R. model.

When it comes to training/validating the model, the loss function of the D.D.M. part of the hybrid model must be defined. By eq. (61), it is clear that $PBM(\boldsymbol{s}_i)$ must the calculated before the D.D.M. can predict the residuals. At the same time, the solution is defined as:

$$\boldsymbol{s}_{i+1} - PBM(\boldsymbol{s}_i) = \boldsymbol{res}_{i+1}^{sol} \tag{62}$$

The D.D.M. is needed to minimize the loss function, which is now defined as:

$$f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\left\|(\hat{\boldsymbol{s}}_{j,i+1} + \boldsymbol{res}_{j,i+1}^{pred}) - \boldsymbol{s}_{j,i+1}^{sol}\right\|^2 \rightarrow min$$

$$\Leftrightarrow f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\left\|\boldsymbol{res}_{j,i+1}^{pred} - (\boldsymbol{s}_{j,i+1}^{sol} - \hat{\boldsymbol{s}}_{j,i+1})\right\|^2 \rightarrow min$$

$$\Leftrightarrow f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\left\|\boldsymbol{res}_{j,i+1}^{pred} - \boldsymbol{res}_{j,i+1}^{sol}\right\|^2 \rightarrow min \tag{63}$$

5.2 Hybrid-Physics-Data-Residual model for the n-body problem

In this section, the H.P.D.R.M. for modeling the n-body problem will be presented. The entire solar system is considered but for simplification purposes, residuals for only one planet is predicted. When constructing the H.P.D.R.M., the pipeline from figure 23 is used. This is how the model will look from a testing/application point-of-view. For the case of training/validating the model, the approach is somewhat different. This thesis tackles two approaches when setting up the training phase: 1) single-step training method, and 2) multi-step training method.

Approach 1) considers a single timestep ($n = 1$) in the training phase, that is: the N.N. takes in one input and returns one output, for the loss function eq. (63) to thereafter be calculated and minimized. For this case, a pre-constructed dataset with approximated solutions from the P.B.M., $\hat{\boldsymbol{s}}_{i+1}$ (eq. (56)), is needed. The D.D.M. will take in a collected input from both the NASA dataset and the P.B.M. dataset, predict the residual by eq. (60), minimize eq. (63), and then repeat this process for every datapoint in the datasets. This is in line with the general way the D.D.Ms. throughout this thesis have been trained and is the easiest to implement and fastest approach of the two.

Approach 2) aims to stabilize the D.D.M. by minimizing the loss function for a more global version of the truncation error of the P.B.M., instead of evaluating the local truncation error as in approach 1. This approach takes into account n timesteps in the training phase, which implies: the N.N. goes through multiple timesteps before minimizing the loss function. Here, the P.B.M. and the D.D.M. work in tandem right from the start of the training phase. The pipeline from figure 23 is used for $\boldsymbol{n}$ consecutive timesteps, hence the multi-step. Eq. (61) is used for the first timestep, being based on the initial condition, and eq. (64) for the consecutive $n-1$ timesteps, being based on predictions and approximations from the D.D.M. and P.B.M., respectively:

$$PBM\big(\boldsymbol{s}_{i+1}^{pred}\big) + DDM\big(\boldsymbol{s}_{i+1}^{pred}, PBM\big(\boldsymbol{s}_{i+1}^{pred}\big)\ \big) = \hat{\boldsymbol{s}}_{i+2} + \boldsymbol{res}_{i+2}^{pred} = \boldsymbol{s}_{i+2}^{pred} \tag{64}$$

The loss function measures the difference between the predictions and solutions for all considered timesteps, which yields two $n \times size(\boldsymbol{s})$ tensors of predicted residuals and solution residuals, respectively, to be evaluated in the loss function, which now is defined as:

$$f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\sum_{i=1}^{n}\left\|\boldsymbol{res}_{j,i+1}^{pred} - \boldsymbol{res}_{j,i+1}^{sol}\right\|^2 \rightarrow min \tag{65}$$

In short, this method evaluates the error of the predictions the N.N. makes over n timesteps, and is in line with the iterative process of the test-phase/real-world application of the H.P.D.R.M. This method, though the slowest and somewhat harder to implement, may yield a more stable H.P.D.R.M. which has a higher tolerance for poor predictions.

Another change from the previous P.B.M. and D.D.M. is the size of the dataset used in training the H.P.D.R.M. The dataset considered is a subset of the data used for the previous models and spans approximately merely the first 900 days of the

total dataset. The assumption is made that this dataset is sufficient for training the D.D.M. sub-model of the H.P.D.R.M. The reason for this reduced dataset assumption stems from another assumption: a correction made to the state vector for a single planet, results in overall improved results for the entire system. This gives the possibility to only looking at residuals for a single planet, which simplifies the problem. The planet whose state vector will be corrected, is the innermost planet of the Solar System. This is motivated by the fact that this planet yields the overall largest error in the results from the P.B.M., as shown in table 2. This planet also has the smallest orbital period, thus a dataset of 900 days represents multiple periods for the given planet. This is expected to be sufficient for the D.D.M. to predict satisfying residuals. 80% of the dataset is used for training, thus the D.D.M. have O.O.D. for testing which contains multiple orbital periods.

5.2.1 Position-based Hybrid-Physics-Data-Residual model

When creating D.D.Ms. for predicting planetary motion throughout this thesis, only positions were considered so far. It was viewed as a simplification for the overall model to omit velocities, and it was hypothesized that the D.D.M. could perform well even when excluding this vital information; to compensate the lack of velocities as inputs, it should be however stressed, that positions for two consecutive timesteps were used instead. (It should be noted that two consecutive timesteps are being used in the H.P.D.R.M. as well, though for another purpose, as stated in section 5.1). As section 4.3.2 showed, this hypothesis turned out true. A natural question then arises for the H.P.D.R.M. in response to these results: Can one make similar simplification to the H.P.D.R.M. (i.e., omitting velocities) as for the pure D.D.M., and still obtain valid results? When looking at the P.B.M. part of the H.P.D.R.M., as eq. (28) shows, initial velocities cannot be excluded from this sub-model. The next part to consider in the H.P.D.R.M. is then the D.D.M., though before looking at this sub-model, an interesting observation must be considered:

Change in the symplectic flow: A key property mentioned throughout chapter 2 and 3 is the symplectic nature of both the Hamiltonian system for planetary motion and the numerical integrator considered. As presented in the mentioned chapters, the flow of the solution for the system is symplectic, and therefore a symplectic numerical integrator is used to mimic the symplectic flow. This symplectic flow results in the area outlined from the velocity/momentum and position component of the solution in phase-space being conserved. An interesting idea is what happens to the flow as a correction is made to only the position component in the approximated solution from the P.B.M. In the following result, a simple plot shows what happens when this type of correction in the position

component is made to the innermost planet. Eq. (62) is used to find the residuals for a single step with the P.B.M. from datapoints in the NASA dataset. The P.B.M. is again run for a single initial condition from the NASA dataset, and the new residual dataset is used as the correction term in the position for the innermost planet after every consecutive iteration.

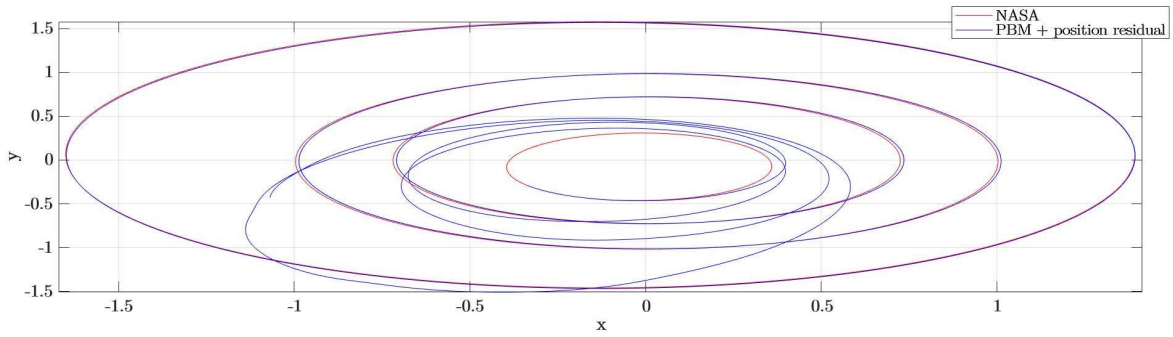

Figure 22: 2d plot of PBM results + position residual vs. NASA data.

As figure 24 shows, the approximated solution from the P.B.M. with the position correction drifts from the exact solution within the first orbital period, and the drift becomes larger after every iteration. This is likely due to the fact that there is no corresponding correction in the velocity component of the approximated solution, and the resulting flow of the solution is no longer symplectic. The orbits will most likely continue to enlarge and at some point, the planets will diverge.

