Internship report

Data driven reduced modelling of the Vlasov-Poisson equation

Author:
Guillaume Steimer

Supervisors:
Emmanuel Franck
Vincent Vigon
Laurent Navoret
Nicolas Crouseilles

M2 CSMI 2020-2021

UFR of Mathematics and Computer Science

University of Strasbourg

Acknowledgements

This Master internship was a great and enriching experience that would not have been possible without the support and guidance that I benefited from many people.

First and foremost, I would like to thank my supervisors Messrs. Emmanuel Franck, Vincent Vigon, Laurent Navoret at Strasbourg University and Nicolas Crouseilles at INRIA Rennes - Bretagne Atlantique. Through their writings, their criticism and their good advice, they contributed to the success of this internship. They have fuelled my thinking and helped me to move forward. I would also like to thank them for their support, availability and constant feedback throughout this six months period and despite the distance that sometimes separated us.

Then, I am very grateful for the financial support of INRIA Nancy - Grand Est, which made this project possible. I also would like to thank Strasbourg University for providing me with an office and computer equipment that contributed to the smooth running of the internship.

Last but by no means least, I would like to express my heartfelt thank to my partner Céline Murail. She has been an unfailing supporter of my academic journey for many years and especially during this internship. With her own outlook, she gave me precious advice and helped me during the redaction of the report and the preparation of the defence.

Contents

1 Introduction

When a gas is heated to a very high temperature (from 10^2 to 10^9 kelvins), atom's electrons tend to be ripped from their nuclei. This result in a blend of free electrons, anions, cations, neutral atoms and even aggregates. When the electron density (number of free electrons per unit of volume) is high enough, this substance - although quasineutral - becomes highly electrically conductive. At this point, electromagnetic fields govern its behaviour. This state of matter is called plasma and known as the fourth state of matter following gas, solid and liquid.

Above-mentioned electromagnetic fields are generated by charged particles themselves and their motions. It implies that a charged particle motion depends on all other particles, and inversely. It leads to complex dynamics with multi-scales phenomena. In addition, an external electromagnetic field can be added to the system.

Even though plasma is the most common state of known matter in the Universe: it accounts for 99% of it and we can find it in stars, solar winds and nebula; such matter is rare on Earth. In spite of this, it is not unusual to see a plasma in nature such as in lightnings. Indeed, those intense electrostatic discharge visible during a thunderstorm ionises the air on their path which results in a very hot plasma.

In practice, in a world in contraction where demand for energy is greater than ever, producing and controlling plasma is a major objective. Indeed, as in the sun, a plasma can host a reaction that produces a colossal amount of energy: the thermonuclear fusion reaction during which hydrogen isotopes can merge into a lighter helium atom. It releases a great amount of energy in the form of heat that can be converted into electricity. Thus, it does generate far less waste. In consequence, it is vital to understand plasma dynamics and establish mathematical models of plasma in fusion reactors projects such as the JET (Joint European Torus) in the United Kingdom as seen in figure 1:

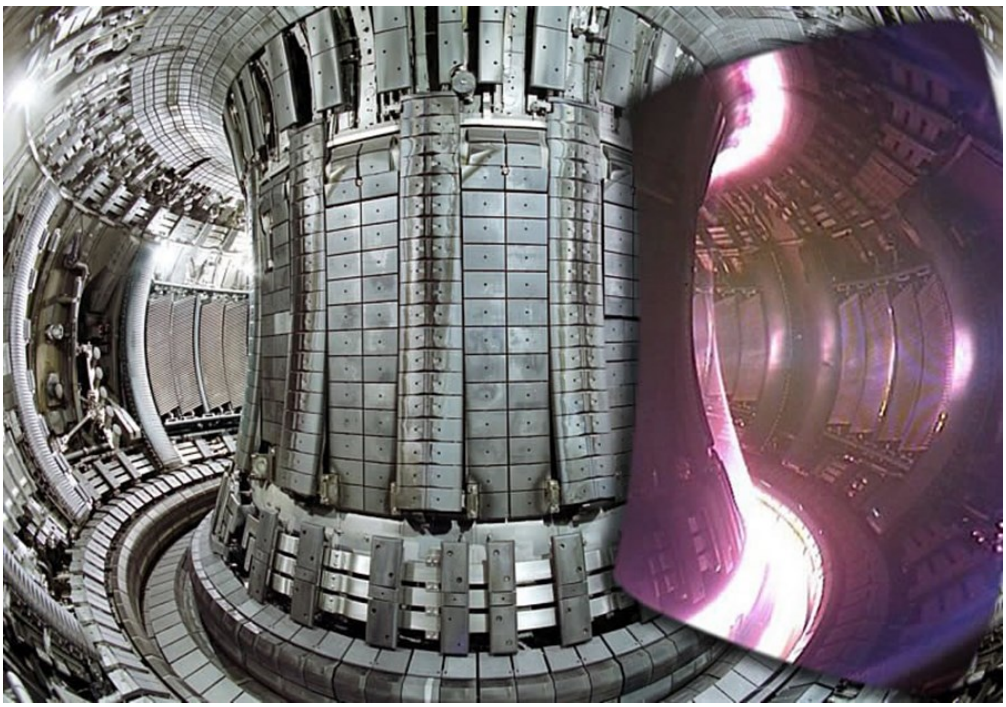


Figure 1: JET with and without plasma (source: iter.org)

Our work focuses on the process of plasma modelling and especially on methods to reduce its computational complexity in numerical simulations: we proceed to a Model Order Reduction (MOR). We start with a physical model i.e. a Partial Differential Equation (PDE) that we discretise into an high dimension Ordinary Differential Equation (ODE). In fact, we introduce a large number of particles representing the model dynamics, each one trajectory follows a set of ODEs. Together, they form this high dimension ODE. The goal is to reduce the latter into a small dimension ODE: the reduced model. The latter truthfully reproduces the large system dynamics in a restricted range of parameters such as time or initial conditions. The reduced system can be depicted as a small number of reduced particles following ODEs.

We resume the process of interest on figure 2. Given an initial state represented by physical variables in a high dimension, we project it on a reduced manifold of small dimension. This is the compression phase. These reduced variables follow a reduced model true to the one that follow prior physical variables in a range of parameters. Then, we simulate the reduced model and make a decompression step on any result of interest as the final state or the plasma distribution at various times.

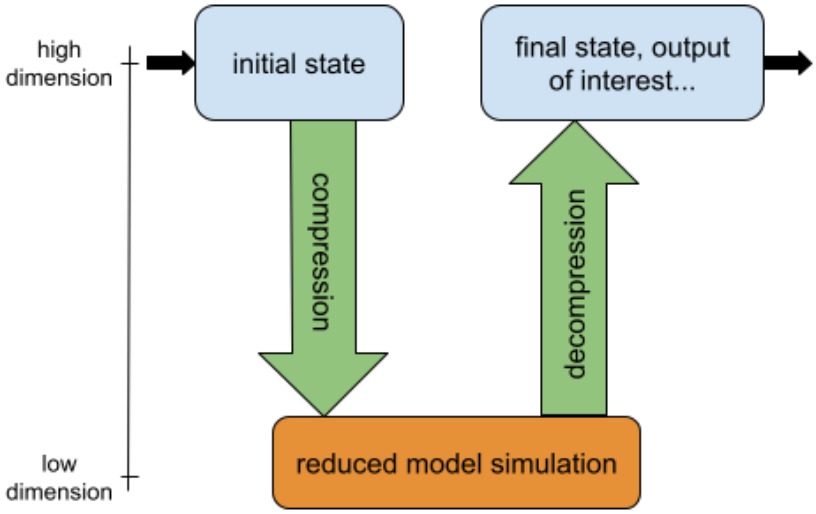


Figure 2: Process of interest

We explore data driven methods for constructing these compression-decompression steps and the associated reduced model. Such methods involve first and foremost the creation of a dataset of particles trajectories following this dynamics; this is where we begin. It is quite comparable to the offline phase of classic reduced modelling such as reduced basis. Then, we focus on a method to compress and decompress variables i.e. project in a reduced space and back these physical variables. At last but not least, we work on a mean to learn a reduced model. To accomplish this, we use different Neural Networks (NN): they are simple to assemble and can be adapted to a wide range of problems. Although methods we use rely on large datasets, they are fast, precise and adaptable.

In this report, we present the initial model and its numerical discretization in section 2. Then, in section 3, we present our data driven methods for the compression-decompression steps: we rely on NNs called autoencoders. Afterwards in section 4, we present an Hamiltonian NN designed to learn the reduced dynamics. In sections 5 and 6, we finally show some results of this process. Finally, we conclude in the last section. In addition, we provide technical elements and details in appendices A beside a personal analysis of the internship in appendix B.

The internship takes place in the TONUS (TOkamaks and NUmerical Simulations) team from INRIA (Institut national de recherche en sciences et technologies du numérique) Nancy Grand Est located at the IRMA (Institut de Recherche Mathématique Avancée) in Strasbourg. This team is specialised in PDEs modelling, numerical schemes and high performance computing for plasma physics simulation. It deals with mathematical and computing issues related to the project ITER (International Thermonuclear Experimental Reactor). This internship is supervised by Emmanuel Franck, Laurent Navoret, Vincent Vigon and Nicolas Crouseilles who are working on machine learning for simulation, numerical modelling of plasma and others.

2 Vlasov-Poisson equation and data generation

We work on a 1D position and 1D velocity plasma.

2.1 Presentation of the Vlasov-Poisson equation

First of all, we consider a kinetic representation of plasma. Particles are characterised by their statistical distribution function $f(x, v, t)$ in position-velocity phase space. It satisfies the Vlasov-Poisson equation; it describes a collisionless and non-relativistic plasma. The latter is submitted to an electric field either external $E_{ext}(x, t)$ or self-consistent $E(x, t)$ i.e. generated by the charged particles themselves. We place the system at the electric limit $|E + E_{ext}| \gg c\,|B|$, with B the magnetic field, c the speed of light in vacuum. It allows us to neglect B at zeroth order.

Definition 2.1 (Vlasov-Poisson equation)**.** *The Vlasov equation is:*

$$\partial_t f + v\nabla_x f + \frac{q}{m}(E + E_{ext})\nabla_v f = 0, \tag{1}$$

where $f(x, v, t)$ is a particle distribution at time $t \in \mathbb{R}_+$, $q \in \mathbb{R}$ is their charge, $m > 0$ their mass, $x, v \in \mathbb{R}$ their positions and velocities. The self-consistent electric field $E(x, t) \in \mathbb{R}$ satisfies the Poisson equation:

$$\Delta\phi = -\frac{\rho}{\epsilon_0}, \quad E = -\nabla\phi \tag{2}$$

where $\phi(x, t)$ is the electric potential, $\rho(x, t)$ is the charge density:

$$\rho(x, t) = q \int f(x, v, t)dv,$$

and ϵ_0 is the permittivity of free space. E_{ext} is determined by:

$$E_{ext} = -\nabla\phi_{ext}$$

where ϕ_{ext} is a given external electric potential.

The Vlasov equation is a non linear transport equation. More details on the origin of the Vlasov-Poisson equation are given in the annex (A.1).

2.2 Discretisation of the Vlasov-Poisson equation

The model is established, we need to introduce a numerical technique that faithfully reproduces the dynamics of the prior in order to create the above-mentioned dataset. We start by making a discretisation of the model with [1]. We add an initial condition $f^0(x^0, v^0) := f(x(0), v(0), 0)$.

To do so, we introduce a discrete particles distribution:

$$f_N(x(t), v(t), t) = \sum_{i=1}^{N} w_i \delta(x - x_i(t))\delta(v - v_i(t))$$

with N if the number of particles, $x_i, v_i \in \mathbb{R}$ are the position and velocity of the i^{th} particle, and w_i its weight in the distribution.

Remark. *The particles considered are computational objects. One of these could represent thousands of ions. Indeed, a single litre of gas contains about 10^{22} particles. In a simulation, we typically use $10^3 - 10^6$ particles. As we will see below, the acceleration only depends on the ratio q/m, hence dynamics are not changed.*

We present a fundamental result for the following numerical method.

Property 2.2. *If the above-mentioned set of particles is solution of the system of ordinary differential equations:*

$$\begin{cases} \dot{x}_i = v_i, \\ \dot{v}_i = \dfrac{q}{m}(E + E_{ext})(x_i(t)), \\ x_i(0) = x_i^0, v_i(0) = v_i^0, \end{cases} \tag{3}$$

then, f_N is a solution of the Vlasov-Poisson equation defined in definition 2.1 in the sense of distributions and associated to the initial condition $f_N^0(x^0, v^0) = \sum_{i=1}^N w_i \delta(x - x_i^0)\delta(v - v_i^0)$.

Proof. Let $\psi \in C_c^\infty(\mathbb{R}^d \times \mathbb{R}^d \times (0, +\infty))$ be a test function and $T \in (0, +\infty)$ such that $[0, T]$ bounds the support of ψ on the third variable.

We remind that, using distribution calculus:

$$\langle f_N, \psi \rangle = \sum_{i=1}^N \int_0^T w_i \psi(x_i(t), v_i(t), t) dt,$$

Therefore, with the derivation property:

$$\langle \partial_t f_N, \psi \rangle = -\langle f_N, \partial_t \psi \rangle = -\sum_{i=1}^N \int_0^T w_i \partial_t \psi(x_i(t), v_i(t), t) dt.$$

Thus, let us write the time derivative of ψ:

$$\frac{d\psi}{dt}(x_i(t), v_i(t), t) = \dot{x}_i(t)\nabla_x \psi + \dot{v}_i(t)\nabla_v \psi + \partial_t \psi(x_i(t), v_i(t), t).$$

We inject it in the previous equation:

$$\langle \partial_t f_N, \psi \rangle = \sum_{i=1}^N w_i \left[\int_0^T \left(\dot{x}_i(t)\nabla_x \psi(x_i(t), v_i(t), t) + \dot{v}_i(t)\nabla_v \psi(x_i(t), v_i(t), t) \right) dt + \underbrace{\int_0^T \partial_t \psi dt}_{=0, \psi \in C_c^\infty} \right].$$

Then, through identification:

$$\begin{aligned} \langle \partial_t f_N, \psi \rangle &= \sum_{i=1}^N w_i \int_0^T \left[v_i \nabla_x \psi(x_i(t), v_i(t), t) + \frac{q}{m}(E + E_{ext})(x_i, t)\nabla_v \psi(x_i(t), v_i(t), t) \right] \\ &= -\langle v \nabla_x f_N, \psi \rangle - \left\langle \frac{q}{m}(E + E_{ext})\nabla_v f_N, \psi \right\rangle. \end{aligned}$$

At last, we retrieve Vlasov-Poisson equation:

$$\left\langle \partial_t f_N + v\nabla_x f_N + \frac{q}{m}(E + E_{ext})\nabla_v f_N, \psi \right\rangle = 0, \ \forall \psi \in C_c^\infty(\mathbb{R}^d \times \mathbb{R}^d \times (0, +\infty))$$

and f_N satisfies the Vlasov-Poisson equation in the sense of distributions. □

Remark. *We remark that equations (3) are precisely the equations given by Newton laws of motion. In fact, the Vlasov equation has been obtained with Newton laws as $N \to +\infty$.*

Thanks to property 2.2, we have a forthright method to simulate a plasma: we take an amount of charged particles with an initial condition; we compute $E + E_{ext}$ at each time-step and update positions and velocities by solving equations (3). This is the principle of the PIC (Particle In Cell) method; it is described in section 2.4.

2.3 Elements of Hamiltonian mechanics

We want to solve equations (3) which is in fact an hamiltonian system, as we see below. We note $x = (x_i) \in \mathbb{R}^N$ and $v = (v_i) \in \mathbb{R}^N$.