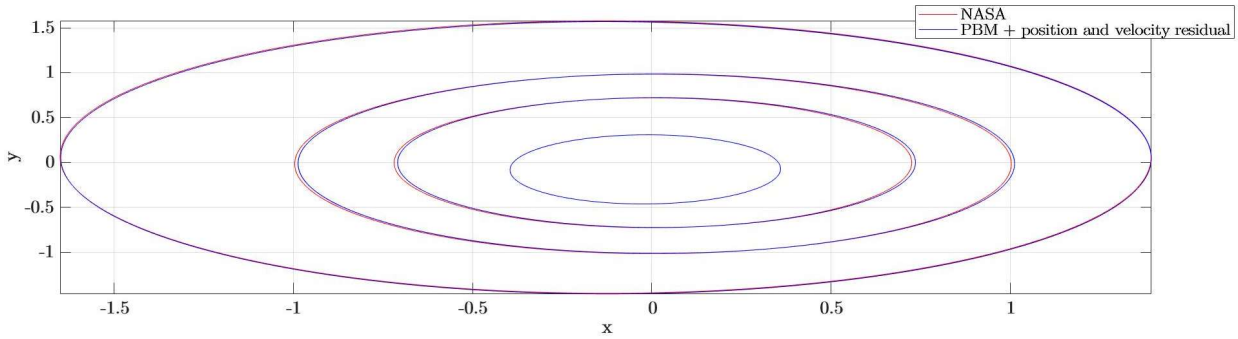

Figure 25: 2d plot of PBM results + position and velocity residual vs. NASA data.

As figure 25 shows, if a corresponding correction is made to the velocity, as well as to the position, the approximated solution becomes the exact, and the flow keeps its symplectic nature.

It is now clear that the P.B.M. not only needs a correction in positions, but also a corresponding correction in velocities. This results in the D.D.M. needing to predict residuals for both positions and velocities, or the D.D.M. needing to predict a correction term for only the position that compensates for the lack of correction

in velocity. The latter is quite interesting, as it has been conceptualized that D.D.Ms. can arrive at predictions not bounded by known mathematics and physics, as stated throughout this thesis. It is thereby not outside the scope of reason to try to construct such a D.D.M. For this simplified model, eq. (61) becomes:

$$PBM(\boldsymbol{s}_i) + DDM\big(\boldsymbol{x}_i, PBM(\boldsymbol{x}_i)\big) = \hat{\boldsymbol{s}}_{i+1} + (\boldsymbol{res}_{i+1}^{pred}, 0) = \boldsymbol{s}_{i+1}^{pred} \qquad (66)$$

where $\boldsymbol{res}$ merely addresses corrections in the positions, as there is not correction in the velocity component. The associated loss function from eq. (63) is now given as:

$$f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\left\|(\hat{\boldsymbol{x}}_{j,i+1} + \boldsymbol{res}_{j,i+1}^{pred}) - \boldsymbol{x}_{j,i+1}^{sol}\right\|^2 \rightarrow min$$

$$\Leftrightarrow f_{loss} = \frac{1}{N}\sum_{j=1}^{N}\left\|\boldsymbol{res}_{j,i+1}^{pred} - \boldsymbol{res}_{j,i+1}^{sol}\right\|^2 \rightarrow min \qquad (67)$$

Results by single-step training phase: The following results show the predictions from the simplified H.P.D.R.M. described above, using the single-step training method presented in the beginning of section 5.2. The predictions are plotted against the exact solution from the NASA data. For visualization purposes, plots for the outer four planets are omitted.

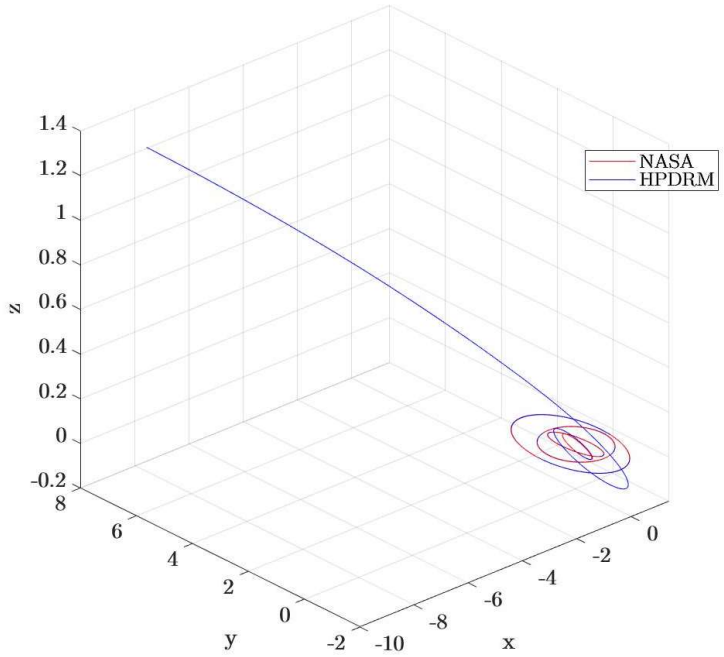

Figure 26: 3d plot of H.P.D.R.M. results with position residual and single-step training vs. NASA data.

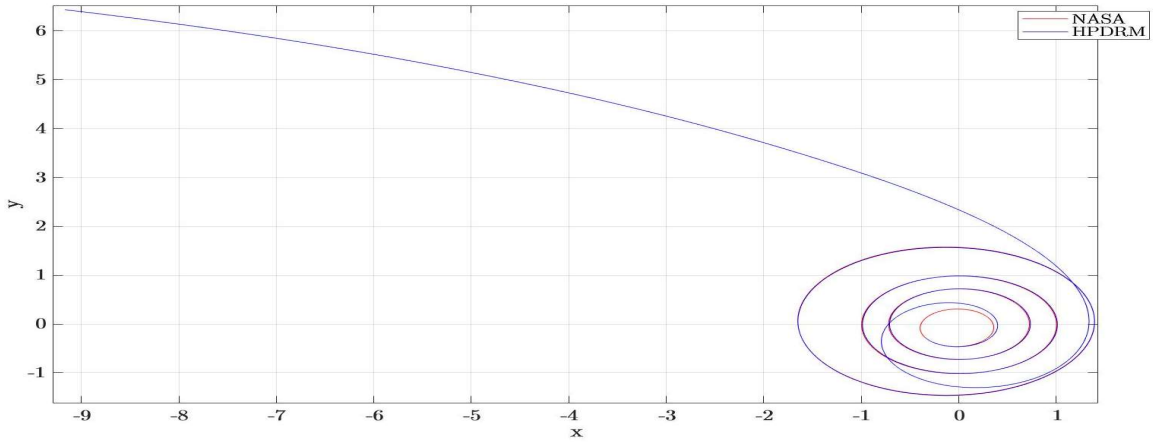

Figure 27: 2d plot of H.P.D.R.M. results with position residual and single-step training vs. NASA data.

As figure 26 and 27 shows, the innermost planet diverges relatively quickly; after one-fourth orbital period the planet has already left its initial orbit, and after approximately another period, it diverges rapidly, exiting the Solar System. The reason behind these poor results may be the exclusion of the velocity residual. Another reason may be that the error stems from the D.D.M. sub-models' unstable predictions. As the D.D.M. is trained only using residuals from a single step by the P.B.M., if the D.D.M. makes a bad prediction which offsets the planets orbit to such a degree that the D.D.M. is not generalized to predict a new corresponding residual, the D.D.M. may continue to predict poor residuals. In other words, the D.D.M. may not be stable for accumulative errors made by both the P.B.M. and the D.D.M. itself. A possible solution to this instability is to introduce the multi-step training method presented at the start of section 5.2.

Results by multi-step training method: The next results show how the simplified H.D.P.R.M. performs by using the multi-step training method, with steps $n = 4$.

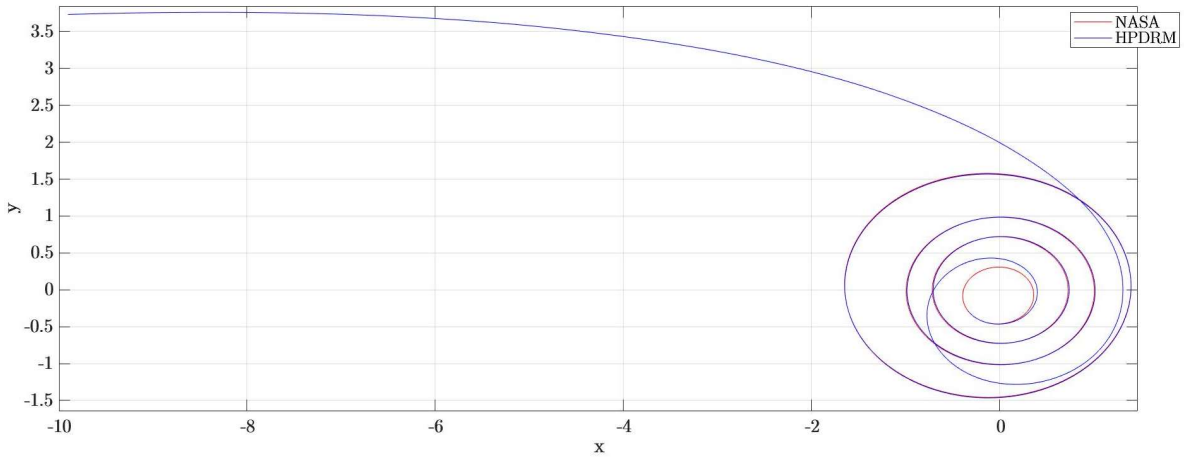

Figure 28: 2d plot of H.P.D.R.M. results with position residual and single-step training vs. NASA data.

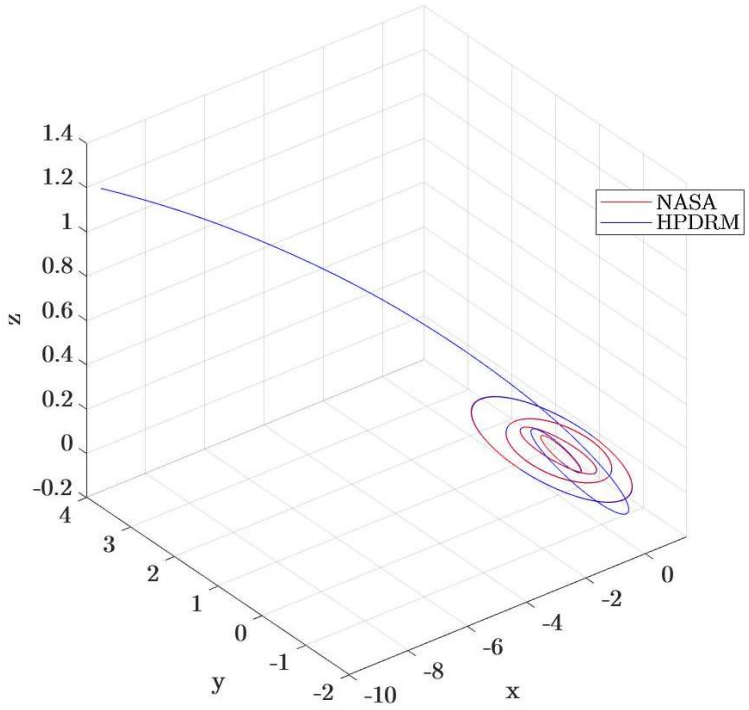

Figure 29: 3d plot of H.P.D.R.M. results with position residual and multi-step training vs. NASA data.

As figure 28 and 29 reveals, the position of the innermost planet still diverges, and there has not been any notable changes from the previous results. As both single-step and multi-step training yields similar results, this disproves the hypothesis of excluding velocity residuals in the H.P.D.R.M.

5.2.2 Position and velocity-based Hybrid-Physics-Data-Residual model

As section 5.2.1 showed, the omission of velocity residuals in the H.P.D.R.M. results in diverging state-vectors. Therefore, the simplified H.P.D.R.M. in eq. (66) is discarded, and the full H.P.D.R.M. from eq. (61), with loss function eq. (63), is reinstated. An important feature in this model is the use of a pooling layer in the N.N. architecture, as presented in section 4.3.2. In this model however, the smaller local N.Ns. and the pooling layer are used to separate the position $\boldsymbol{x}$ and velocity $\boldsymbol{y}$ component in the state vector $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{y})$. This is motivated by the difference in magnitude and units of the two variables. As previously, only residuals for one planet are predicted and added to the P.B.M. approximation. Also, only the four innermost planets are plotted, as the results from the outermost planets do not offer any vital information. It should be noted that all the planets are still included in the calculations by the model.

Results by single-step training method: As there was no significant differences in the results from the two training methods presented in section 5.2.1, the single-step training method is again used. This is, as mentioned, a simpler modeling approach and follows the same approach as all the previous D.D.Ms. throughout this thesis. If the D.D.M. is trained accurately, it is well with reason to expect the D.D.M. to make valid prediction for the residuals.

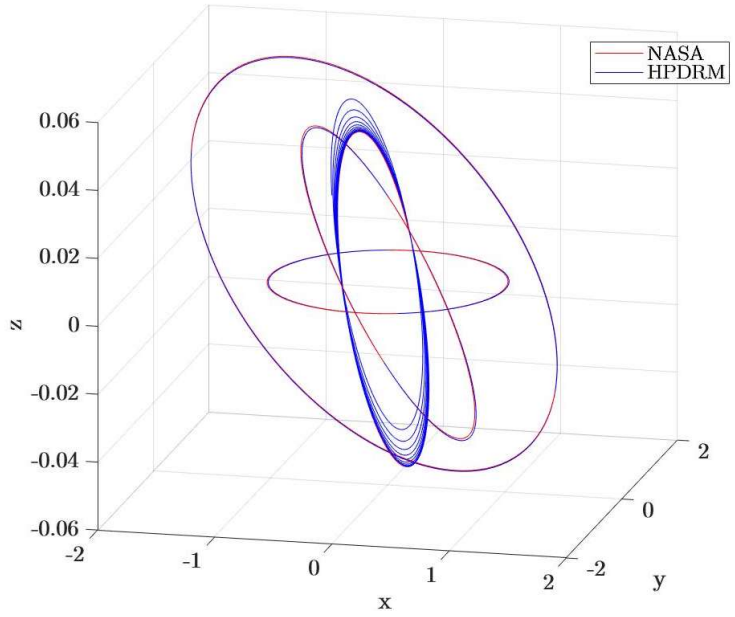

Figure 30: 3d plot of H.P.D.R.M. results with position and velocity residual and single-step training vs. NASA data.

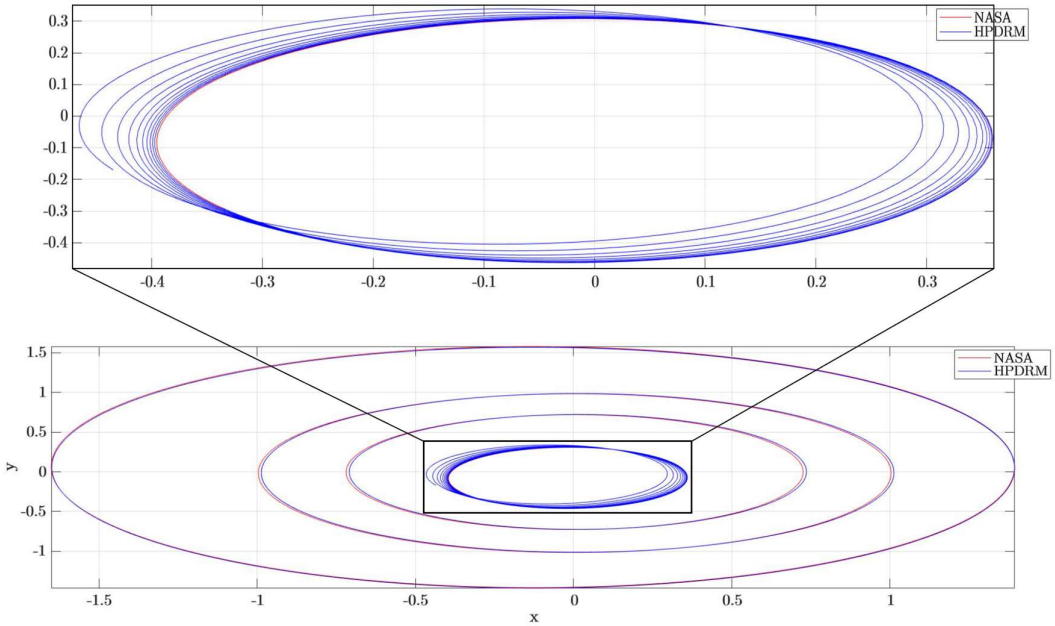

Figure 31: 2d plot of H.P.D.R.M. results with position and velocity residual and single-step training vs. NASA data.

As figure 30 and 31 shows, the H.P.D.R.M. now manages to predict what looks to be a stable orbit of the innermost planet, though these orbits have an increasing drift towards the left.

Results by multi-step training method: Even though the multi-step training method showed no substantial improvement from the single-step training method in section 5.2.1, it is still believed that the former can improve stability for the D.D.M. sub-model of the H.P.D.R.M. Thus, a final hybrid model is tested; H.P.D.R.M. with state vector residuals and multi-step training, with steps of $n = 4$.