Before going further, we must introduce some elements of Hamiltonian mechanics, a framework that provides a general and abstract quiver to describe physical systems. We focus on conservative systems - as the one we study - those are said to be hamiltonians. To achieve that, we define the hamiltonian $\mathcal{H}$ of such a system. It allows to describe its dynamics.

Definition 2.3 (Symplectic formulation)**.** *Let be* $\mathcal{H} : \mathbb{R}^{2N} \to \mathbb{R}$, $u := \begin{pmatrix} x \\ v \end{pmatrix} \in \mathbb{R}^{2N}$ *follows an hamiltonian dynamics associated to* $\mathcal{H}$ *if it satisfies the differential system:*

$$\dot{u} = \mathbb{J}_{2N} \nabla_u \mathcal{H}(u)$$

where $\mathbb{J}_{2N} := \begin{pmatrix} 0 & \mathbb{I}_N \\ -\mathbb{I}_N & 0 \end{pmatrix}$ *is a skew-symmetric matrix. Then,* $\mathcal{H}$ *is called hamiltonian of the system. This differential system is named the symplectic formulation.*

In addition, the Hamiltonian is naturally preserved by the symplectic form: $\mathcal{H}(u(t)) = \mathcal{H}(u(0))$*, for all* $t > 0$*.*

In our case, the hamiltonian is:

$$\mathcal{H}(u(t)) = \sum_{i=1}^{N} \left[\frac{1}{2} v_i^2(t) + \frac{q}{m} \phi(x_i(t)) \right]$$

We find this expression starting from the discrete Vlasov-Poisson equation (2.2), we identify the hamiltonian using definition 2.3:

$$\begin{cases} \dot{x}_i = v_i = \dfrac{\partial \mathcal{H}}{\partial v_i}(u) \\ \dot{v}_i = \dfrac{q}{m} E(x_i) = -\dfrac{q}{m} \nabla \phi(x_i) = -\dfrac{\partial \mathcal{H}}{\partial x_i}(u), \end{cases}$$

Remark. *In physics, an hamiltonian represents the energy on the system, we note that it is the sum of kinetic and potential energies of every particle. Nonetheless, this is not true in general. In the reduced model, the hamiltonian does not corresponds to a physical energy.*

In our case, the hamiltonian has an additional property: it is separable i.e. it can be expressed as a sum of a function depending solely on positions and a function depending solely on velocities:

$$\mathcal{H}(u) = \mathcal{H}^1(x) + \mathcal{H}^2(v).$$

From a computational point of view, preserving the hamiltonian i.e. the symplectic structure is an interesting property as it enables to ensure long time stability of the numerical method. Such methods are called symplectic and one of such method is the Störmer-Verlet scheme given in [2]. In the separable case, its n^{th} step is written as follows:

$$\begin{cases} v^{n+\frac{1}{2}} = v^n - \dfrac{1}{2} \Delta t \nabla_x \mathcal{H}^1 \left(x^n \right), \\ x^{n+1} = x^n + \Delta t \nabla_v \mathcal{H}^2 \left(v^{n+\frac{1}{2}} \right), \\ v^{n+1} = v^{n+\frac{1}{2}} - \dfrac{1}{2} \Delta t \nabla_x \mathcal{H}^1 \left(x^{n+1} \right), \end{cases} \tag{4}$$

where Δt is the time step and (x^n, v^n) denotes the numerical approximation of (x, v) at time t^n. In addition of preserving the hamiltonian, it is fast and precise. This scheme is also called Leapfrog due to its chased steps between x and v we can see on figure 3:

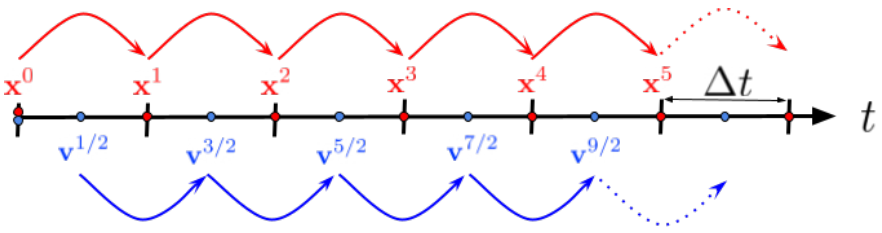


Figure 3: Illustration of the Leapfrog scheme

You can find an illustration of the use of non symplectic schemes on long run simulations in annex A.4.

2.4 Data generation with a PIC algorithm

To reproduce the dynamics at a numerical level, we use a symplectic scheme such as equation (4). To compute the electric field at each time step, we pass by the electric potential. As N could be large, this calculation can be extensive. We present a PIC (Particle In Cell) method as presented in [3, 4]. It is a common tool used to bypass this difficulty. Furthermore, this method is well generalised with a 3D Vlasov model.

2.4.1 PIC presentation

In this type of methods, we consider so-called macroparticles (or super-particles) (x_i, v_i) already introduced in property 2.2. A macroparticle is a computational object composed of an aggregate of real particles. In our case, it can be millions of ions. PIC includes two types of interactions: binary PP (particle-particle) interactions and PM (particle-mesh) interactions: particles only interact through average fields. We do not consider the prior due to computation cost in $\mathcal{O}(N^2)$ complexity. In the latter, we use a mesh of size $N_x + 1$ with $N_x \ll N$ as a calculus intermediate as detailed below. It results in a $\mathcal{O}(NN_x)$ complexity which is a major improvement.

A PIC algorithm can be divided into two parts: a particle mover that solves the equation of motion (3) and a field solver that determines the electric field using Poisson equation.

The particle mover: also called pusher solves the equation of motion as described in property 2.2. We use a Störmer-Verlet scheme presented in equation (4). We note $E^n(x^n)$ the self-induced electric field at particles positions x^n at the n^{th} time step. We note T the total time of simulation and Δt the time step. It is rewritten as:

$$\begin{cases} \dfrac{1}{\Delta t}\left(x^{n+1} - x^n\right) = v^{n+\frac{1}{2}} \\ \dfrac{1}{\Delta t}\left(v^{n+\frac{1}{2}} - v^{n-\frac{1}{2}}\right) = \dfrac{q}{m}\left(E_{ext}(x^n) + E^n(x^n)\right) \end{cases} \tag{5}$$

The field solver: regarding equation (5), we have to compute $E^n(x^n)$ with a PM method. On a mesh: we compute the electric potential via the charge calculus, then we derive the electric field. Finally, we interpolate it on particles positions.

Let $[0, L]$ be the space domain partitioned into $N_x + 1$ cells of size $\Delta x = \frac{L}{N_x}$. Each node has a charge $\rho(i\Delta x) = \rho_i^n$, and a potential $\phi_i^n, i \in \{0, \cdots, N_x\}$. We set $\epsilon_0 = 1$. The latter PM method to compute $E^n(x^n)$ follows those steps:

- we set $\rho_i^n = 0$ at every node,
- for every particle at position $\tilde{x}$ and charge q:
  - ◇ we find its position on the grid between the i^{th} and $(i+1)^{th}$ nodes with $i = \lfloor \tilde{x}/\Delta x \rfloor$,
  - ◇ we split the charge of the particle between these nodes according to the particle relative position. The distance between the i^{th} node and the particle is $h = \tilde{x} - i\Delta x$ so the splitting is:

$$\rho_i^n \mathrel{+}= \overline{\rho}\frac{\Delta x - h}{\Delta x}, \quad \rho_{i+1}^n \mathrel{+}= \overline{\rho}\frac{h}{\Delta x} \quad \text{where } \overline{\rho} = \frac{q}{\Delta x}.$$

  Let us take a look on figure 4 to depict why. We observe a charge q between the nodes i and $i+1$. The closer q is to i, the higher the proportion of q belongs to i and conversely. This proportion correspond to the red length over the cell length. Therefore, we consider that the proportion $\dfrac{\Delta x - h}{\Delta x}$ of the charge belongs to i (and the subsidiary part to $i+1$). As the charge density is lineal, we consider proportions of $\overline{\rho}$:

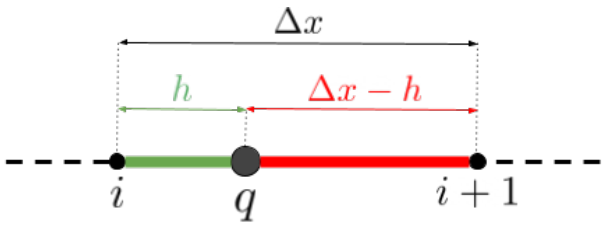


Figure 4: Illustration of the charge splitting

- we solve $\Delta\phi^n = -\rho^n$ with a finite difference method. You can find more details on the finite differences schemes of the field solver, including details on boundary conditions in annex A.2,
- we compute $E^n = -\nabla\phi^n$ on the mesh with a centered finite difference formula,
- finally, we can use a linear interpolation to find the electric field at each particle's position.

Let us show the field solver's process graphically on a simple example with $N = 3$, $N_x = 6$, $q/m = +1$ and Neumann boundary conditions ($\phi' \equiv 0$ on the borders and $\phi(0) = 0$). We start with the computation of ρ^n and ϕ^n on figure 5. Mesh cells are delimited by black dashes. In black dots, we see particles at their positions. We observe charge splittings with ρ^n (orange). ρ^n is null on borders as their are no particles in borders cells. The maximum is between the second and the third cell as this is the only node neighbour to two particles. Then, we observe the derived potential (green).

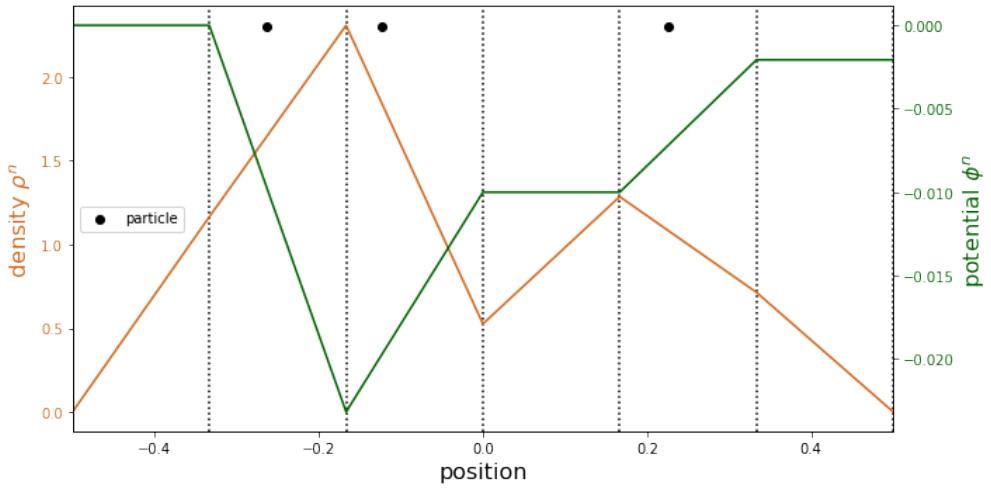


Figure 5: Grid calculation of the charge and the potential

Next, we look at E^n calculation on figure 6. From ϕ^n (green), we derive the electric field on the grid. We observe its linear interpolation on the grid with the colormap: warm colours represent positive field and cold colours represent negative field. As $q/m = +1$, the acceleration is E^n: in red areas particles go to the right and in blue areas particles go to the left, as represented by lime arrows.

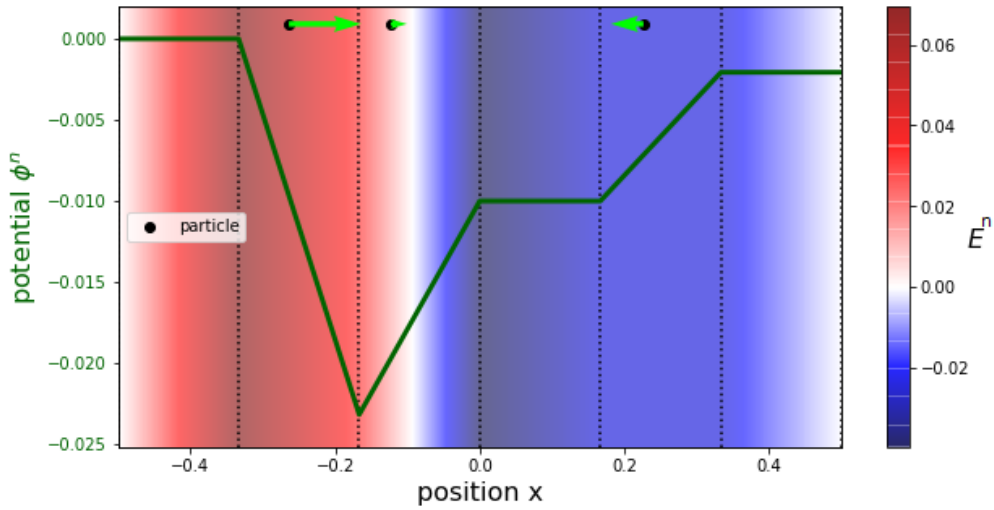


Figure 6: Grid calculation of of E^n and interpolation

We end up with a practical mean to generate datasets that we used to train our machine learning models and test their accuracy. This generation and training constitute a costly offline phase as in the reduced basis.

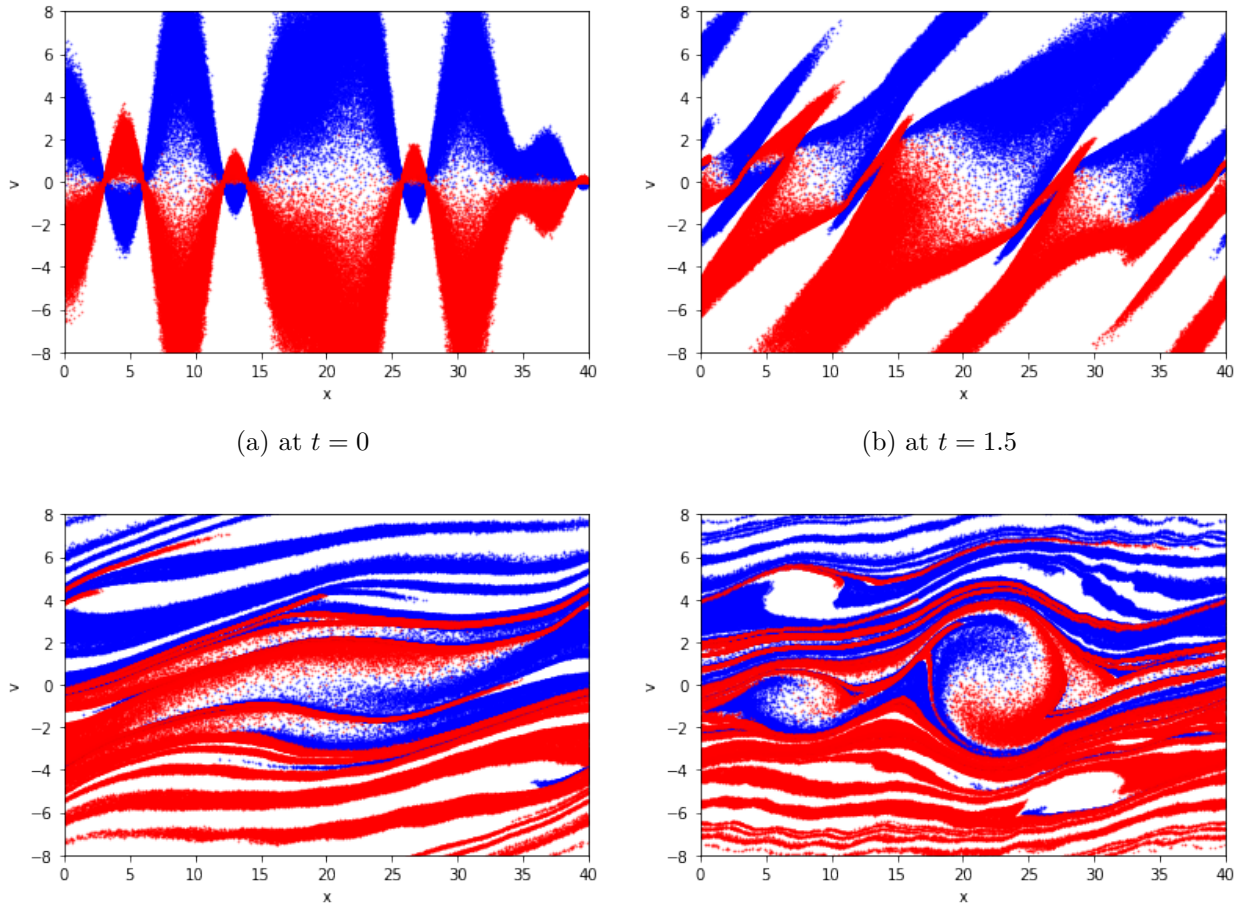
2.4.2 Illustration

Let us look at a small illustration of this process. We consider two perturbed beams of electrons moving in opposite directions and only subject to a self-consistent electric field i.e. we only consider repulsive electron-electron interactions.