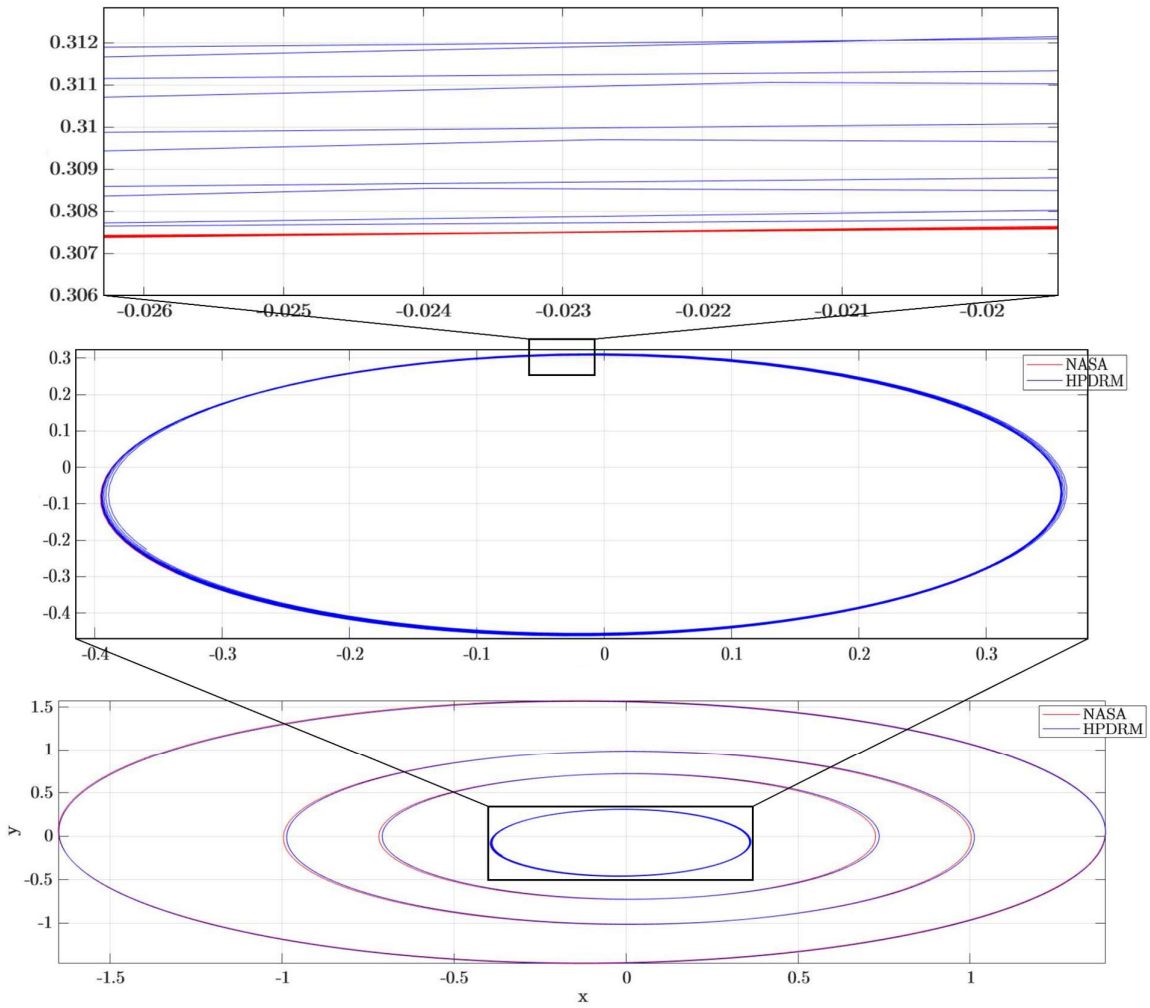

Figure 32: 2d plot of H.P.D.R.M. results with position and velocity residual and multi-step training vs. NASA data.

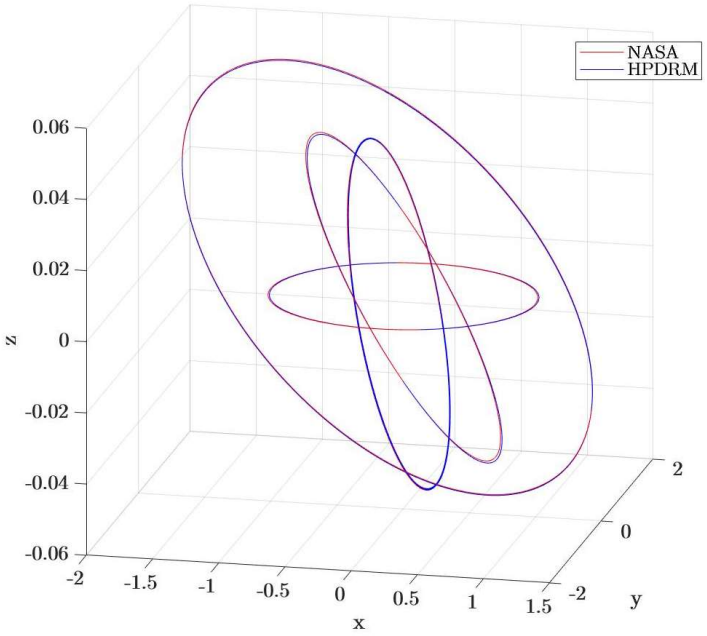

Figure 33: 3d plot of H.P.D.R.M. results with position and velocity residual and multi-step training vs. NASA data.

As figure 32 and 33 shows, the H.P.D.R.M. now manages to predict stable orbits for the innermost planet. Though there seems to be a small drift in the orbit toward the right, this is significantly small in comparison to the previous single-step training method, and the orbit may seem stationary at first glance. The effect of increasing n has not yet been tested, though it is hypothesized that training and optimizing for increased number of steps will result in less drift in the orbit, and thus more accurate results. As n is increased, it would be interesting to investigate the effect of adaptive optimization techniques. As the multi-step approach predicts values based on previous prediction before optimizing, the first sequences of predictions will have large deviations from the solution. The first optimization sequences in the training process can therefore be demanding for the optimization algorithm and make it hard for the N.N. to learn. As n increases, these deviations becomes even larger. To counter this, an adaptive optimization technique could include an increasing number for timesteps in the optimization as the networks learn more from the data. This could result in the model being stable over larger time-intervals and being able to learn the path of the planets, in contrast to only learning the direction for a single timestep.

As the figures above show the results of the overall model, it is interesting to look at the performance of each sub-model. The first sub-model to look at is the D.D.M. Figure 34 and 35 show the residual predictions of the D.D.M. against the exact residual. At the start, the D.D.M. seems to make excellent predictions, though

over time the residuals moves slower than the real solution. This fits well with the apparent drift in the planet's orbit. There are also small details in the solution graph (i.e., small spikes) that the D.D.M. did not manage to capture. Overall, the D.D.M. managed to learn and predict the periodic shape of the solution well for the given time-interval.

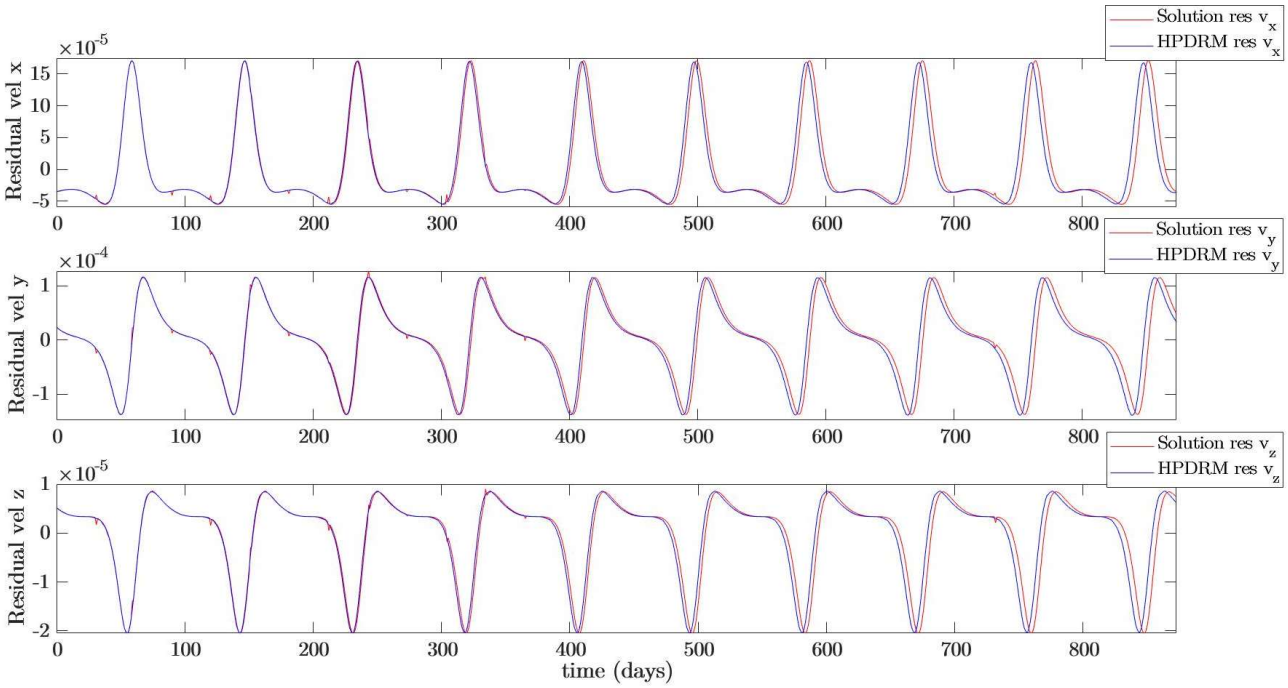

Figure 34: Position residuals predictions by HPDRM vs solution residual

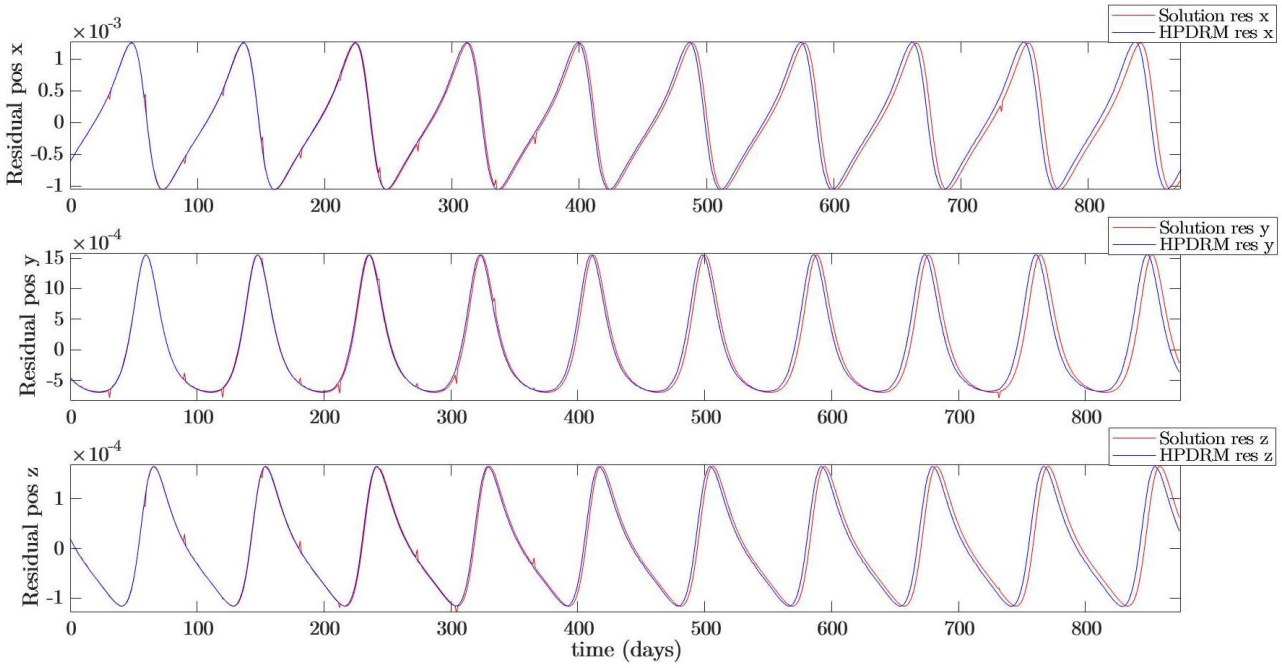

Figure 35: Velocity residuals predictions by HPDRM vs. solution residual

Though residuals are only added to the innermost planet, the average errors of the results from all the planets are included in table 11, which are calculated in the same manner as the other errors presented in this thesis, by eq. (34) and eq. (35). This is interesting to look at, as there may be changes in the results from the P.B.M. sub-model for the other planets. This is since each planet influences the evolution of the others, thus, a change in one planet's state vector corresponds to a change in the other planets, even ever so slightly. Table 11 shows the same trend as the results from the P.B.M. in chapter 3; the errors decrease with the size of the orbital period. The errors are also overall small. The results from the H.P.D.R.M. are presented here, though a full comparison between these results from each of the final models throughout this thesis will be carried out in the upcoming chapter 6.

Table 11: Average absolute and relative error of position and velocity by HPDRM for each planet

Celestial object	Avg. absolute error (position)	Avg. absolute error (velocity)	Avg. relative error (position)	Avg. relative error (velocity)
Mercury	$7.0150 \cdot 10^{-5}$	$5.3559 \cdot 10^{-6}$	$5.9476 \cdot 10^{-6}$	$6.5482 \cdot 10^{-6}$
Venus	$2.6617 \cdot 10^{-5}$	$3.9140 \cdot 10^{-7}$	$1.2445 \cdot 10^{-6}$	$6.5465 \cdot 10^{-7}$
Earth	$3.0670 \cdot 10^{-5}$	$3.3358 \cdot 10^{-7}$	$1.0379 \cdot 10^{-6}$	$6.5553 \cdot 10^{-7}$
Mars	$2.7683 \cdot 10^{-5}$	$1.7181 \cdot 10^{-7}$	$6.1360 \cdot 10^{-7}$	$4.1526 \cdot 10^{-7}$
Jupiter	$2.9042 \cdot 10^{-6}$	$5.1467 \cdot 10^{-9}$	$1.9485 \cdot 10^{-8}$	$2.2365 \cdot 10^{-8}$
Saturn	$2.3897 \cdot 10^{-6}$	$5.1727 \cdot 10^{-9}$	$8.9512 \cdot 10^{-9}$	$2.9770 \cdot 10^{-8}$
Uranus	$2.3003 \cdot 10^{-6}$	$5.8345 \cdot 10^{-9}$	$4.2512 \cdot 10^{-9}$	$4.8038 \cdot 10^{-8}$
Neptune	$2.1591 \cdot 10^{-6}$	$5.7770 \cdot 10^{-9}$	$2.4087 \cdot 10^{-9}$	$6.2842 \cdot 10^{-8}$

6 Comparison of the Hybrid-Physics-Data-Residual model, Physics-based model, and Data-driven model

In this following chapter, the errors of the results from the final H.P.D.R.M., P.B.M., and D.D.M. will be compared. Firstly, the average errors as presented throughout this thesis will be collected in a joint table and evaluated with respect to each other model. Secondly, errors of the H.P.D.R.M. and P.B.M. will be compared for the same time-interval as the H.P.D.R.M. was tested for; to recall the latter has been trained for a significantly shorter time interval. The reason for evaluating the methods on this restricted interval will be clarified in section 6.2.

6.1 Results of average errors

Throughout this thesis, average absolute and relative errors for the results by a single timestep have been calculated using eq. (34) and eq. (35). The errors are, as mentioned in section 3.3, calculated accordingly to make it possible to compare methods using different step-sizes and time-intervals. Only errors in positions are compared in this section, as the D.D.M. does not directly predict velocities. For the P.B.M., the entire time-interval for the dataset from NASA was considered when approximating solutions to the system. As for the D.D.M., only half of the dataset was considered when predicting the planetary motions. This was a consequence of the need for training data and O.O.D. when testing, as a substantial part of the dataset was needed for training. The D.D.M. errors presented below come from the final D.D.M. with pooling from section 4.3.2. The H.P.D.R.M. was trained and tested only for a small subset of the NASA dataset. This was due to time constraints, and the assumption that the size of the subset was sufficient for training and testing the H.P.D.R.M.

Table 12 shows the average errors mentioned above for each planets by the different models. The main differences between the models are as follows:

- For the four innermost planets, the P.B.M. and D.D.M. outperform the H.P.D.R.M., with the P.B.M. performing somewhat better than the D.D.M.
- For the outermost planets, the P.B.M. and H.P.D.R.M. outperform the D.D.M., with the P.B.M. performing the best of the former.

Table 12: Average absolute and relative errors of HPDRM vs. PBM vs. DDM for each planet.

	H.P.D.R.M		P.B.M.		D.D.M.	
Celestial object	Avg. absolute error (position)	Avg. relative error (position)	Avg. absolute error (position)	Avg. relative error (position)	Avg. absolute error (position)	Avg. relative error (position)
Mercury	$7.0150 \cdot 10^{-5}$	$5.9476 \cdot 10^{-6}$	$6.8067 \cdot 10^{-6}$	$5.9479 \cdot 10^{-8}$	$1.0290 \cdot 10^{-6}$	$1.0048 \cdot 10^{-7}$
Venus	$2.6617 \cdot 10^{-5}$	$1.2445 \cdot 10^{-6}$	$6.9013 \cdot 10^{-6}$	$3.3281 \cdot 10^{-8}$	$2.8340 \cdot 10^{-7}$	$1.5273 \cdot 10^{-8}$
Earth	$3.0670 \cdot 10^{-5}$	$1.0379 \cdot 10^{-6}$	$5.1918 \cdot 10^{-6}$	$1.8107 \cdot 10^{-8}$	$2.7049 \cdot 10^{-6}$	$1.0543 \cdot 10^{-7}$
Mars	$2.7683 \cdot 10^{-5}$	$6.1360 \cdot 10^{-7}$	$7.1478 \cdot 10^{-6}$	$1.6256 \cdot 10^{-8}$	$8.4157 \cdot 10^{-7}$	$2.1392 \cdot 10^{-8}$
Jupiter	$2.9042 \cdot 10^{-6}$	$1.9485 \cdot 10^{-8}$	$1.0662 \cdot 10^{-5}$	$7.1360 \cdot 10^{-9}$	$6.5533 \cdot 10^{-6}$	$4.8936 \cdot 10^{-8}$
Saturn	$2.3897 \cdot 10^{-6}$	$8.9512 \cdot 10^{-9}$	$1.1771 \cdot 10^{-5}$	$4.2948 \cdot 10^{-9}$	$1.2111 \cdot 10^{-5}$	$4.9452 \cdot 10^{-8}$
Uranus	$2.3003 \cdot 10^{-6}$	$4.2512 \cdot 10^{-9}$	$1.1812 \cdot 10^{-5}$	$2.1405 \cdot 10^{-9}$	$5.5237 \cdot 10^{-4}$	$1.1097 \cdot 10^{-6}$
Neptune	$2.1591 \cdot 10^{-6}$	$2.4087 \cdot 10^{-9}$	$1.4212 \cdot 10^{-6}$	$1.6469 \cdot 10^{-10}$	$4.1414 \cdot 10^{-4}$	$5.3506 \cdot 10^{-7}$

Table 13 shows the average errors in the Hamiltonian for the H.P.D.R.M. and P.B.M. by eq. (38) and eq. (39). The D.D.M. is not included here as it does not predict velocities, and the Hamiltonian thus cannot be directly calculated without approximating velocities from changes in position. As the table shows, the P.B.M. has the lowest error in the Hamiltonian.

Table 13: Average absolute and relative error in Hamiltonian for HPDRM vs. PBM.

	H.P.D.R.M.		P.B.M.	
Hamiltonian	Avg. absolute error	Avg. relative error	Avg. absolute error	Avg. relative error
	$1.1290 \cdot 10^{-12}$	$8.1015 \cdot 10^{-9}$	$2.4024 \cdot 10^{-13}$	$1.8411 \cdot 10^{-10}$

6.1 Hybrid-Physics-Data-Residual model vs. Physics-based model on equal time-interval

In the section above, average errors for single time-steps was considered. An interesting observation is that these averages do not justify the H.P.D.R.M.'s performances. If one investigates the general absolute and relative error by eq. (32) and eq. (33), the comparison between the models yields a different outcome. For this, the H.P.D.R.M. and the P.B.M., which both use a step-size $\Delta t = 1$ day, is compared for the same time interval of 900 days. As the D.D.M. uses both different step-size, time-interval, and do not predict velocities, the D.D.M. is excluded from this comparison.