We consider $N = 400\,000$ macro-particles with ratio $q/m = -1$ in a periodic spacial domain $[0, 40]$ and with a time-step $\Delta t = 0.5$. We compute the self-consistent field over a grid of $N_x = 1000$ cells.

To facilitate the observation, half of particles are coloured in red and the remaining part in blue. We see the evolution of this system on phase portraits on figure 7 with position on abscissa and velocity on ordinate.

The sub figure 7a shows the initial distribution. On 7b at time $t = 1.5$, we observe a natural motion: particles with a positive velocity move to the right and vice versa. Next, we begin to observe electron-electron interactions on 7c: there are some oscillations at time $t = 8$. Then, these repulsive interactions create vortices at time $t = 17.5$ on sub figure 7d.



(a) at $t = 0$

(b) at $t = 1.5$

(c) at $t = 8$

(d) at $t = 17.5$

Figure 7: PIC simulation with opposite direction's electron beams

3 Reduced variables by autoencoders neural networks

As data can be generated, the next step is to manage reduction i.e. compress and decompress variables, using deep learning methods. We note the pair position-velocity $u = \begin{pmatrix} x \\ v \end{pmatrix} \in \mathbb{R}^{2N}$ from the PIC method.

We want to find a pair of reduced position-velocity $\overline{u} = \begin{pmatrix} \overline{x} \\ \overline{v} \end{pmatrix} \in \mathbb{R}^{2K}, K \ll N$ that follows a reduced dynamics true to the original one, in a restricted range of parameters such as time or initial conditions. Passing from u to $\overline{u}$ would allow us to transform a dimension $2N$ ODE to a much smaller one of dimension $2K$.

First, we expose the limits of classical reduction methods and then we present ours based on autoencoder Neural Networks (NN). We are interested in reduction methods that preserve the hamiltonian.

3.1 Limitations of classical reduction methods

Classical reduction methods on the Vlasov-Poisson model have been explored such as [5] in which they found a linear reduction called the PSD (Proper Symplectic Decomposition) and inspired from the POD (Proper Orthogonal Decomposition) which preserve the symplectic structure of the reduced model. Moreover, the PSD directly provides a reduced model on the reduced variable $\overline{u}$.

In fact, the ODE reduction is performed with respect to time and other parameters like those defining the initial condition. That is to say we make the hypothesis a solution u is parametrised by t and μ. In other words, we make the approximation $u(t, \mu) = \mathcal{G}(\xi(t, \mu))$ with $\mathcal{G}$ an $\mathbb{R}^{2K} \to \mathbb{R}^{2N}$ transformation. The PSD makes the hypothesis that the reduction $\mathcal{G}(\xi) = A\xi, A \in \mathbb{R}^{2N \times 2K}$ is linear.

When the PDE is linear i.e. the electric field is purely external, the linear reduction is satisfactory. Otherwise, when the PDE is strongly non-linear i.e. with a self-induced electric field, this hypothesis is not satisfied and results are unacceptable. You can find more details on the PSD in annex A.3. An example is detailed in section 5.1.

3.2 Reduction method by autoencoder neural networks

We want to explore another type of reduction method based on NNs. In this report, we focus on feed forward neural networks: they are composed by a succession of layers. Inputs go through the first layer, the output of one layer is the input of the next until the final output. In fact, a layer is the operation $z' = \sigma(Wz + b)$ with z the input, W a weights matrix, b the bias, σ a non-linearity such as tanh and z' is the output. W dimensions are given by x dimension and z' desired dimension called the number of neurons or units of the layer. The union of weights matrix and biases form the parameters of the NN. The latter are tuned to fit to a data set of inputs and corresponding outputs in order to minimise a cost or loss function.

We work on NNs called AutoEncoders (AE). As the PSD, it is an unsupervised learning algorithm: it means that data are not labelled. AEs have been developped for this type of compression-decompression tasks. In other words, an AE encodes data. The encoding process is fitted comparing the original data with the regenerated one from the encoding (a successive compression and decompression).

We focus on the red part of the process scheme on figure 8. The construction of the model on the reduced variables will be the subject of the next section 4.

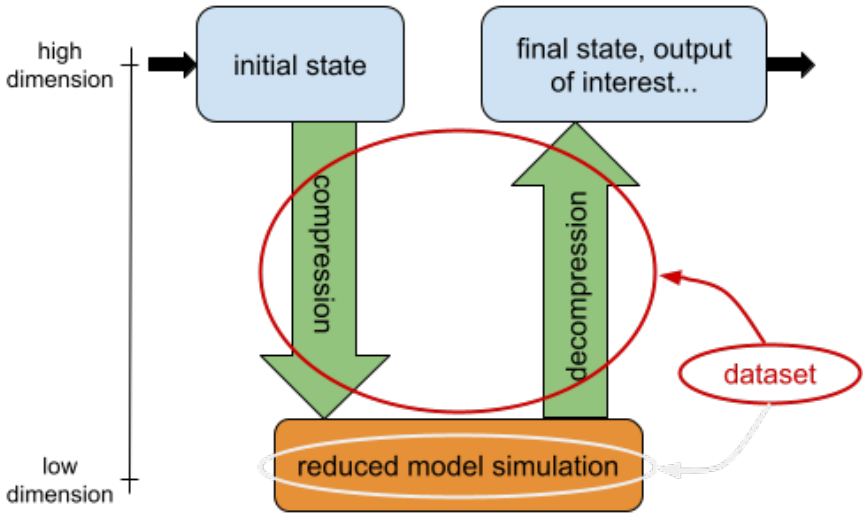


Figure 8: Process of interest

In the following, we present different AE architectures i.e. the NNs layouts and then the frame that includes them and manages compression and decompression.

3.3 Autoencoders architectures

Architectures always have more or less the same shape: a succession of neuron layers in a form of a double bottleneck, the first one for the compression and the second one for decompression, as can be seen on figure 9. The union of these constitutes the AE.

We encode a high dimension input $u \in \mathbb{R}^n, n \in \{N, 2N\}$ into a low dimension code $\overline{u} \in \mathbb{R}^k, k \in \{K, 2K\}$. k and n values will be detailed later. The first bottleneck represents the encoder $\mathcal{F}_\theta : \mathbb{R}^n \to \mathbb{R}^k$ with $\theta \in \Theta$ the set of NN parameters such that $\mathcal{F}_\theta(u) = \overline{u}$. The second bottleneck is the decoder $\mathcal{G}_\theta : \mathbb{R}^k \to \mathbb{R}^n$ such that $\mathcal{G}_\theta(\overline{u}) = u$. The objective of the training is to find optimal parameters for the encoder and the decoder:

$$\hat{\theta} = \underset{\theta \in \Theta}{\operatorname{argmin}} \sum_{u_i \in U_{\text{dataset}}} \left\| u_i - \left(\mathcal{G}_\theta \circ \mathcal{F}_\theta\right)(u_i) \right\|_2$$

where $U_{\text{dataset}} \subset \mathbb{R}^n$ is the training dataset composed by all position-velocity pairs u obtained from a PIC algorithm. Then, we choose $\mathcal{F}_{\hat{\theta}}, \mathcal{G}_{\hat{\theta}}$ for encoder and decoder respectively. In other words, we want, for any data u from the dataset, that the result $\left(\mathcal{G}_\theta \circ \mathcal{F}_\theta\right)(u)$ of the compression followed by the decompression is close to the input data u. From now on, we note $\theta := \hat{\theta}$.

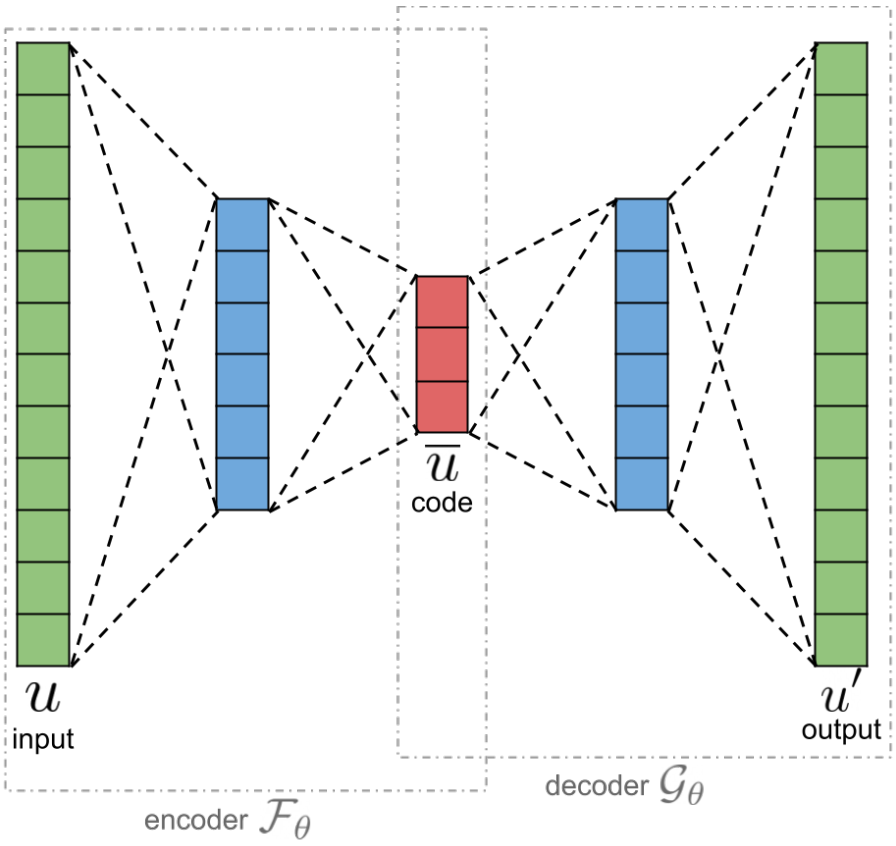


Figure 9: Autoencoder scheme

Remark. *This figure let us think that the autoencoder must be symmetric i.e. $\mathcal{F}_\theta$ and $\mathcal{G}_\theta$ must have symmetric layers. Nonetheless, it is not mandatory.*

For any training input u, the loss or cost function is, as expected:

$$\mathcal{L}_{AE} = \|u - (\mathcal{G}_\theta \circ \mathcal{F}_\theta)(u)\|_2$$

Remark. *AE learn the reduction and its pseudo-inverse which are potentially strongly non-linear unlike the PSD.*

An AE is intended to be a close representation of the identity on the dataset. It passes by a low dimension representation which we use as reduced variables. Let us present two different architectures we designed:

Dense: The encoder is composed of multiple dense layers, the first one input dimension is n and the last one output dimension is k and inversely in the decoder. We construct the decoder as a reflection of the encoder. The main drawback of this architecture is the large number of parameters, especially when $N = 10^4$ or more. In these cases, this architecture becomes inoperable. You can find more details on its construction in annex A.5.2.

Light: Instead of using dense layers i.e. every unit of a layer is connected to every unit of the next layer, we use a lighter structure. A connection between two units signifies that the input from the second unit depends on the output of the first one with a parameter called weight. We make packets of size l units in a given layer. Each packet is fully connected to one packet of size m units in the next layer. The union of size m packets composes this layer. We can mix these or not. The (l, m) pairs list define the architecture. It reduces the number of connections i.e. the number of parameters or weights. We keep a dense layer before the encoder output.

Let us illustrate it on an example without packet mixing. We focus on an encoder part on figure 10. The first layer is composed of 16 units. The first pair is $(4, 3)$: as visible in red rectangles, we make groups of 4 units in the first layer and connect them with 3 units groups in the second layer. It results in a 12 units second layer. Visible with blue rectangles, the second pair is $(4, 3)$: we make packets of 4 units in the second layer and connect them to packets of 3 units in the third layer. With orange rectangles, we observe the last pair which is $(6, 2)$.

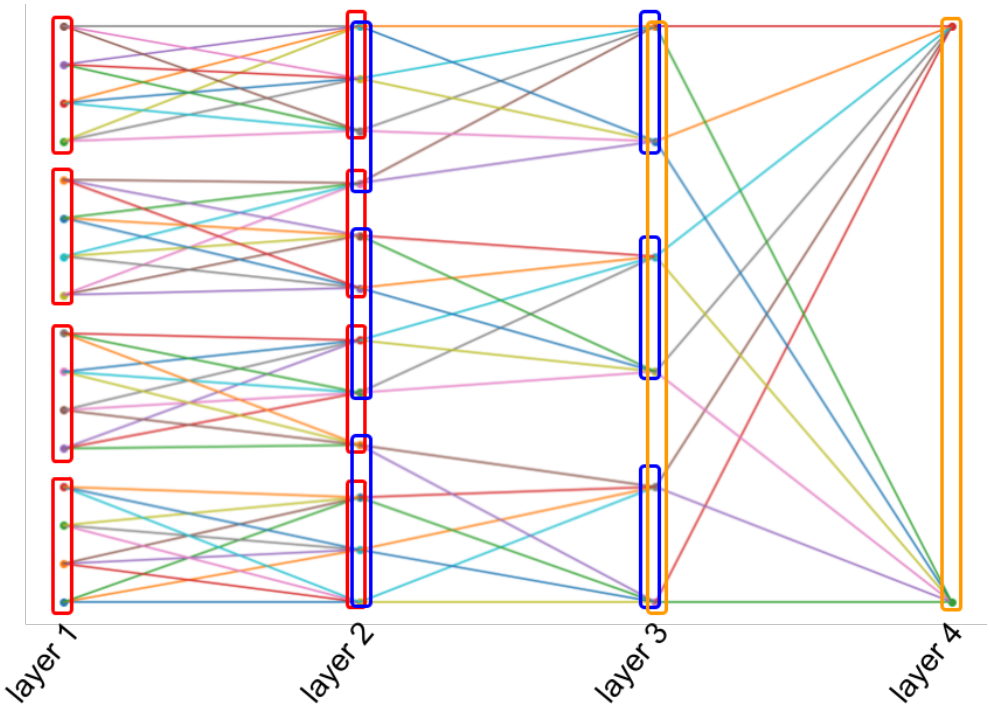


Figure 10: Light architecture example (source: Vincent Vigon)

The construction of the network i.e. the (l, m) pairs list is much more experimental. In spite of that, the light architecture is a featherweight when it comes to the number of parameters. It is vital with large systems i.e. $N = 10^4$ and more.

3.4 Autoencoders frames

In our case, an input u is composed of positions concatenated with velocities. In order to build relevant reduced models, we can create autoencoders adapted to this split structure. We present three different manners to assemble dense or light architectures.

Classic: We use a single network to code $u \in \mathbb{R}^{2N}$ i.e. $n = 2N, k = 2K$, we encode data using:

$$\begin{cases} \mathcal{F}_\theta : & \mathbb{R}^{2N} \to \mathbb{R}^{2K} \\ & u \mapsto \overline{u} \end{cases} \qquad \begin{cases} \mathcal{G}_\theta : & \mathbb{R}^{2K} \to \mathbb{R}^{2N} \\ & \overline{u} \mapsto u \end{cases}$$

We do not take into account u structure. Going further, We note them AE and LAE, with a dense and a light architecture, respectively. It is a good start to test different architectures.

Separated: We use two networks to exploit the fact that u is a concatenation. One manages positions and the other manages velocities i.e. $n = N, k = K$. Encoders $\mathcal{F}^1$ and $\mathcal{F}^2$ are $\mathbb{R}^N \to \mathbb{R}^K$ transformations and conversely for decoders $\mathcal{G}^1$ and $\mathcal{G}^2$. We use:

$$\begin{cases} F^1_\theta(x) = \overline{x} \\ F^2_\theta(v) = \overline{v} \end{cases} \qquad \begin{cases} G^1_\theta(\overline{x}) = x \\ G^2_\theta(\overline{v}) = v \end{cases}$$

If architectures are dense, we note this frame SAE and if they are light, we note it SLAE. As for the loss, we note $\mathcal{F}_\theta(u) := \begin{pmatrix} F^1_\theta(x) \\ F^2_\theta(v) \end{pmatrix}, \mathcal{G}_\theta(\overline{u}) := \begin{pmatrix} G^1_\theta(\overline{x}) \\ G^2_\theta(\overline{v}) \end{pmatrix}$. As numerical results will show in section 5, the separated frames permits to use lighter architectures. This gives a bit more capacity to handle large systems.

Mono-block: With a single network, we exploit the fact that $v = \dot{x}$. We have $n = N, k = K$. We compute outputs with the jacobians of the model:

$$\begin{cases} F_\theta(x) = \overline{x} \\ d_x F_\theta v = \overline{v} \end{cases} \qquad \begin{cases} G_\theta(\overline{x}) = x \\ d_{\overline{x}} G_\theta \overline{v} = v \end{cases}$$

If the network is dense, the frame is noted MAE and in the other case it is noted MLAE. Due to the use of a single network, the mono-block frame provides the lightest AE. Nevertheless, data go through the NN by batches: the shape of an input is (N_t, N) with N_t the number of time steps. The shape of coded data is (N_t, K). Hence, jacobians have a shape (N_t, K, N) or (N_t, N, K). If we consider $N_t = 10^3, N = 10^4, K = 10$, the latter tensor has 10^8 coefficients which make GPU computations complicated. We can bypass this issue by passing data by batches with size a fraction of N_t, but it makes calculations slower.

A first example of the superiority of AEs compared to the PSD is provided in section 5.1. An in-depth analysis of autoencoders performance and construction strategies is provided in section 5.2. In addition, you can find more training details in annex A.5.

Contrary to the PSD, AE does not provide a simple expression for the reduced model. There exists a mean to express the projection hence the reduced model of an AE according to [6]. Nevertheless, it can be complex. As a consequence, we prefer to learn the reduced model. It is achieved in the next section 4.

4 Hamiltonian reduced model using neural networks

In order to simulate the dynamics of reduced variables as shown on the red part of the process on figure 11, the reduced model is yet to be found. This section focus on the learning of this model.

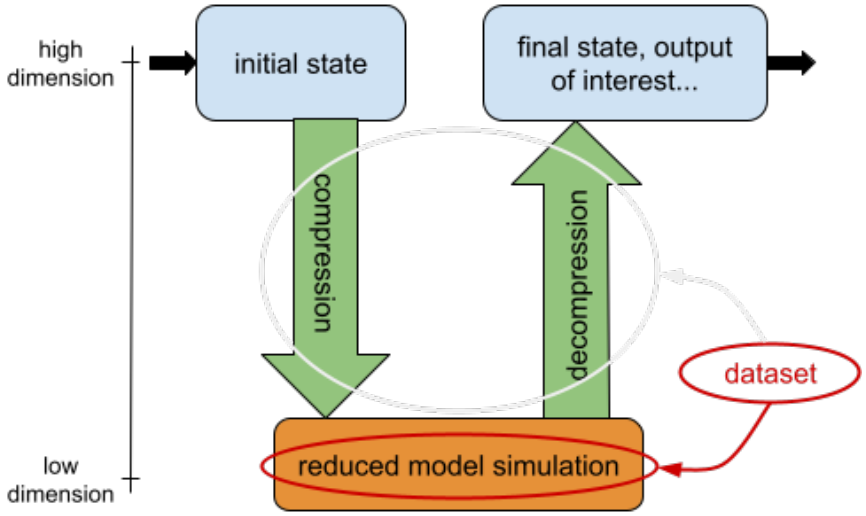


Figure 11: Process of interest

As above-mentioned, it is difficult to write the projection of an AE as in [6]. An alternative is to learn it. To do so, the idea is to learn a model that, with an input $\overline{u}$, predict the corresponding output $\dot{\overline{u}}$. In other words, the model approaches time derivation. Unfortunately, this type of model does not work: they do not preserve the hamiltonian (see section 2.3) and could lead to numerical instabilities.

As in the PSD (see annex A.3), we aim to preserve an hamiltonian structure in the reduced model to guarantee numerical stability in long time simulation. In annex A.4, you can find an example that explains why this is important.

As a consequence, the objective is to learn an hamiltonian reduced system. This solution has been explored in my previous semester project [7]. It is called an Hamiltonian Neural Network (HNN) from [8]: we learn directly the hamiltonian of the reduced system. In contrary to the hamiltonian of the original model, this one does not correspond to any energy. We derive $\dot{\overline{u}}$ using the symplectic formulation i.e. Hamilton's equations defined in section 2.3. In annex A.4, we can observe the success of an HNN.

Thus, knowing that the physical hamiltonian is separable, we make the assumption that the reduced variables hamiltonian also is. We use a pair of networks $\mathcal{H}_\theta^1, \mathcal{H}_\theta^2$ with $\theta \in \Theta$ their parameters, so that the reduced variables hamiltonian model is $\mathcal{H}_\theta := \mathcal{H}_\theta^1 + \mathcal{H}_\theta^2$. To compute the reduced dynamics, we use Hamilton's equations:

$$\begin{cases} \dot{\overline{x}}_\theta = \dfrac{\partial \mathcal{H}_\theta}{\partial \overline{v}}(\overline{u}) = \nabla_{\overline{v}} \mathcal{H}_\theta^2(\overline{v}), \\ \dot{\overline{v}}_\theta = -\dfrac{\partial \mathcal{H}_\theta}{\partial \overline{x}}(\overline{u}) = -\nabla_{\overline{x}} \mathcal{H}_\theta^1(\overline{x}). \end{cases}$$

where $(\dot{\overline{x}}_\theta, \dot{\overline{v}}_\theta)$ shall be close to the true reduced variables derivatives $(\dot{\overline{x}}, \dot{\overline{v}})$. In fact, we use a finite difference to approximate the latter, as these quantities are not always available and can be estimated from the input dataset. Hence, the loss is:

$$\mathcal{L}_{HNN} = \left\| \nabla_{\overline{v}} \mathcal{H}_\theta^2(\overline{v}(t)) - \frac{\overline{x}(t+\Delta t) - \overline{x}(t-\Delta t)}{2\Delta t} \right\|_2 + \left\| \nabla_{\overline{x}} \mathcal{H}_\theta^1(\overline{x}(t)) + \frac{\overline{v}(t+\Delta t) - \overline{v}(t-\Delta t)}{2\Delta t} \right\|_2$$

Once the HNN is trained, it is important to predict its dynamics with a symplectic scheme as a Störmer-Verlet scheme as described in section 2.3. You can see a first example on the success of the combination AE with HNN in section 5.1.

5 First results: overview and analysis

Let us analyse some results of the whole process AE with HNN. We create the dataset with a PIC algorithm. Then, we train the AE. At last, we train the HNN with AE outputs. This section is organised as follow: we show the PSD faulty behaviour on a test case and a solution provided by an MAE with a HNN. Then, we search for the most efficient autoencoder framework with its architecture. At last, we look at some caveats. In addition to numerical errors, we look out the number of parameters and the speed of execution.

Thus, we focus on confined particles with Dirichlet boundary conditions. The electric confinement field E_{ext} is symmetric and has two parameters: the free space $L_{free} \leqslant L$ is the size of the domain L where the confinement field is null and the confinement intensity I on the borders. E_{ext} is:

$$E_{ext}(x) = \begin{cases} I\left(x + \frac{L_{free}}{2}\right)^3 & \text{if } x < -\frac{L_{free}}{2}, \\ 0 & \text{if } -\frac{L_{free}}{2} < x < \frac{L_{free}}{2}, \\ I\left(x - \frac{L_{free}}{2}\right)^3 & \text{else.} \end{cases}$$

We observe this confinement on figure 12. As we work with $q/m = -1$ i.e. anions such as electrons and knowing the acceleration due to the confinement is $-E_{ext}$, particles close to the left border are pushed to the right and conversely for the right border.

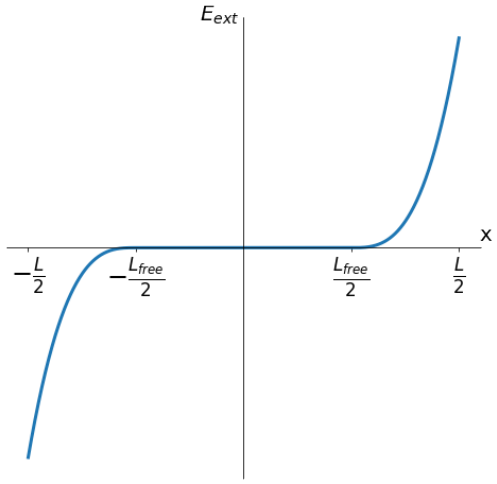


Figure 12: Confinement field

5.1 PSD faulty behaviour solved by our data driven process

We show that our process AE with HNN improves reduction performance over a PSD on a test case with a self-consistent electric field.

The physical system is composed of $N = 1000$ charged particles on a domain $[-1, 1]$ submitted to a self-consistent electric field and confined with a powerful external electric field. This confinement field is characterised by $L_{free} = 1.2, I = 1600$. Initial positions follow a uniform distribution on $[-0.5, 0.5]$ and initial velocities follow a standard normal distribution. We proceed to a simulation on a time $t = 5.0$ with a time-step $\Delta t = 0.001$. On figure 13, we observe trajectories of each particle in phase space. This constitutes our reference solution.

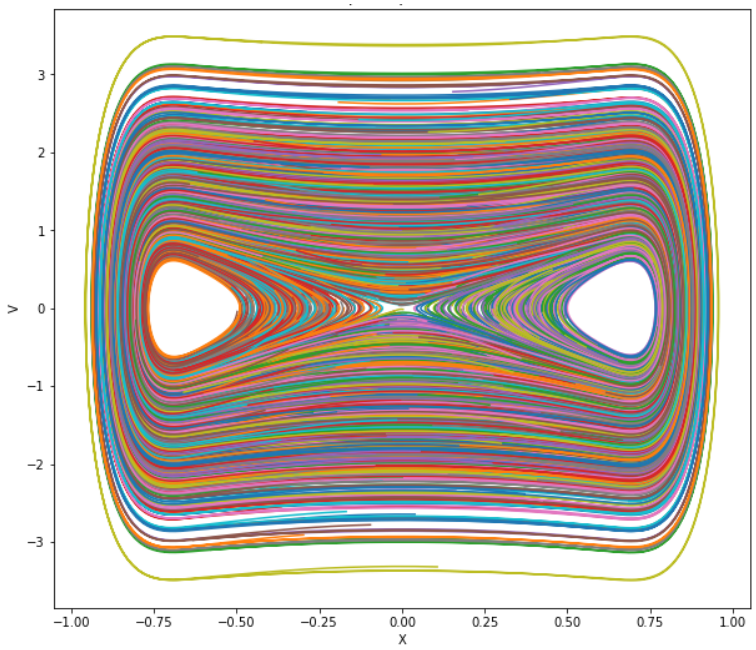


Figure 13: Reference Phase portrait of the dynamics given by PIC

Then, we perform the PSD using every time step as input. As recalled in annex A.3, the PSD relies on a Singular Value Decomposition (SVD): we look at singular values magnitudes in descending order to find the correct dimension K i.e. the number of singular vectors relevant to describe the data. Indeed, a singular value magnitude gives an idea of the importance of the corresponding singular vector. On figure 14, we observe that $K = 35$ (the 35^{th} singular value is represented with a red triangle) is more than sufficient to describe the dynamics.

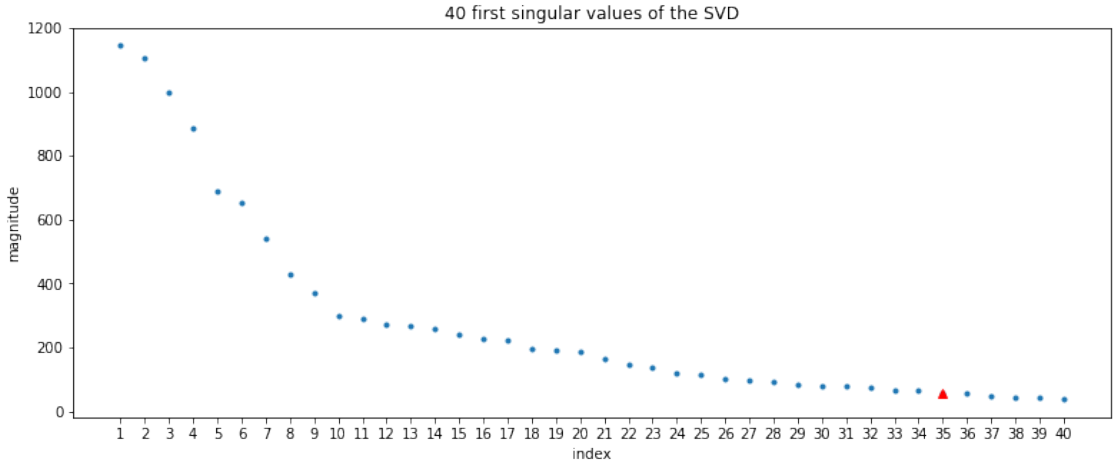


Figure 14: Singular values magnitudes in descending order

We build the matrix A. To illustrate its behaviour on non-linear dynamics, we compare two results with the reference from figure 13. First, we operate the encoder's identity: we compress every trajectory in the reduced space and back i.e. what should be the identity operation. We call it the PSD's identity for the corresponding case. Then, we simulate the reduced model with the same initial condition and decompress its reduced trajectories. In both cases, we except the resulting phase portrait to be as close to the reference as possible. Thus, with $K = 35$, we expect indeed a very efficient reduction.

We observe these results on figure 15. On the left, the PSD's identity fails to reproduce reference trajectories. Velocities are too elevated next to both borders and trajectories are oscillating. At least, the confinement is respected. On the right, the reduced model simulation gives worst results: the confinement is broken, velocities are almost three times more elevated than the reference and the two symmetrical areas without particles are not preserved.