Table 14: Absolute and relative errors in position by HPDRM vs. PBM for each planet.

	H.P.D.R.M		P.B.M.	
Celestial object	Absolute error (position)	Relative error (position)	Absolute error (position)	Relative error (position)
Mercury	1.8126	$1.5368 \cdot 10^{-1}$	4.2746	$3.6241 \cdot 10^{-1}$
Venus	$6.8773 \cdot 10^{-1}$	$3.2155 \cdot 10^{-2}$	$6.8773 \cdot 10^{-1}$	$3.2155 \cdot 10^{-2}$
Earth	$7.9247 \cdot 10^{-1}$	$2.6817 \cdot 10^{-2}$	$7.9247 \cdot 10^{-1}$	$2.6817 \cdot 10^{-2}$
Mars	$7.1529 \cdot 10^{-1}$	$1.5854 \cdot 10^{-2}$	$7.1529 \cdot 10^{-1}$	$1.5854 \cdot 10^{-2}$
Jupiter	$7.5039 \cdot 10^{-2}$	$5.0345 \cdot 10^{-4}$	$7.5039 \cdot 10^{-2}$	$5.0345 \cdot 10^{-4}$
Saturn	$6.1747 \cdot 10^{-2}$	$2.3129 \cdot 10^{-4}$	$6.1747 \cdot 10^{-2}$	$2.3129 \cdot 10^{-4}$
Uranus	$5.9437 \cdot 10^{-2}$	$1.0984 \cdot 10^{-4}$	$5.9437 \cdot 10^{-2}$	$1.0984 \cdot 10^{-4}$
Neptune	$5.5788 \cdot 10^{-2}$	$6.2238 \cdot 10^{-5}$	$5.5788 \cdot 10^{-2}$	$6.2238 \cdot 10^{-5}$

As table 14 and 15 show, the error in both position and velocity for the innermost planet is lower for the H.P.D.R.M. than for the P.B.M. As there is no correction for the other planets, one would initially expect the errors by the H.P.D.R.M. and P.B.M. to be identical for these planets. There are though some insignificant differences between the errors for the other planets, ranging from differences in the 6^{th} to 10^{th} significant figure. These small differences are expected as each planet influences the evolution of the others, though it is not directly clear if the correction for the innermost planet will result in an improvement for the other planets. The important observation is that the trajectories of the other planets remain stable.

Table 15: Absolute and relative errors in velocity by HPDRM vs. PBM for each planet.

	H.P.D.R.M		P.B.M.	
Celestial object	Absolute error (velocity)	Relative error (velocity)	Absolute error (velocity)	Relative error (velocity)
Mercury	$1.3839 \cdot 10^{-1}$	$1.6919 \cdot 10^{-1}$	$3.0144 \cdot 10^{-1}$	$3.6854 \cdot 10^{-1}$
Venus	$1.0113 \cdot 10^{-2}$	$1.6915 \cdot 10^{-2}$	$1.0113 \cdot 10^{-2}$	$1.6915 \cdot 10^{-2}$
Earth	$8.6193 \cdot 10^{-3}$	$1.6938 \cdot 10^{-2}$	$8.6193 \cdot 10^{-3}$	$1.6938 \cdot 10^{-2}$
Mars	$4.4393 \cdot 10^{-3}$	$1.0730 \cdot 10^{-2}$	$4.4393 \cdot 10^{-3}$	$1.0730 \cdot 10^{-2}$
Jupiter	$1.3298 \cdot 10^{-4}$	$5.7787 \cdot 10^{-4}$	$1.3298 \cdot 10^{-4}$	$5.7787 \cdot 10^{-2}$
Saturn	$1.3365 \cdot 10^{-4}$	$7.6922 \cdot 10^{-4}$	$1.3365 \cdot 10^{-4}$	$7.6922 \cdot 10^{-4}$
Uranus	$1.5075 \cdot 10^{-4}$	$1.2412 \cdot 10^{-3}$	$1.5075 \cdot 10^{-4}$	$1.2412 \cdot 10^{-3}$
Neptune	$1.4927 \cdot 10^{-4}$	$1.6237 \cdot 10^{-3}$	$1.4927 \cdot 10^{-4}$	$1.6237 \cdot 10^{-3}$

To evaluate any change in the total system, the Hamiltonian can be analyzed, as it is a metric for the behavior of the overall system. Figure 36 and table 16 display both the evolution and the error made in the Hamiltonian. Based on these, it is concluded that the error in the overall system is lower for the H.P.D.R.M. in comparison to the P.B.M.

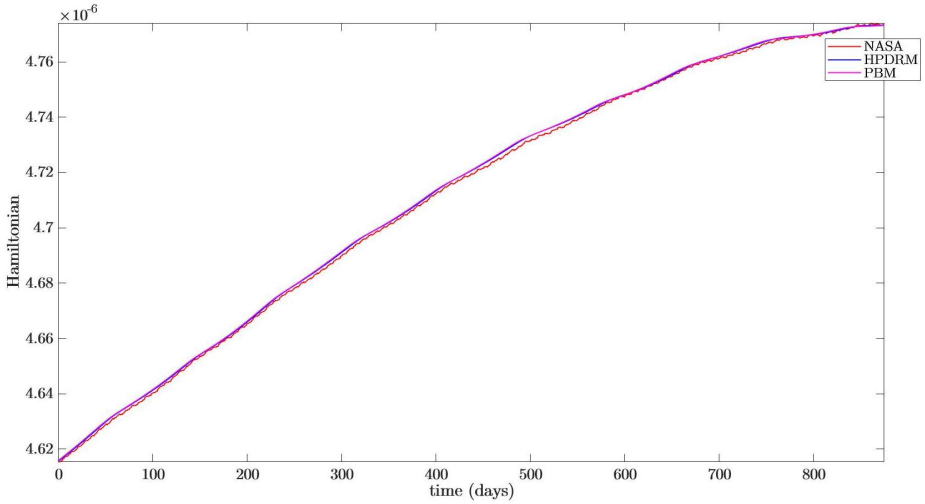

Figure 36: Hamiltonian by HPDRM vs. PBM vs. solution by NASA data.

Table 16: Absolute and relative error in Hamiltonian by HPDRM vs. PBM.

	H.P.D.R.M.		P.B.M.	
	Absolute error	Relative error	Absolute error	Relative error
Hamiltonian	$2.4292 \cdot 10^{-8}$	$1.7430 \cdot 10^{-4}$	$2.9680 \cdot 10^{-8}$	$2.1310 \cdot 10^{-4}$

Figure 37 shows the residuals predicted by the H.P.D.R.M. against the residuals of the P.B.M. from chapter 4. The position-components and velocities are not included, as these show similar results as figure 37. As the figure shows, the residual of the H.P.D.R.M. is significantly smaller than that of the P.B.M., further supporting that the H.P.D.R.M. preforms better than the P.B.M. on an equal time-interval.

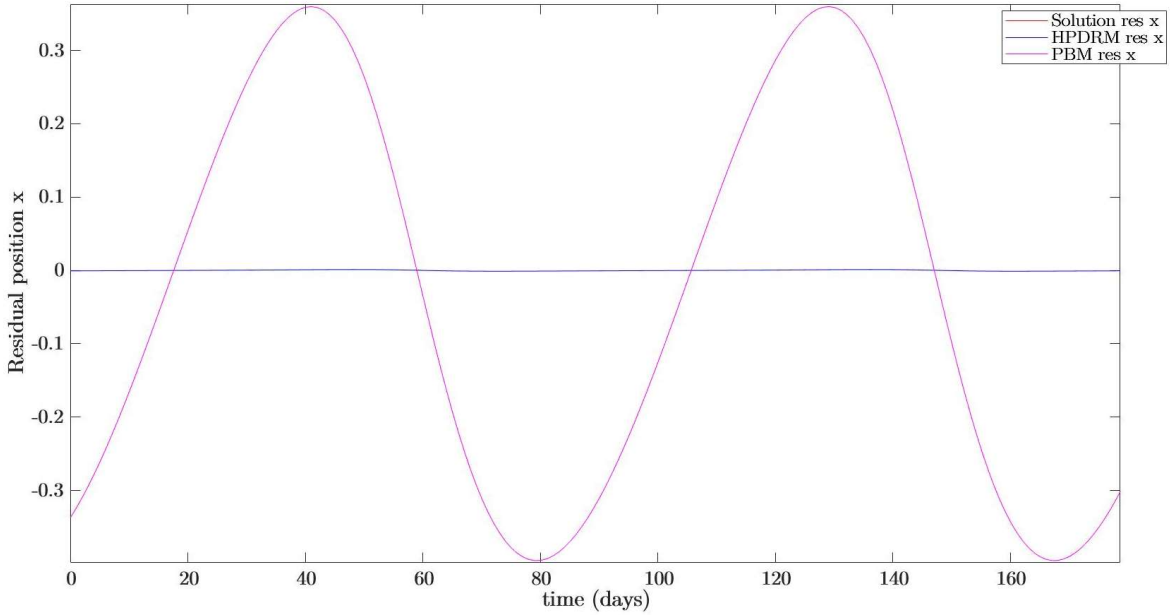

Figure 37: Position residual in x-direction by HPDRM vs. PBM vs. solution residual.