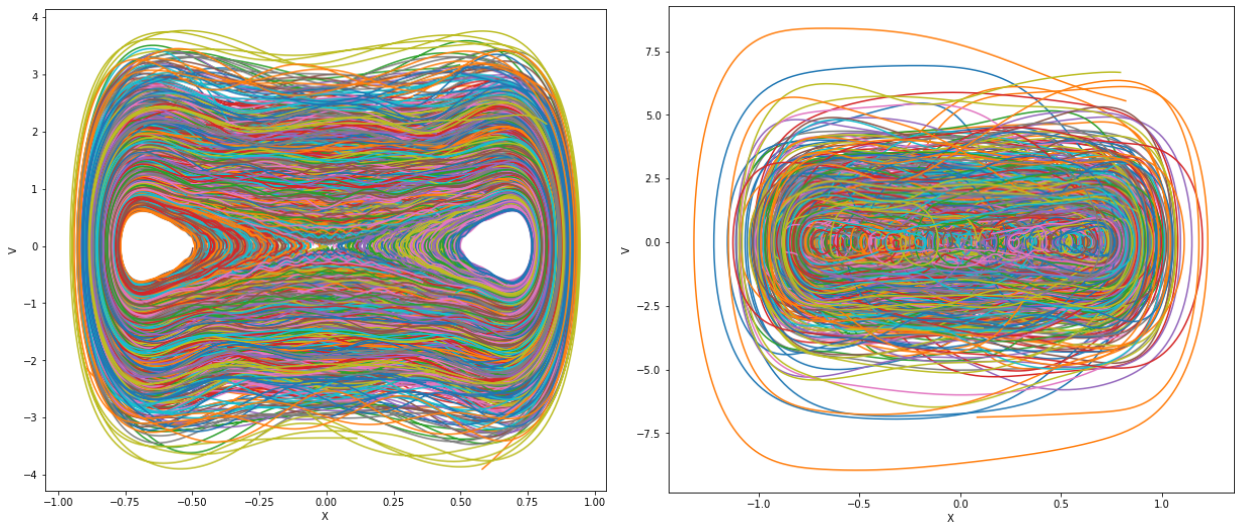


Figure 15: Phase portraits of the PSD's identity (left) and the PSD reduced model decompressed simulation (right)

Remark. *Providing a detailed analysis of the PSD is not part of this report. We consider it as a first result of reference for data driven techniques.*

You can find more details concerning the drawing of initial conditions in annex A.5.

Let us see if we can improve results from the PSD. We use a MAE. Whilst we used $K = 35$ to keep most of the information in the PSD, we use $K = 8$ for the MAE. The encoder structure is composed of hidden dense layers of units $(100, 100, 10, 10, 8, 8)$ and inversely for the decoder. We observe the reconstructed phase portrait on figure 16. As we can see, the reconstruction is much better with a smaller K. We will provide more quantitative results in the next section 5.2.

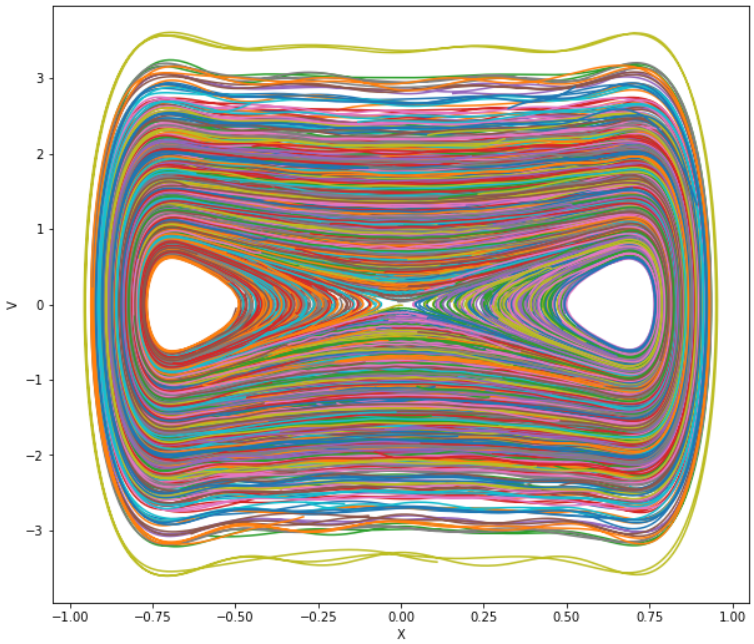


Figure 16: Phase portrait of the MAE encoder's identity

Then, we finish this model order reduction with an HNN. It is composed by 3 hidden layers of 300 units each. Let us observe the result on figure 17. On the left we remind the reference phase portrait. On the right we observe the resulting phase portrait of the three steps process: the compression of initial conditions with a MAE, the learned reduced model with an HNN prediction and the predicted trajectories decompression. The result is close from the MAE's identity above-seen on figure 16: the reduced dynamics is learned. There is a small downside for highest velocities trajectories. It is due to the lack of learned examples in this part of the phase space. As previously, we will provide more quantitative results in the next section 5.2.

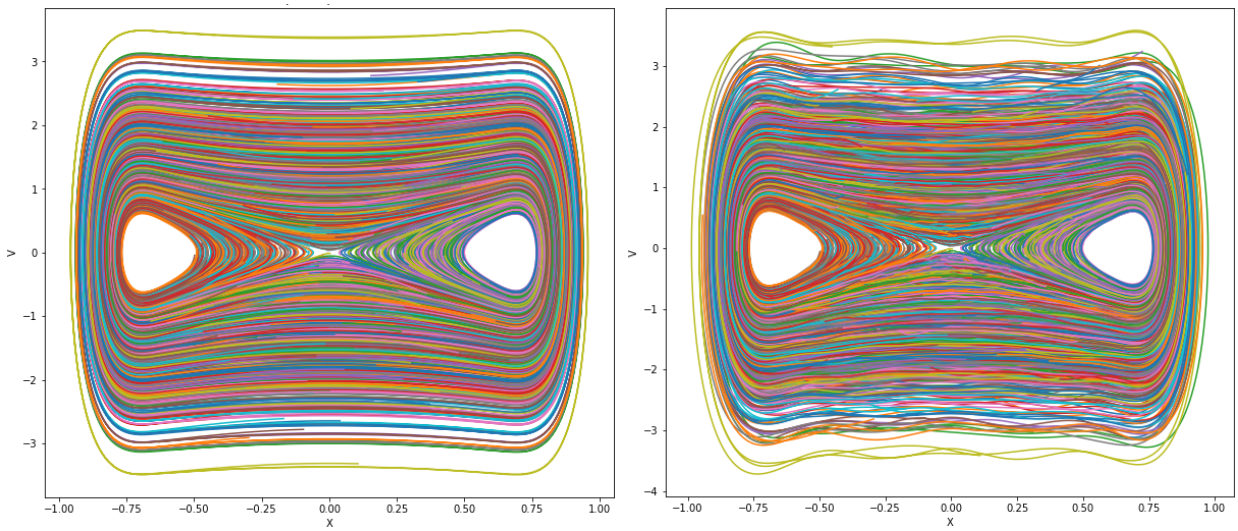


Figure 17: Phase portraits of reference (left) and the HNN reduced model decompressed simulation (right)

5.2 First results on autoencoders coupled with hamiltonian neural networks

We start our analysis with a system composed of $N = 10^3$ particles. We study the different autoencoder frameworks to choose one. Subsequently, we focus on long time simulation. At last, we discuss some limitations.

We set $L = 2, L_{free} = 1.2, I = 1600, K = 7$. The simulation lasts $T = 2$ with $\Delta t = 5 \times 10^{-4}$. The initial condition follows an uniform distribution over $[-0.5, 0.5]$ on positions and a standard normal distribution on velocities.

We compare six autoencoders: AE, MAE, SAE, LAE, SLAE and MLAE coupled with an HNN made of 3 hidden layers of 300 units each that learns the hamiltonian gradient. We describe dense architectures by listing their number of units per layer. We use a rough decrease with $b = 9$ (see annex A.5.2) and repeat each layer two times. It leads to (222,222,24,24,14,14) for the AE and (111,111,12,12,7,7) for MAE and SAE. For light architectures, we list pairs as described in section 3.3. For the LAE, we use ((200,180),(100,90),(108,80),(30,20),(20,10),(20,10),(20,10)) and for SLAE and MLAE we use ((200,180),(50,40),(45,40),(20,10),(20,10),(10,8)). We observe the number of parameters on figure 18:

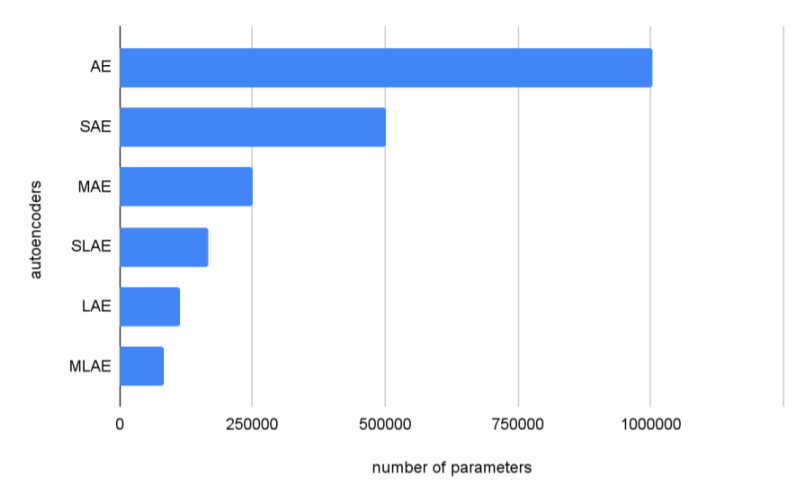


Figure 18: Number of parameters per autoencoder

We aim to compare the number of parameters with equivalent performance. As light architectures construction is much more experimental, results are more nuanced. As expected, the heaviest is the AE, its variations SAE and MAE split its weight by two and four, respectively. Light architectures are the lightest.

The objective is not to compare autoencoders of the same number of parameters as reducing them is a challenge. We have found light enough architectures to obtain correct examples.

On figure 19, we observe on the top sub-figure the MSE (Mean Squared Error) between the reference trajectories given by PIC and both the encoder's identity (a compression followed by decompression of trajectories) and the HNN decompressed prediction (HNN predicts reduced trajectories; we decompress them in order to compare them to the reference) on the bottom sub-figure.

We split dense and light architectures results on following figures. Concerning encoders identities, all autoencoders are working. We note that light architectures produce more stable results. In contrary, dense architectures results are peakier. On HNNs predictions, the dynamics is learned.

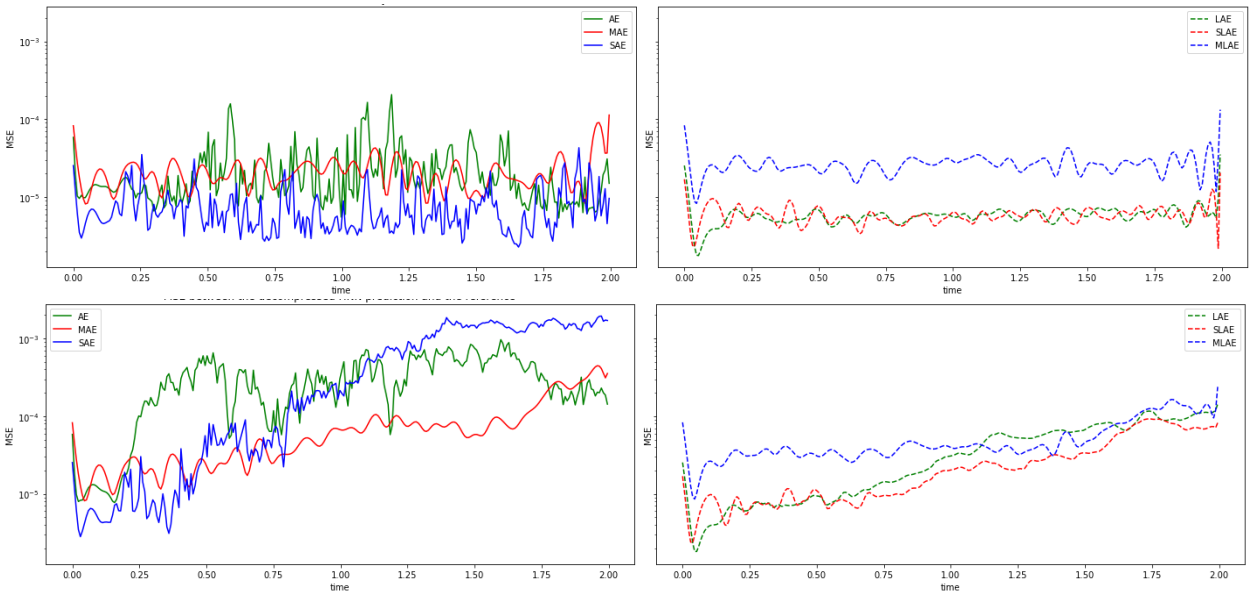


Figure 19: MSE between the reference and the encoder's identity (up) and between the reference and the decompressed HNN prediction (down)

We abandon the MLAE. Indeed, as explained in section 3.4, jacobian computations are very extensive and slow down both training and predictions in comparison to other solutions.

We increase T and restart training with new data in order to find significant differences between the different reductions. We set $T = 6, \Delta t = 10^{-3}$. Let us observe MSEs again on figure 20.

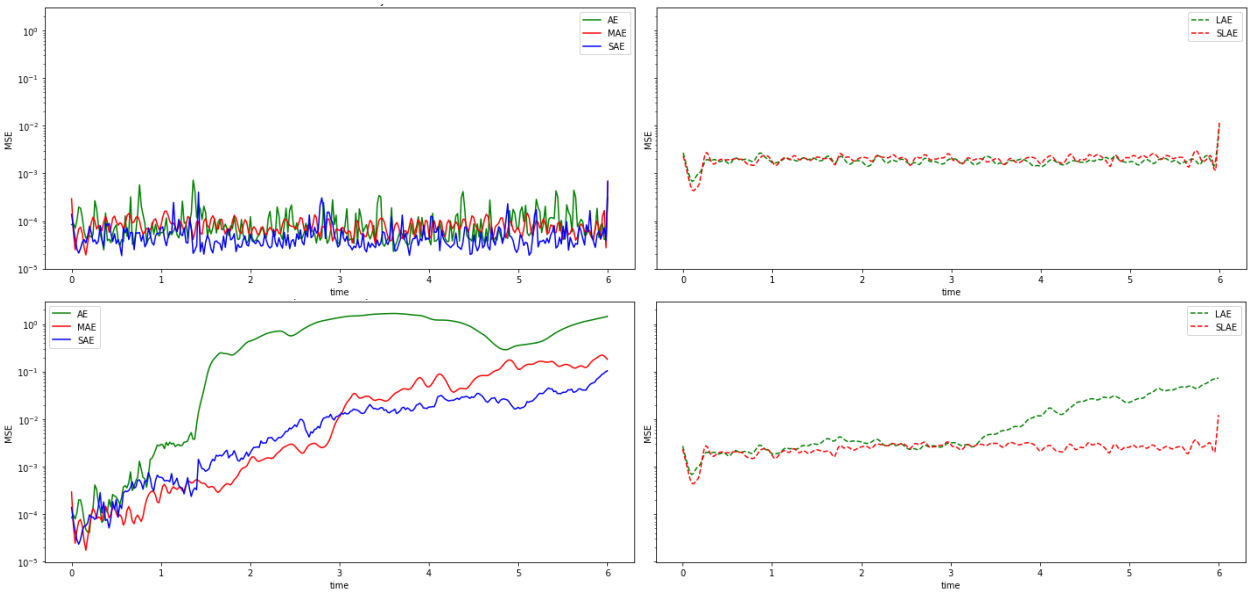


Figure 20: MSE between the reference and the encoder's identity (up) and between the reference and the decompressed HNN prediction (down)

We start with dense architectures. Looking at encoders identities, conclusions are the same. For HNNs predictions, the first autoencoder to fail is the AE: predictions deviates greatly from the reference when $T > 1.5$. Other results are also deteriorated. We observe a particle trajectory prediction with these three autoencoders on figure 21. We observe positions on the top and velocities on the bottom. The rows depicts, from left to right, AE, MAE and SAE predictions. HNNs predictions from the AE compression is the worst, then comes the MAE, and at last the SAE is the best.