So, are the results from the average errors in section 6.1 and the errors presented in this section contradicting? Even though table 12 and 13 nicely summarize the results throughout this thesis, due to the results from this section, it is identified that the comparisons in table 12 and 13 as are inconclusive. More representative, the H.P.D.R.M. performs better than the P.B.M. on an equal time-interval. It would though be interesting to investigate how the H.P.D.R.M. would perform on the full time-interval of 225 years – possibly after training with a larger dataset, and also employing larger number of steps in the multi-step training approach. These questions remain for future research.

7 Conclusion

Through a series of important observations of results and model improvements, this thesis has shown that hybrid modeling of the n-body problem can be achieved, and that the resulting model can improve the results of the standardized physics-based models. Numerical tests have also demonstrated that a pure data-driven model can predict the orbital motion of planets. The following key observations and modeling approaches, discovered through the work on this thesis, have played a significant role in achieving the results:

Choice and handling of data: When creating a data-driven and hybrid model, the choice of data defines what problem the model will be able to solve. The dataset needs to be sufficiently large for the model to be able to generalize, but also needs to match the specific problem one tries to model. The data also needs to be sorted and scaled in such a way that the model prioritizes prominent data equally, as shown in section 4.3.2 with the guiding example. Another important aspect of data pre-processing is to make the data available for interpretation by the model. Even though patterns in the data clearly exist, the patterns may have to fine details for the model to capture. The data needs to be pre-processed in such a way that the data becomes coarser, e.g., a sparse discretization for time-sequences. However, the amount of data should not be decreased. This is where downsampling came in hand, as demonstrated in section 4.3.1, as it do not remove points in-between the coarse points in the dataset; it constructs different input-output pairs with the same data, only making it coarser in appearance to the N.N., thus maintaining the density and quality data.

Constructing the model: When constructing a data-driven model, there are countless way for tuning hyperparameters, implementing optimization techniques and designing Neural Network architecture. Even though there are some generalized ways for constructing such models, every model needs to be designed and finetuned to each specific problem. This became prominent in section 4.3.2 and 5.2.2, as a new architecture was needed in order to separate information from each planet and for the model to learn specific pattern for each planet. Here, the smaller local N.N. and the pooling layer helped separate planets, as well as split different types of variables. This need for specific model architecture was also true for section 5.2.2 as one needed to implement the P.B.M. inside, the training process of the D.D.M. to construct a stable model.

Model synergy: For the hybrid model, the last critical observation is that one cannot alter a crucial aspect of a sub-model and except its counterpart to

compensate for simplification without any further alterations. This became clear in section 5.2.1, where the exclusion of a correction in the velocity components disrupted one of the key properties of the P.B.M.; it changed the flow of the numerical integrator such that it was no longer symplectic. Thus, it is important to construct hybrid models in such a way that the main properties of the sub-models are kept.

From the observations made throughout this thesis, some interesting questions have come forth:

- How sensitive is the P.B.M. and its symplectic flow to changes in the state-vectors? For how large of an alteration in the flow of position/velocity is the method still stable?
- Can the D.D.M. be altered to consider positions only and predict a position residual that compensates for non-corrected velocity? This can most likely be done by considering a different solution dataset and evaluate and minimize the loss function for different values than what was used in this thesis.
- How does an increase in number of timesteps effect the results of the H.P.D.R.M. with multi-step training approach?
- Can an adaptive optimization algorithm be implemented to improve training of the H.P.D.R.M. with multi-step training approach?

These are questions left for future research.

Due to time constraints, a larger hybrid model including corrections for multiple planets was not tested. Though, the results from section 5.2.2 indicate that such a model is possible and can improve the results of the P.B.M. even more. A more precise data-driven model is also not out of reach, and with more data, it can be expected that a stable D.D.M. for the entire Solar System can be trained and tested.

For future research, a more in-depth analysis of the results would prove interesting. Also, by training and testing the models for equal and longer time-intervals, results would more comparable, and would likely solidify the claim that the H.P.D.R.M. presented in this thesis can be more accurate than the P.B.M.

References

- Alahi, A., Goal, K., Ramanathan, V., Robicquet, A., Fei-Fei, L., Savarese, S. (2016). *Social LSTM: Human Trajectory Prediction in Crowded Spaces.* Proceeding of the IEEE conference on computer vision and pattern recognition: 961-971.
- Alligood, K. T., Sauer, T. D., Yorke, J. A. (1996). *Chaos: An Introduction to Dynamical Systems.* New York: Springer.
- Benettin, G., Giorgilli, A. (1994). *On the Hamiltonian Interpolation of Near-to-the-Identity Symplectic Mappings with Application to Symplectic Integration Algorithms*. Journal of Statistical Physics, **74**(5/6): 1117-1143.
- Blakseth, S. S., Rasheed, A., Kvamsdal, T., San, O. (2022). *Deep neural network enabled corrective source term approach to hybrid analysis and modeling*. Neural Networks, 146: 181-199.
- Boden, M. A. (2016) *AI: Its nature and future*. Oxford University Press.
- Breen, P. G., Foley, C. N., Boekholt, T., Zwart, S. P. (2020). *Newton versus the machine: solving the chaotic three-body problem using deep neural networks.* Monthly Notices of the Royal Astronomical Society, **494**(2): 2465-2470.
- Cai, L., Zhu, Y. (2015). *The Challenges of Data Quality and Data Quality Assessment in the Big Data Era*. Data Science Journal, **14**: 2.
- Chisari, N. E., Zaldarriaga, M. (2011). *Connection between Newtonian simulation and general relativity.* Physical Review D, **83**(12): 123505.
- Claesen, M., De Moor, B. (2015). *Hyperparameter Search in Machine Learning*.
- Daw, A., Karpatne, A., Watkins, W., Read, J., Kumar, V. (2017) *Physics-guided Neural Network (PGNN): An Application in Lake Temperature Modeling.*
- DeVogelaere, R. (1956). *Methods of integration which preserve the contact transformation property of the Hamilton equations.* Technical report (University of Notre Dame. Dept. of Mathematics).
- Diacu, F. (1996). The *Solution of the n-body Problem.* The Mathematical Intelligencer, **18**(3): 66-70.

- Diacu, F., Holmes, P. (1999). *Celestial Encounters: The Origin of Chaos and Stability.* Princeton University Press.
- Ford, M. (2018). *Architects of Intelligence: The truth about AI from the people building it.* Packt Publishing Ltd.
- Goldstein, H., Poole, G., Safko, J. (2002). *Classical Mechanics.*
- Goodfellow, I., Bengio, Y., Courville, A. (2016). *Deep Learning*. MIT Press.
- Greenberg, L. (1990). *The Numerical Solution of the N-Body Problem*. Computers in Physics, **4**: 142.
- Hairer, E. (2006). *Long-time energy conservation of numerical integrators*. Foundations of computational mathematics, Santander 2005: 162-180.
- Hairer, E., Wannar, G., Lubich, C. (2006). *Geometrical Numerical Integration*. Springer Series in Computational Mathematics, **31**.
- Hayes, W. B. (2007). *Is the outer Solar System chaotic?* Nature Physics, **3**(10): 689-691.
- Heggie, D. C. (2005). *The classical gravitational N-body problem.*
- Heggie, D., Hut, P. (2003). *The Gravitational Million-Body Problem, A Multidisciplinary Approach to Star Cluster Dynamics*. Cambridge: Cambridge University Press.
- Hutter, F., Kotthoff, L., Vanschoren, J. (2019). *Automated Machine Learning: Methods, Systems, Challenges*: 3-34. Springer Nature.
- Kingma, D. P., Ba, J. L. (2014). *ADAM: A Method for Stochastic Optimization*.
- Paszke, A., et al. (2019). *Pytorch: An Imperative Style, High-Performance Deep Learning Library.* Advances in Neural Information Processing Systems, **32**.
- Rai, R., Sahu, C. K. (2020). *Driven by Data or Derived Through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques With Cyber-Physical System (CPS) Focus.* IEEE Access, **8**: 71050-71073.
- Raissi, M., Perdikaris, P., Karniadakis, G. E. (2017). *Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations.*
- Renaud, F., Appleton, P. N., Xu, C. K. (2010). *N-body Simulation of the Stephan's Quintet.* The Astrophysical Journal, **724**(1): 80-91.
- Sharma, S., Sharma, S., Athaiya, A. (2017). *Activation functions in neural networks.* Towards Data Science, **6**(12): 310-316.

- Souza, F. A. A., Araújo, R., Mendes, J. (2016). *Review of soft sensor methods for regression applications.* Chemometrics and Intelligent Laboratory Systems, **152**: 69-79.
- Synge, J. L. (1940). *On the electromagnetic two-body problem*. Department of Applied Mathematics, University of Toronto.
- Wang, R., Maddix, D., Faloutsos, C., Wang, Y., Yu, R. (2021). *Bridging Physics-based and Data-driven modeling for Learning Dynamical Systems*. Proceedings of Machine Learning Research, **144**: 1-14
- Willard, J., Jia, X., Xu, S., Steinbach, M., Kumar, V. (2020). *Integrating physics-based modeling with machine learning: A survey.*
- Winter, A. (1941). *The Analytical Foundations of Celestial Mechanics.* Courier Corporation.
- Yam, J. Y., Chow, T. W. (2000). *A weight initialization method for improving training speed in feedforward neural network*. Neurocomputing, **30**(1-4): 219-232.