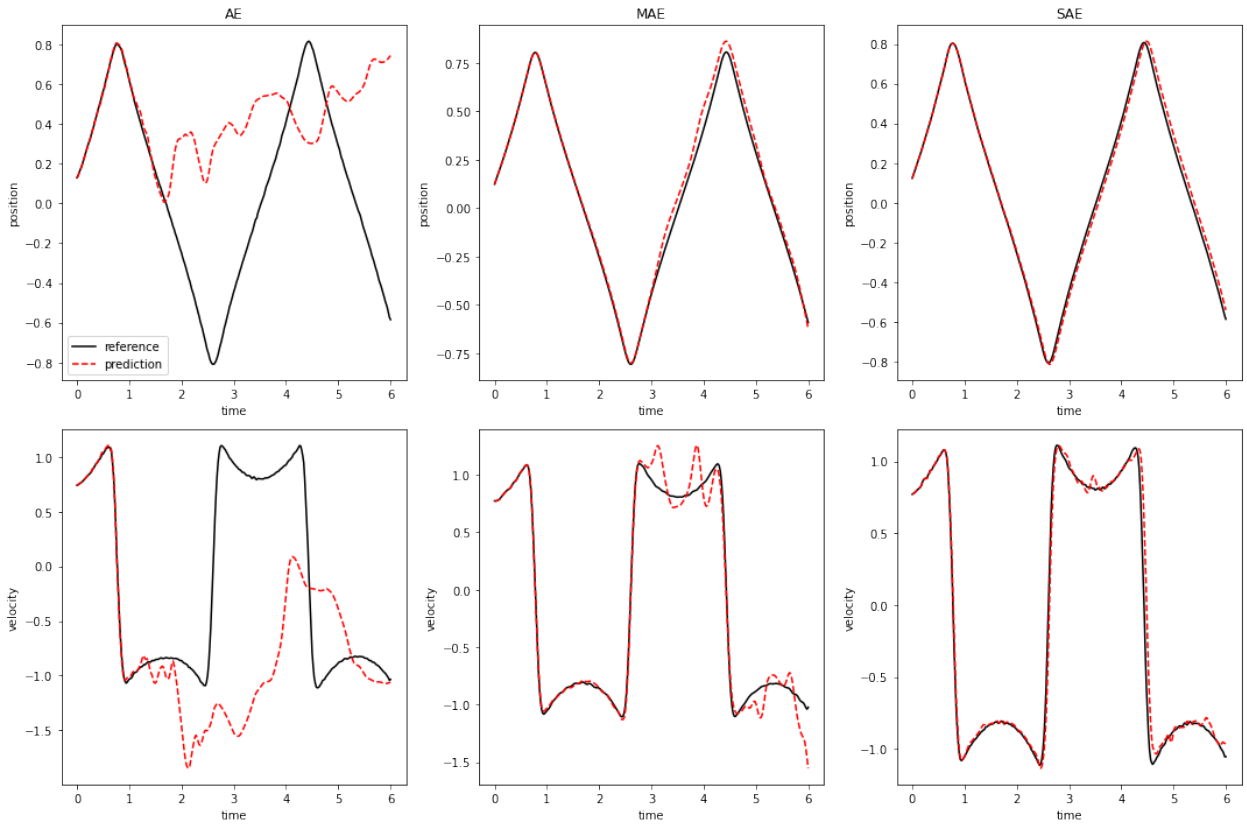


Figure 21: A particle position (up) and velocity (down) and its prediction with different autoencoders (one per column)

Returning to figure 20 analysis on light architectures, encoders identities are both working well. Nonetheless, HNN predictions with the SLAE are more precise than with the LAE. Indeed, the prior deviates from the reference when $T > 3$. The latter is working well.

We noticed that light architectures, separated and mono-block frameworks provides smoother errors. It might give us a beginning of explanation. We remind that an autoencoder sole goal is to represent the identity without any control on the intermediate representation which is none other than reduced variables. Given that the HNN learns from this reduction, the quality of the latter is important. We observe some phase portraits of reduced variables on figure 22. A column corresponds to an autoencoder. A line corresponds to a different reduced particle. We pick some reduced variables trajectories given by the autoencoder (in black) and we compare them to their HNN predictions (in red).

We notice that AE reduced trajectories are very twisted. In contrary, the other reduced trajectories are much smoother.It is probably why the HNN struggles to learn the prior.

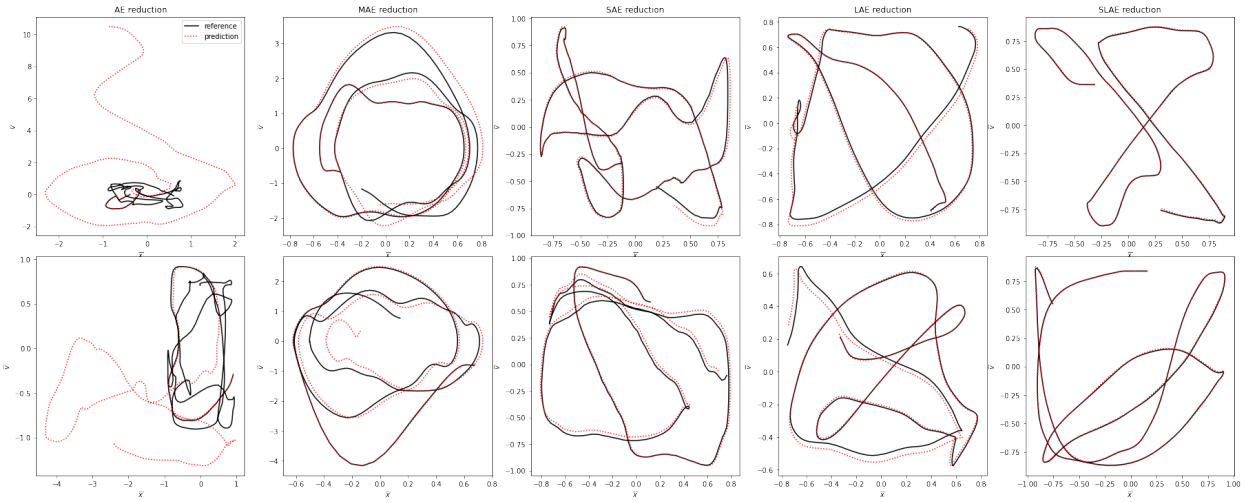


Figure 22: Some reduced trajectories from AE, MAE, SAE, LAE, SLAE (left to right)

Concerning separated frameworks, it comes from the fact that positions and velocities are treated by two independent networks. As for the mono-block framework, the latter treatment is the same. Thus, its loss takes into account the model and its jacobians, so parameters are fitted according to both. Regarding light architectures, it might come from the fact that there are fewer connections between units hence an output unit is submitted to fewer hidden units outputs.

Remark. *Autoencoders is part of the unsupervised machine learning, this is what makes it powerful for its ability to treat various problems, but we do not control the coded data quality. It would be beneficial to be able to add a smoothness criterion over reduced variables.*

Drawing an intermediate conclusion, we abandon classic frameworks. Going further, we want to increase N. Hence, we discontinue the use of dense architectures that uses too many parameters. Indeed, we saw that light architectures, with far fewer parameters, provides equivalent or better results. From now on, we focus on SLAE models.

In these examples, we reduced a system of dimension $2N = 2000$ into a reduced system of dimension $2K = 14$: the dimensions ratio is about 130: it is satisfactory. It leads to the fact that the speed of execution of our process is about 25 times faster than the PIC simulation. Before proceeding to a parametrised initial conditions, we would like to push the reduction further.

Our third system is composed of $N = 10^4$ particles. We leave unchanged other physical parameters. We set $K = 14$ for a reduction ratio of 700. We set $T = 2, \Delta t = 10^{-3}$. We use an SLAE of structure (1000,800),(400,200),(400,200),(50,20),(50,20),(10,5). The HNN has a structure (500,500,500) and it learns an hamiltonian gradient.

On the left of figure 23, we observe the MSE between the reference and the encoder's identity in green, and between the reference and the decompressed HNN prediction. The dynamics is well learned. Indeed, we can see on the right of the figure the MSE between the compressed reference and the HNN prediction. We observe that the encoder's identity remains below 10^{-5}. Compressed and decompressed HNN's prediction is bounded by 6×10^{-5}.

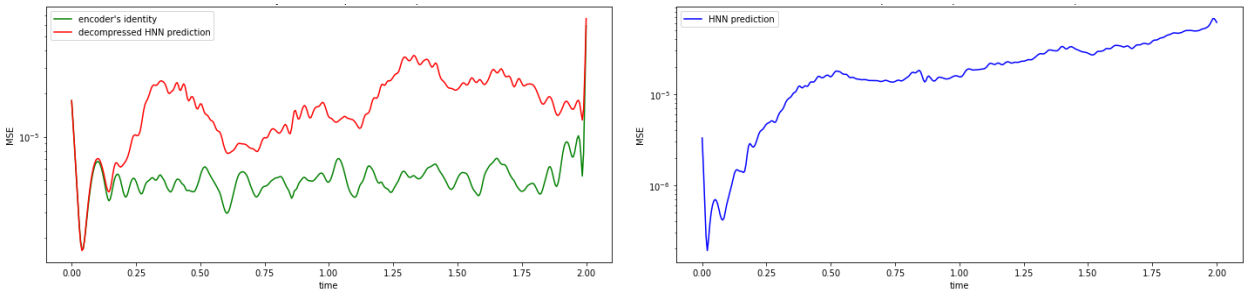


Figure 23: MSE between the encoder's identity, the decompressed HNN prediction and the reference (left) and MSE between the HNN prediction and the compressed reference (right)

Even though the reduction ratio is very large, the dynamics is well learned. We keep the SLAE as our reference autoencoder.

To resume, our Model Order Reduction (MOR) method is functioning. We have chosen an efficient autoencoder that provides satisfactory results. It remains efficient with a greater number of particles and stable in a long time simulation.

5.3 Some caveats

We end this section with some limitations. To begin with, our process does not work when we permute particles nor when we draw a new random initial condition. In addition, we struggle to take into account spatial periodic boundary conditions. Let us show it on a simple example.

The physical system is composed of $N = 1000$ particles submitted only to an external electric field is $E_{ext}(x) = 3\cos(6\pi x)$. The periodic domain is $[-0.5, 0.5]$. The initial distribution follows an uniform distribution over $[-0.5, 0.3]$ for positions and a normal distribution of mean 0.5 and standard deviation 1.0. We use 3 hidden layers of 200 units each.

We augment inputs according to annex A.5.3. We choose to augment only positions and a SAE for reduction. The position encoder has a successive number of units (250,250,31,31,7,7), the velocity encoder has (125,125,15,15,7,7) and decoders have symmetric architectures. The SAE loss is calculated according augmented data.

Let us look at some trajectories prediction on figure 24. Each column corresponds to a particle, first row plots are positions as a function of time and second row plots are velocities as a function of time. Encoder's identity prediction (green) is superposed to reference trajectories (black). In contrary, HNN prediction is not satisfactory at all.

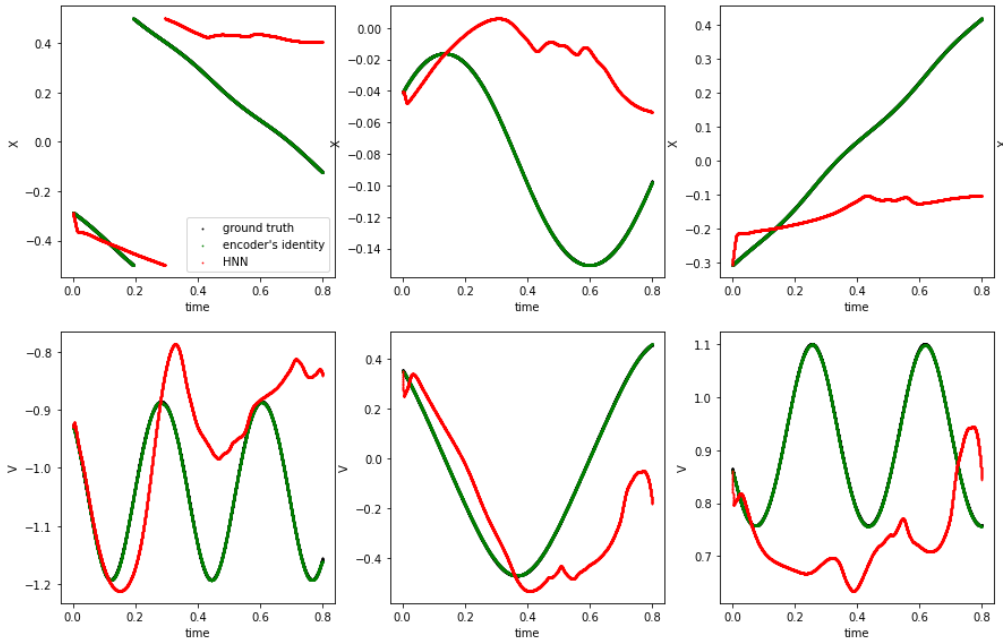


Figure 24: Predictions of positions (up) and velocities (down) as a function of time

In fact, the HNN does not succeed in learning reduced augmented data, as we can see on the loss graph as a function of the number of training steps on figure 25. In blue, we observe losses on the train set and in red from the validation set. It is clear that our process does not work on periodic data yet.

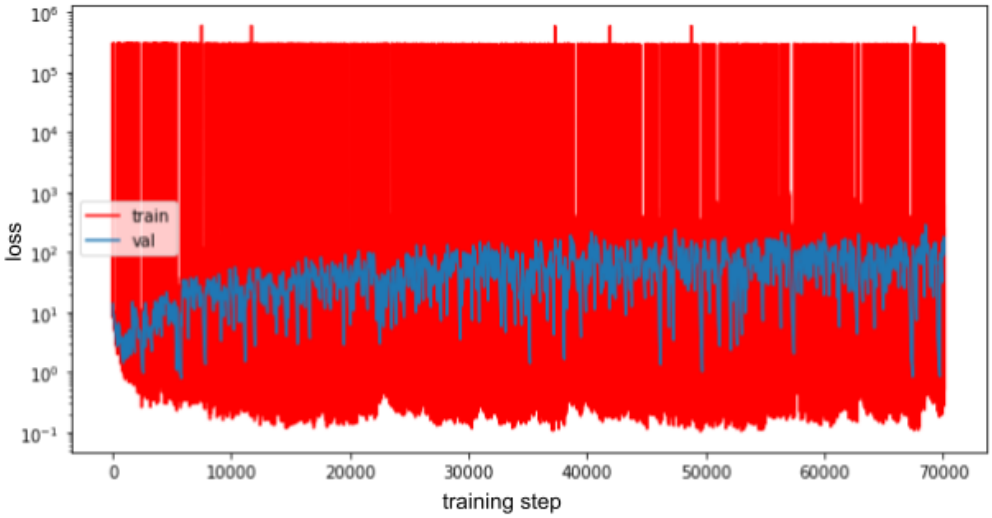


Figure 25: HNN training

Working on different augmentations or with a heavier HNN does not improve results at all.

6 Main results: model order reduction on a parametrised initial condition

We test our process on an important element of reduced order modelling: the parametrisation. Defining some quantities as parameters and training our reduced model with a range of these parameters, we expect the model to work with other parameters in this range. Time is already a parameter. We focus on the initial distribution. We keep a standard normal distribution on velocities and parametrise the position distribution: we use a beta distribution $B(\alpha, \beta)$ translated on $[-0.4, 0.4]$ with α, β its two parameters. We set $\beta = 1.5$ and use α as the initial condition parameter. α values used are $\{2.2, 2.6, 3.0, 3.4, 3.8\}$. We consider $N = 1000$ particles, $T = 2, \Delta t = 2.5 \times 10^{-4}$ and a confinement $L_{free} = 1.2, I = 1600$ on a domain of size $L = 2$. We look at a particle trajectory with the different α values used in the dataset on figure 26: they are sufficiently different.