Appendix A.

A.1 NASA data

The inital values and exact solutions to the n-body problem troughout this thesis is gathered from the Astroquery package in Python. An example on the format of the state vectors when retrived can be found in figure 24. The actual intital values used can be seen in table 17 below.

The mass and orbital period parameter used for the n-body problem throughout this thesis is retrieved using the JPL Horizons online application, available at: https://ssd.jpl.nasa.gov/horizons/app.html#/
An example on how the values from the application looks can be found in figure 25. The values for the used parameters can be seen in table 17 below.

```
JPLHorizons instance "199"; location=500@10, epochs={'start': '2022-1-01', 'stop': '2022-1-02', 'step': '6h'}, id_type=None
        x                   y           ...          vz
        AU                  AU          ...        AU / d
------------------ ------------------- ... -------------------
0.3590281495798379 -0.04075441601397297 ... 0.002595199233022554
0.3583917364766049 -0.03344378031524808 ... 0.002651042414474084
0.3576152599534237 -0.02612008017616273 ... 0.002706514957274584
0.3566973414308776 -0.01878605501316686 ... 0.002761578668607523
0.3556366441583043 -0.01144451610676748 ... 0.002816194064553844
JPLHorizons instance "299"; location=500@10, epochs={'start': '2022-1-01', 'stop': '2022-1-02', 'step': '6h'}, id_type=None
         x                   y          ...          vz
         AU                  AU         ...        AU / d
------------------- ------------------ ... -------------------
-0.06786496495899677 0.7160351157024785 ... 0.001138209566779818
-0.07291444287843717 0.7155127885528458 ... 0.001135449938558974
-0.07796029857582776 0.7149549128411402 ... 0.001132633660099826
-0.08300228103344187 0.7143615133330643 ... 0.001129760864241626
-0.08804013938760374 0.7137326165930999 ... 0.001126831686819953
```

Figure 24: Example on format of state vectors queried from NASA, for Mercury and Venus.

```
*******************************************************************************
 Revised: April 12, 2021              Earth                          399

 GEOPHYSICAL PROPERTIES (revised May 9, 2022):
  Vol. Mean Radius (km)    = 6371.01+-0.02   Mass x10^24 (kg)= 5.97219+-0.0006
  Equ. radius, km          = 6378.137        Mass layers:
  Polar axis, km           = 6356.752        Atmos          = 5.1    x 10^18 kg
  Flattening               = 1/298.257223563 oceans         = 1.4    x 10^21 kg
  Density, g/cm^3          = 5.51            crust          = 2.6    x 10^22 kg
  J2 (IERS 2010)           = 0.00108262545   mantle         = 4.043 x 10^24 kg
  g_p, m/s^2  (polar)      = 9.8321863685    outer core     = 1.835 x 10^24 kg
  g_e, m/s^2  (equatorial) = 9.7803267715    inner core     = 9.675 x 10^22 kg
  g_o, m/s^2               = 9.82022         Fluid core rad  = 3480 km
  GM, km^3/s^2             = 398600.435436   Inner core rad  = 1215 km
  GM 1-sigma, km^3/s^2     =      0.0014     Escape velocity = 11.186 km/s
  Rot. Rate (rad/s)        = 0.00007292115   Surface area:
  Mean sidereal day, hr    = 23.9344695944    land           = 1.48 x 10^8 km
  Mean solar day 2000.0, s = 86400.002        sea            = 3.62 x 10^8 km
  Mean solar day 1820.0, s = 86400.0         Love no., k2    = 0.299
  Moment of inertia        = 0.3308          Atm. pressure   = 1.0 bar
  Mean surface temp (Ts), K= 287.6           Volume, km^3    = 1.08321 x 10^12
  Mean effect. temp (Te), K= 255             Magnetic moment = 0.61 gauss Rp^3
  Geometric albedo         = 0.367           Vis. mag. V(1,0)= -3.86
  Solar Constant (W/m^2)   = 1367.6 (mean), 1414 (perihelion), 1322 (aphelion)
 HELIOCENTRIC ORBIT CHARACTERISTICS:
  Obliquity to orbit, deg  = 23.4392911  Sidereal orb period  = 1.0000174 y
  Orbital speed, km/s      = 29.79       Sidereal orb period  = 365.25636 d
  Mean daily motion, deg/d = 0.9856474   Hill's sphere radius = 234.9
*******************************************************************************
```

Figure 25: Example of parameters of Earth when retrived through the JPL Horizon app.

Table 17: Parameters and initial values of each celestial object.

Celestial object	Mass [kg]	Initial position [AU]	Initial velocity [AU/day]	Orbital period [day]
Sun	$1.9885 \cdot 10^{30}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$	0
Mercury	$3.302 \cdot 10^{23}$	$\begin{pmatrix} -3.49235720 \cdot 10^{-1} \\ -2.68106220 \cdot 10^{-1} \\ 1.04103085 \cdot 10^{-2} \end{pmatrix}$	$\begin{pmatrix} 1.13823601 \cdot 10^{-2} \\ -2.10284211 \cdot 10^{-2} \\ 2.76287013 \cdot 10^{-3} \end{pmatrix}$	87.969
Venus	$4.8685 \cdot 10^{24}$	$\begin{pmatrix} -1.59814174 \cdot 10^{-1} \\ 7.00733609 \cdot 10^{-1} \\ 1.84274483 \cdot 10^{-2} \end{pmatrix}$	$\begin{pmatrix} -1.97900139 \cdot 10^{-2} \\ -4.61041668 \cdot 10^{-3} \\ 1.08514630 \cdot 10^{-3} \end{pmatrix}$	224.70
Earth	$5.97219 \cdot 10^{24}$	$\begin{pmatrix} -2.2980823 \cdot 10^{-1} \\ 9.55937344 \cdot 10^{-1} \\ 4.40432615 \cdot 10^{-4} \end{pmatrix}$	$\begin{pmatrix} -1.70054291 \cdot 10^{-2} \\ -4.08773955 \cdot 10^{-3} \\ -2.63747882 \cdot 10^{-6} \end{pmatrix}$	365.26
Mars	$6.4171 \cdot 10^{23}$	$\begin{pmatrix} 1.39304142 \\ 8.63328171 \cdot 10^{-2} \\ -3.30519817 \cdot 10^{-2} \end{pmatrix}$	$\begin{pmatrix} -3.09310857 \cdot 10^{-4} \\ 1.51490614 \cdot 10^{-2} \\ 3.23956700 \cdot 10^{-4} \end{pmatrix}$	686.98
Jupiter	$1.89818722 \cdot 10^{27}$	$\begin{pmatrix} 1.58233559 \\ -4.92504002 \\ -1.59322341 \cdot 10^{-2} \end{pmatrix}$	$\begin{pmatrix} 7.10431397 \cdot 10^{-3} \\ 2.66301781 \cdot 10^{-3} \\ -1.70244208 \cdot 10^{-4} \end{pmatrix}$	4332.6
Saturn	$5.6834 \cdot 10^{26}$	$\begin{pmatrix} 4.44809246 \\ 7.90792377 \\ -3.16877892 \cdot 10^{-1} \end{pmatrix}$	$\begin{pmatrix} -5.16852649 \cdot 10^{-3} \\ 2.72425204 \cdot 10^{-3} \\ 1.55953611 \cdot 10^{-4} \end{pmatrix}$	10756
Uranus	$8.6813 \cdot 10^{25}$	$\begin{pmatrix} -16.3825466 \\ 8.19949045 \\ 2.44591748 \cdot 10^{-1} \end{pmatrix}$	$\begin{pmatrix} -1.78764841 \cdot 10^{-3} \\ -3.69629464 \cdot 10^{-3} \\ 9.36327710 \cdot 10^{-6} \end{pmatrix}$	30685
Neptune	$1.02409 \cdot 10^{26}$	$\begin{pmatrix} -24.1419975 \\ -18.3204345 \\ 9.33013111 \cdot 10^{-1} \end{pmatrix}$	$\begin{pmatrix} 1.88354153 \cdot 10^{-3} \\ -2.47939313 \cdot 10^{-3} \\ 7.74786039 \cdot 10^{-6} \end{pmatrix}$	60189

A.2 Coding

The codes for this thesis is available at:
https://github.com/AsmSyn/Master_Thesis

The dataset used is not added here, as GitHub repositories do not allow for single datafiles of such size. The dataset can however be queried from NASA using one of the given codes.

All code is written in Phyton 3.9 and MATLAB R2020b. In Python, the following packages was prominently used: Astroquery 0.4.6, Numpy 1.22.2, Matplotlib 3.5.2, Tensorboard 2.9.0, Pytorch 1.11.0.
Note: Tensorboard must be initiated in a terminal and set to read files in same directory as the given code is storing the Tensorboard data.

A.3 Table of abbreviations

Abbreviation	Definition
A.I.	Artificial Intelligence
A.N.N.	Artificial Neural Network
D.D.M.	Data-driven model
D.L.	Deep Learning
F.D.N.N.	Feedforward Deep Neural Network
H.P.D.M	Hybrid-Physics-Data model
H.P.D.R.M.	Hybrid-Physics-Data-Residual model
M.L.	Machine Learning
N.N.	Neural Network
O.D.E.	Ordinary differential equation
O.O.D.	Out-of-distribution
P.B.M.	Physics-based model
R.M.	Residual model