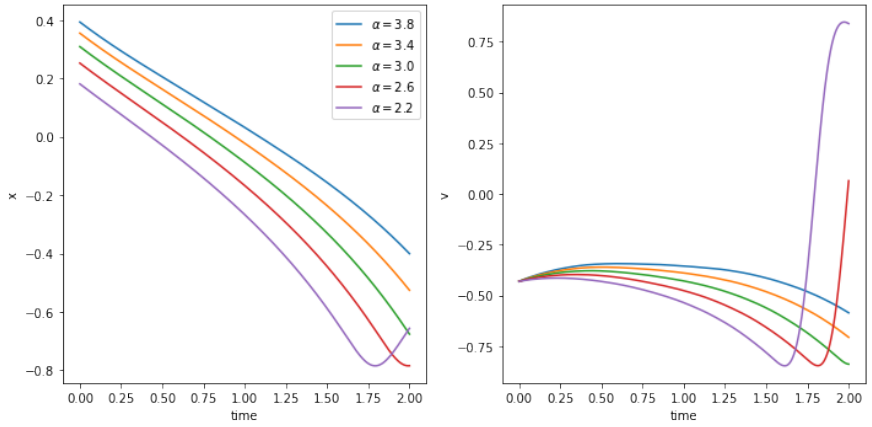


Figure 26: A particle position (left) and velocity (right) as a function of time for the different values of α

We set $K = 7$ (the dimensions ratio is 130). We keep the same physical parameters as above. Each encoder from the SLAE has a structure ((200,180),(50,40),(45,40),(20,10),(20,10),(10,8)) (about 80 000 parameters) and the HNN has 3 hidden layers of 400 units each. It learns the hamiltonian gradient.

Let us observe the usual MSEs as a function of time on figure 27. We see the MSE between the encoder's identity and the reference on the left and between the HNN prediction and the reference, for different α. We expect to see correct results for $\alpha \in [2.2, 3.8]$. It is indeed the case: the reference (black) is calculated from $\alpha = 3.0$ which belongs to the training set. $\alpha = 2.8, 2.9$ (orange, green) do not belong to the training set and results are almost identical to $\alpha = 3.0$ and bounded by an error of almost 10^{-3}. It can be seen that results deteriorate as α moves away from $[2.2, 3.8]$ with $\alpha = 4.2, 4.8$ (blue, red). Errors are about 10^{-2} and 10^{-1} respectively. Conclusions are the same for both the encoder's identity and HNN predictions.

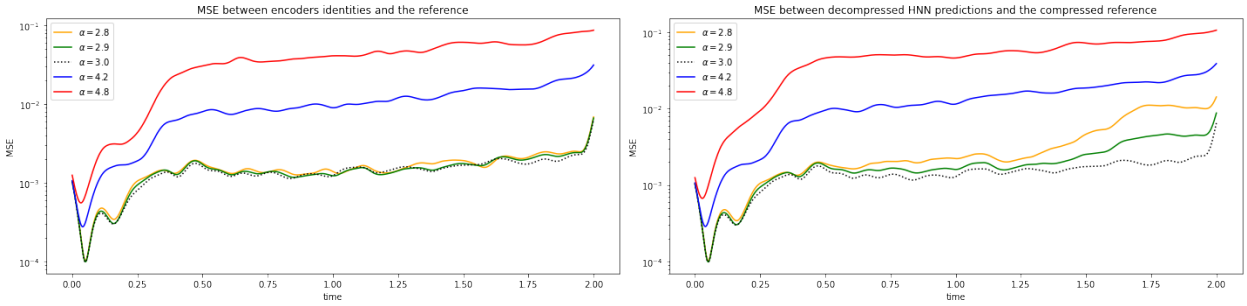


Figure 27: MSE between the encoder's identity and the reference (left) and MSE between the HNN prediction and the compressed reference (right) for different α

Let us observe the total MSE of a prediction for $\alpha \in [1, 5]$ on figure 28. We added dashed lines at $\alpha \in \{2.2, 2.6, 3.0, 3.4, 3.8\}$ used during the training, they correspond to errors minima. Error remains bounded by 10^{-2} on $[2.2, 3.8]$ and it is deteriorating as α moves away from this interval.

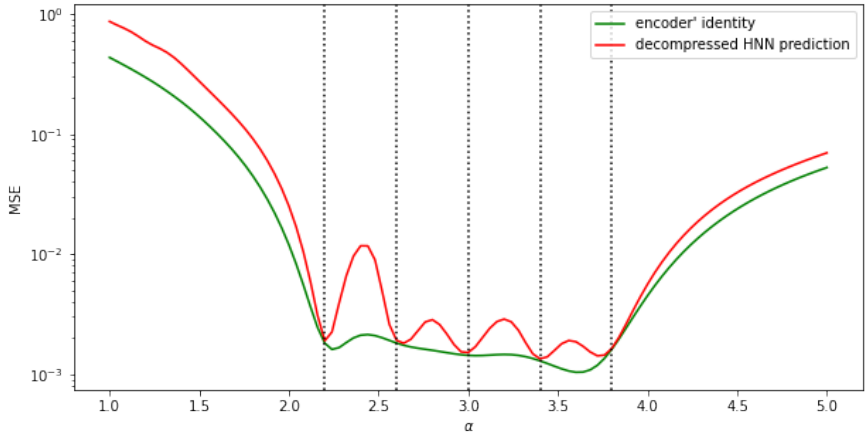


Figure 28: MSE as a function of α for the encoder's identity and the decompressed HNN prediction

To finish, we can plot on figure 29 the histogram of the absolute differences between the reference solution and the HNN prediction at different times. We note (x_{pred}, v_{pred}) the latter. It is an error histogram. On axes, we spot the value of the difference, the warmer the colour of the pixel is, the more that difference value has been observed. We note that predictions are more precise on positions. Indeed, errors are bounded by 0.24 and most of them are bounded by 0.75 (red and orange area). In contrary, velocities are bounded by 1.45 and most of them by 0.15. In addition, warm pixels spread out as time increase: predictions become less precise.

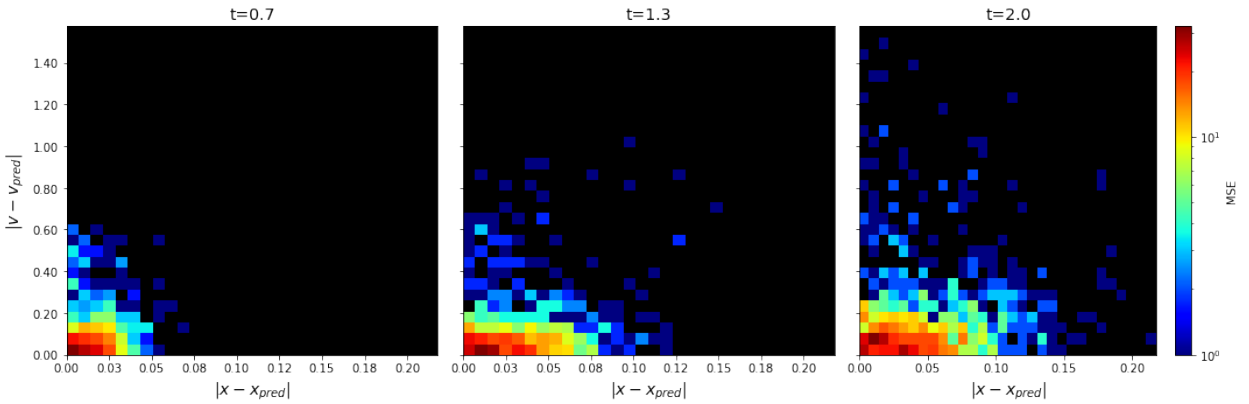


Figure 29: Histograms of absolute differences between the reference solution and the HNN prediction at time $t = 0.7$ (left), $t = 1.3$ (middle) and $t = T = 2.0$ (right)

In conclusion, our MOR based on autoencoders with hamiltonian neural networks if functioning well. We are able to make precise predictions with varying parameters. The reduced system remains stable in long time simulation and the process is scalable on a large number of particles. Nonetheless, light autoencoders construction remains difficult. For instance, we could obtain a great autoencoder with a validation loss of 10^{-6} that does not provide good reduced variables and results in mediocre HNN performance, especially with varying initial condition's parameter. In contrary, the autoencoder used above has a validation loss of 10^{-3} showed a great performance. Thus, the choice of the reduction ratio is arduous: it has to maintain a balance. Indeed, a smaller K facilitate the HNN learning. In contrast, a larger K permits a compression-decompression process of greater quality with less information loss.

7 Conclusion

In conclusion, we develop a full data driven method to reduce the Vlasov-Poisson equation. The core difficulty of this model is that a particle motion depends on every other particle position, which implies genuinely non linear dynamics. Our goal is to find a reduced model that follows three criteria: it has to be true to the original in a range of parameters, efficient with a large number of particles and stable in long time simulations.

In order to satisfy the third criterion, we rely on Hamiltonian mechanics; it is a mathematical and abstract mechanics formalism that permits to describe various physical systems in a general manner. In fact, we work on hamiltonian systems: their energy is conserved. As a consequence, we can describe them using the above-mentioned formalism with the symplectic formulation. Such symplectic structure preserves the hamiltonian. Preserving this structure at a numerical level with so-called symplectic schemes grant the stability of our methods.

Concerning this internship, we develop a PIC algorithm with a symplectic scheme to accurately simulate a Vlasov-Poisson dynamics and generate a dataset for our data driven Model Order Reduction (MOR). Then, we propose to use a powerful neural network called an autoencoder (AE) to reduce physical variables into low dimension reduced variables. We show on several examples that it is an efficient non-linear reduction fit to capture this strongly non-linear dynamics with self-induced electric fields. Main drawbacks of AEs is that there is no straightforward way to exhibit the reduced model satisfied by the reduced variables nor is there any guarantee that the reduced variables satisfy a symplectic or hamiltonian system. The latter is a work hypothesis that is made to assure numerical properties. Therefore, we prefer to learn the reduced model with an hamiltonian neural network (HNN) that safeguard the reduced model is symplectic. The combined use of an AE and a HNN form our MOR.

Subsequently, we design several AEs and study them coupled with HNNs on different confined particles systems. The most efficient AE is the separated one with a light architecture: the SLAE. It is a lightweight parameter-wise that permit a great scalability with a large number of particles. In addition, its reduced variables have a great quality: the reduced dynamics is well learned with an HNN. The latter performs well learning the hamiltonian or its gradient. The whole fulfil the three criteria above-mentioned. Thus, the reduction ratio is satisfactory and we showed good results for a ratio up to 700. Going to the heart of MOR, we showed a great learning capacity with parametrised initial conditions. The process can generalise results from its training dataset.

Nonetheless, some results are not convincing. Indeed, we cannot swap particles although indistinguishable in theory. The HNN also struggles on periodic systems. Finally, light architectures construction remains experimental and we lack of tools to discriminate different reduced variables. It could lead to reduced variables of poor quality and an HNN that encounter difficulties. There are a few leads to solve some of these problems: working with larger datasets with more various initial conditions and coupling the training of AEs and HNNs. There also exists some neural networks that allows to swap particles. Most and foremost, developing a symplectic autoencoder would be a breakthrough.

Going forward, it would be interesting to work on a 3D Vlasov model with magnetic fields. The PIC algorithm is up to the task and we it would give more applicability to our work. Thus, this kind of model not only apply to plasma but also to other fields as nanotechnologies. This process could also be extended to other kinetic models.

A Technical appendices

A.1 Origin of the Vlasov-Poisson equation

We consider a plasma composed of a large number of particles described by a statistical distribution $f(\mathbf{x}, \mathbf{v}, t))$. These particles are charged ergo they can be subject to an electromagnetic (electric $\mathbf{E}$ and magnetic $\mathbf{B}$) field. The latter can be self-consistent i.e. generated by the ions themselves and their motion or/and external. To describe it, we use Vlasov-Maxwell equations: it is composed by the Vlasov equation coupled with Maxwell's equations, as described in [1].

Definition A.1 (Vlasov equation)**.** *Let f be a distribution of non-relativistic particles, the collisionless Vlasov equation is written:*

$$\partial_t f + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + \frac{q}{m} (\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f = 0, \tag{6}$$

where $q \in \mathbb{R}$ is the particles charge, $m > 0$ their mass and $f(\mathbf{x}, \mathbf{v}, t)$ represents the particles distribution, position $\mathbf{x} \in \mathbb{R}^d$ and velocity $\mathbf{v} \in \mathbb{R}^d$ depend on time $t \in [0, +\infty)$ and $d \in \{1, 2, 3\}$ the dimension of the system.

The self-consistent electromagnetic fields $(\mathbf{E}, \mathbf{B})$, as it depends on f, can be calculated using Maxwell's equations with sources. We do not make any distinction with an external field as it can be defined the same way.

Definition A.2 (Maxwell's equations with sources)**.**

$$\begin{cases} \nabla \cdot \mathbf{E} = \dfrac{\rho}{\epsilon_0} & (7) \\ \nabla \times \mathbf{E} = -\partial_t \mathbf{B}, & (8) \\ \nabla \cdot \mathbf{B} = 0, & (9) \\ \nabla \times \mathbf{B} = \dfrac{1}{c^2} \partial_t \mathbf{E} + \mu_0 \mathbf{J}, & (10) \end{cases}$$

where $\rho(\mathbf{x}, t) = q \int f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v}$ is the total electric charge density, $\mathbf{J}(\mathbf{x}, t) = q \int f(\mathbf{x}, \mathbf{v}, t) \mathbf{v} d\mathbf{v}$ is the total electric current density, ϵ_0 is the permittivity of free space, μ_0 is the permeability of free space and $c = (\epsilon_0 \mu_0)^{-1/2}$ is the speed of light.

We are more interested in the electric limit of a plasma i.e. $|\mathbf{E}| >> c\,|\mathbf{B}|$. By relying on article [9], we obtain, at zeroth order, the equations $\nabla \times \mathbf{B} = \nabla \cdot \mathbf{B} = 0$. Then, we apply a particular case of Helmholtz-Hodge theorem [10] to demonstrate $\mathbf{B} = 0$ at the zeroth order.

Theorem A.3 (Helmholtz-Hodge theorem)**.** *Let $\mathbf{F} \in \mathcal{C}^2(\mathbb{R}^3, \mathbb{R}^3)$ be a vector field that vanishes faster than $1/|\mathbf{x}|$ as $|\mathbf{x}| \to \infty$ then $\mathbf{F}$ can be decomposed into a curl-free component and a divergence-free component:*

$$\mathbf{F} = -\nabla \phi_{\mathbf{F}} + \nabla \times \mathbf{A}_{\mathbf{F}}$$

where:

$$\phi_{\mathbf{F}}(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{\nabla' \cdot \mathbf{F}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}',$$

$$\mathbf{A}_{\mathbf{F}}(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{\nabla' \times \mathbf{F}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}'.$$

$\phi_{\mathbf{F}}$ is referred to as a scalar potential and $\mathbf{A}_{\mathbf{F}}$ as a vector potential.

Remark. *A more general theorem does not make any hypothesis on the limit of $\mathbf{F}$ when $|\mathbf{x}| \to \infty$, it results in an additional term in the integral forms of both potentials.*

Remark. *This theorem is a bit brutal here. The point is to show that it is sufficient to know the curl and the divergence of a vector field to characterise it. Although potentials are not unique,* $\phi_{\mathbf{F}}$ *and* $\mathbf{A_F}$ *are unique. It permits to understand the structure of Maxwell's equations and why these are sufficient to describe an electromagnetic field.*

As $\mathbf{B} \in \mathcal{C}^2(\mathbb{R}^3, \mathbb{R}^3)$ is a Coulombian field i.e. in $1/|\mathbf{x}|^2$, we apply the decomposition A.3 to $\mathbf{B}$:

$$\phi_{\mathbf{B}}(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{\nabla' \cdot \mathbf{B}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}' = 0,$$

$$\mathbf{A}_{\mathbf{B}}(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{\nabla' \times \mathbf{B}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}' = 0.$$

As a forthright consequence $\mathbf{B} = 0$.

Then, we apply the same decomposition to $\mathbf{E}$ and use equations A.2 with $\mathbf{B} = 0$:

$$\phi_{\mathbf{E}}(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{\nabla' \cdot \mathbf{E}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}' = \frac{1}{4\pi\epsilon_0} \int_{\mathbb{R}^3} \frac{\rho(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}' := \phi(\mathbf{x}),$$

$$\mathbf{A}_{\mathbf{E}}(\mathbf{x}) = \frac{1}{4\pi} \int_{\mathbb{R}^3} \frac{\nabla' \times \mathbf{E}(\mathbf{x}')}{|\mathbf{x} - \mathbf{x}'|} d\mathbf{x}' = 0.$$

Hence, we obtain the electric potential ϕ such that $\mathbf{E} = -\nabla\phi$. Injected in Maxwell Gauss equation (7), we obtain Poisson equation $\Delta\phi = -\frac{\rho}{\epsilon_0}$. At last, we apply $\mathbf{B} = 0$ in definition 6 and obtain the Vlasov-Poisson equation 2.1.

Remark. *The electric potential* ϕ *is a great calculus intermediate to compute* $\mathbf{E}$*, indeed we manipulate a scalar field instead of a vector field. Thus,* ϕ *has a physical interpretation: we define the electric potential energy of a particle of charge* q*:* $q\phi$ *; the particle tends to minimise its energy.*

A.2 PIC field solver: finite differences, boundary conditions and matrix formulation

We start with the Poisson equation using a finite differences scheme as explained in [11]. We study several boundary conditions and their precise formulation: homogeneous Dirichlet, homogeneous Neumann and periodic. Then, we formulate the computation of $E = -\nabla\phi$.

Let $[0, L]$ be the space domain partitioned into $N_x + 1$ cells of size $\Delta x = \frac{L}{N_x}$. Each node has a charge $\rho(i\Delta x) = \rho_i, i \in \{0, \cdots, N_x\}$ and an electric potential ϕ_i.

Homogeneous Dirichlet boundary condition: We resolve the differential equation:

$$\begin{cases} -\Delta\phi = \rho\,, x \in [0, L], \\ \phi(0) = \phi(L) = 0. \end{cases}$$

With finite differences, we obtain a system of linear equations:

$$\begin{cases} \dfrac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2} = -\rho_i, i \in \{1, \cdots N_x - 1\} \\ \phi_0 = 0, \\ \phi_N = 0. \end{cases}$$

We look at neighbours to boundary's nodes: with $i = 1$:

$$\frac{1}{\Delta x^2}\left(2\phi_1 - \phi_2\right) = \rho_1,$$

with $i \in \{2, \cdots N - 2\}$:

$$\frac{1}{\Delta x^2}\left(-\phi_{i-1} + 2\phi_i - \phi_{i+1}\right) = \rho_i,$$

with $i = N - 1$:

$$\frac{1}{\Delta x^2}\left(-\phi_{N-2} + 2\phi_{N-1}\right) = \rho_{N-1}.$$

We obtain the matrix formulation of the linear system with $N_x - 1$ unknowns:

$$A\Phi = R$$

where:

$$A = \frac{1}{\Delta x^2}\begin{pmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{pmatrix}, \Phi = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_{N-1} \end{pmatrix}, R = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{N-1} \end{pmatrix}$$

Adding $\phi_0 = \phi_{N_x} = 0$: we solve the Poisson equation.

Homogeneous Neumann boundary condition: We resolve the differential equation:

$$\begin{cases} -\Delta\phi = \rho \ - \int_0^L \rho \, , x \in [0, L], \\ \phi'(0) = \phi'(L) = 0, \\ \phi(0) = 0. \end{cases}$$

We remark that $\phi(0) = 0$ is imposed to ensure the uniqueness of the solution. Thus, the right hand side is changed in order to ensure a solvability condition. In fact, if we would solve $-\Delta\phi = \rho$ we would obtain by integration:

$$\phi'(L) - \phi'(0) = \int_0^L \rho dx = 0.$$

Hence, this right hand side imposes $\int_0^L \rho dx = 0$. Here, we note $\rho_i = \rho(i\Delta x) - \int_0^L \rho$.

As in the previous paragraph, we obtain the linear system of $N_x - 2$ unknowns:

$$A = \frac{1}{\Delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 2 & -1 \\ 0 & \cdots & \cdots & 0 & -1 & 1 \end{pmatrix}, \Phi = \begin{pmatrix} \phi_2 \\ \vdots \\ \phi_{N-1} \end{pmatrix}, R = \begin{pmatrix} \rho_2 \\ \vdots \\ \rho_{N-1} \end{pmatrix}$$

Adding $\phi_0 = \phi_1 = 0, \phi_{N_x-1} = \phi_{N_x}$, we solve the Poisson equation.

Periodic boundary condition: We solve the differential equation:

$$\begin{cases} -\Delta\phi = \rho - \int_0^L \rho \, , x \in [0, L], \\ \phi(0) = \phi(L) = 0, \\ \phi'(0) = \phi'(L) \end{cases}$$

with the solvability condition and with ϕ L-periodic i.e. $\forall x \, , \phi(x) = \phi(x + L)$. Note that the first and the last node of the mesh are confounded.

Through the same calculation, we obtain the matrix formulation of $N_x - 2$ unknowns:

$$A = \frac{1}{\Delta x^2} \begin{pmatrix} 2 & -1 & 0 & \cdots & \cdots & 0 \\ -1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & \ddots & \ddots & \ddots & -1 \\ 1 & 0 & \cdots & 0 & -1 & 2 \end{pmatrix}, \Phi = \begin{pmatrix} \phi_1 \\ \vdots \\ \phi_{N-2} \end{pmatrix}, R = \begin{pmatrix} \rho_1 \\ \vdots \\ \rho_{N-2} \end{pmatrix}$$

Adding $\phi_0 = \phi_N = 0, \phi_{N-1} = -\phi_1$, we have the solution.

$\nabla\phi$ computation: We calculate $\nabla\phi$ on the grid with a matrix-vector multiplication:

$$\begin{pmatrix} \phi'_0 \\ \vdots \\ \phi'_{N_x} \end{pmatrix} = B \begin{pmatrix} \phi_0 \\ \vdots \\ \phi_{N_x} \end{pmatrix}$$

where $B \in \mathbb{R}^{(N_x+1)\times(N_x+1)}$ is yet to be determined. We distinguish two cases: whether $\phi(\cdot)$ is L-periodic or not. If it is not periodic, we use:

$$B = \frac{1}{2h} \begin{pmatrix} -2 & 2 & 0 & \cdots & \cdots & \cdots & 0 \\ -1 & 0 & 1 & \ddots & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & -1 & 0 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & -2 & 2 \end{pmatrix}$$

Otherwise, we use:

$$B = \frac{1}{2h} \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 & -1 & 0 \\ -1 & \ddots & \ddots & \ddots & & & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & 0 \\ 0 & & & \ddots & \ddots & \ddots & 1 \\ 0 & 1 & 0 & \cdots & 0 & -1 & 0 \end{pmatrix}$$

A.3 Classic reduction method: the Proper Symplectic Decomposition

To begin with, we want to find a linear relation $u = A\overline{u}$ with $A \in \mathbb{R}^{2N\times 2K}$ yet to be determined, as investigated on article [5]. Inspired by the Proper Orthogonal Decomposition (POD), they establish a symplectic version of it called proper symplectic decomposition (PSD).

During the numerical simulation, we save snapshots of the numeric solution at times $t_1, \cdots, t_M$. We consider the matrix $\Delta \in \mathbb{R}^{2N\times 2M}$ of snapshots:

$$\Delta = \begin{pmatrix} x(t_1) & \cdots & x(t_M) & v(t_1) & \cdots & v(t_M) \end{pmatrix}$$

We operate the singular value decomposition (SVD) of Δ.

Definition A.4 (Singular value decomposition)**.** *Let be $M \in \mathbb{R}^{m\times n}$. The singular value decomposition of M is:*

$$M = U\Sigma V^T$$

where $U \in \mathbb{R}^{m\times m}, U \in \mathbb{R}^{n\times n}$ are orthogonal matrix. The columns of U and V are called left-singular vectors and right-singular vectors, respectively. $\Sigma \in \mathbb{R}^{m\times n}$ is a rectangular diagonal matrix, $\sigma_i = \Sigma_{ii}$ are known has singular values values.
Provided that σ_i are sorted in a descending order, Σ is uniquely determined by M.

Remark. *SVD is a generalisation of the eigendecomposition to any rectangular matrix.*

We note the SVD $\Delta = U\Sigma V^T$. We keep the K first left-singular vectors $\Phi := U_{:,:K}$. It allows us to keep the K main features of the motion of u expressed in the snapshots.

As positions and velocities have been kept separately in Δ and by defining:

$$A := \begin{pmatrix} \Phi & 0 \\ 0 & \Phi \end{pmatrix} \tag{11}$$

we achieve to preserve the hamiltonian structure from definition 2.3, as said in [5].

Theorem A.5 (Proper Symplectic Decomposition (PSD))**.** *Let be an hamiltonian system, with $u \in \mathbb{R}^{2N}$ and its hamiltonian $\mathcal{H} : \mathbb{R}^{2N} \to \mathbb{R}$:*

$$\dot{u} = \mathbb{J}_{2N}\nabla_u \mathcal{H}(u)$$

We note $u = A\overline{u}$ where $\overline{u} \in \mathbb{R}^{2K}$ where $A \in \mathbb{R}^{2N\times 2K}$ is the matrix of equation (11) obtained with the K first left-singular vectors of the SVD of the snapshots matrix.

Then, we have the hamiltonian reduced system:

$$\dot{\overline{u}} = \mathbb{J}_{2K}\nabla_{\overline{u}}\tilde{\mathcal{H}}(\overline{u})$$

with $\tilde{\mathcal{H}}(\overline{u}) := \mathcal{H}(A\overline{u})$ is the reduced hamiltonian.
Furthermore, we define the symplectic inverse of A noted A^+ as $A^+ := \mathbb{J}_{2K}^T A^T \mathbb{J}_{2N}$. It is an inverse in the sense that $A^+A = \mathbb{I}_{2K}$. Then, we have the reciprocal transformation $\overline{u} = A^+u$.

Proof. We note $(A\overline{u})$ the linear application $\mathbb{R}^{2K} \to \mathbb{R}^{2N}$ defined by $\overline{u} \mapsto A\overline{u}$.

First, we want to find a relation between $\nabla_u \mathcal{H}(u)$ and $\nabla_{\overline{u}}\tilde{\mathcal{H}}(\overline{u})$, we use the chain-rule on $\mathcal{H}\circ(A\overline{u})$:

$$d_{\overline{u}}\left(\mathcal{H}\circ(A\overline{u})\right) = d_{A\overline{u}}\mathcal{H}\circ d_{\overline{u}}\left(A\overline{u}\right).$$

where we note $d_a f$ the differential of f in a. The Jacobian of f in a is written in the same way.

Hence, we obtain $d_{\overline{u}}\left(\mathcal{H}\circ(A\overline{u})\right) = \left[\nabla_{\overline{u}}\mathcal{H}(A\overline{u})\right]^T$, $d_{A\overline{u}}\mathcal{H} = \left[\nabla_u \mathcal{H}(u)\right]^T$ and $d_{\overline{u}}\left(A\overline{u}\right) = A$. Whence:

$$\left[\nabla_{\overline{u}}\mathcal{H}(A\overline{u})\right]^T = \left[\nabla_u \mathcal{H}(u)\right]^T A.$$

We apply a transposition and the symplectic inverse of A and ends up with the desired relation:

$$\left[A^+\right]^T \nabla_{\overline{u}} \tilde{\mathcal{H}}(\overline{u}) = \nabla_u \mathcal{H}(u).$$

Going back to the hamiltonian system:

$$\begin{aligned} &\frac{d}{dt}(A\overline{u}) = A\dot{\overline{u}} = \mathbb{J}_{2N} \nabla_u \mathcal{H}(u), \\ &\dot{\overline{u}} = A^+ \mathbb{J}_{2N} \nabla_u \mathcal{H}(u). \end{aligned}$$

Using the definition of the symplectic inverse:

$$A^+ \mathbb{J}_{2N} = -\mathbb{J}_{2K}^T A^T,$$

which leads to:

$$\dot{\overline{u}} = -\mathbb{J}_{2K}^T A^T \left[A^+\right]^T \nabla_{\overline{u}} \tilde{\mathcal{H}}(\overline{u}) = \mathbb{J}_{2K} \left[A^+ A\right]^T \nabla_{\overline{u}} \tilde{\mathcal{H}}(\overline{u}) = \mathbb{J}_{2K} \nabla_{\overline{u}} \tilde{\mathcal{H}}(\overline{u}).$$

□

All in all, we find a reduced model using the PSD. Thus, we have an explicit expression for the reduced model with $\tilde{\mathcal{H}}$ which allow us to use a symplectic scheme as in section 2.3 to compute the reduced dynamics.

A.4 Some failures of non symplectic numerical methods

In this annex, we show how important it is to use symplectic i.e. hamiltonian preserving schemes and Neural Networks (NN) on a simple example: the ideal mass-spring system or so-called harmonic oscillator. Results from this annex are taken from a previous project [7].

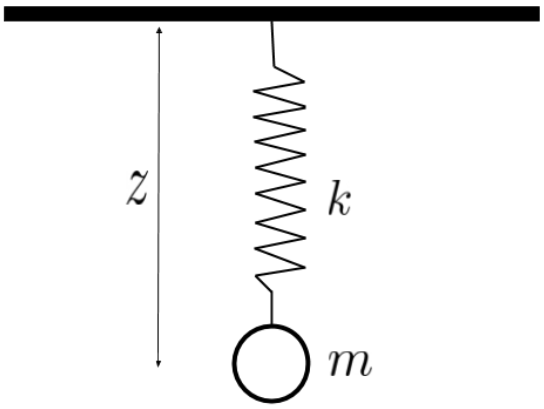


Figure 30: Ideal mass-spring system

As showed on figure 30, it is composed of a mass attached to a spring. At one end, the spring is attached to a fixed point, at the other end, it is attached to a mass. We note m the mass, k the stiffness constant of the spring and z its elongation. It is submitted to its own weight and the recoil force of the spring. We can demonstrate that, with a proper change of variable $q = (km)^{-1/4} z$ and $p = \dot{q}/\omega_0$ with ω_0 the undamped angular frequency $\omega_0 = \sqrt{k/m}$, that the hamiltonian is:

$$\mathcal{H}(q,p) = \frac{\omega_0}{2}(q^2 + p^2).$$

Therefore, using Hamilton's equations:

$$\begin{cases} \dot{q} = \omega_0 p, \\ \dot{p} = -\omega_0 q. \end{cases}$$

With the initial condition $(q,p)(0) = (1,0)$ and $\omega_0 = 1.6$, the phase portrait (q,p) is a circle of radius one and $\mathcal{H} = 0.8$ is constant.

Let us retrieve this dynamics with two schemes: an implicit eulerian scheme which is non symplectic and a symplectic eulerian scheme. As we observe results on figure 31, the implicit scheme does not achieve to reproduce the dynamics. Indeed, the phase portrait is not a circle but a spiral as if the system was damped. Then, the symplectic scheme achieves to reproduce the dynamics.

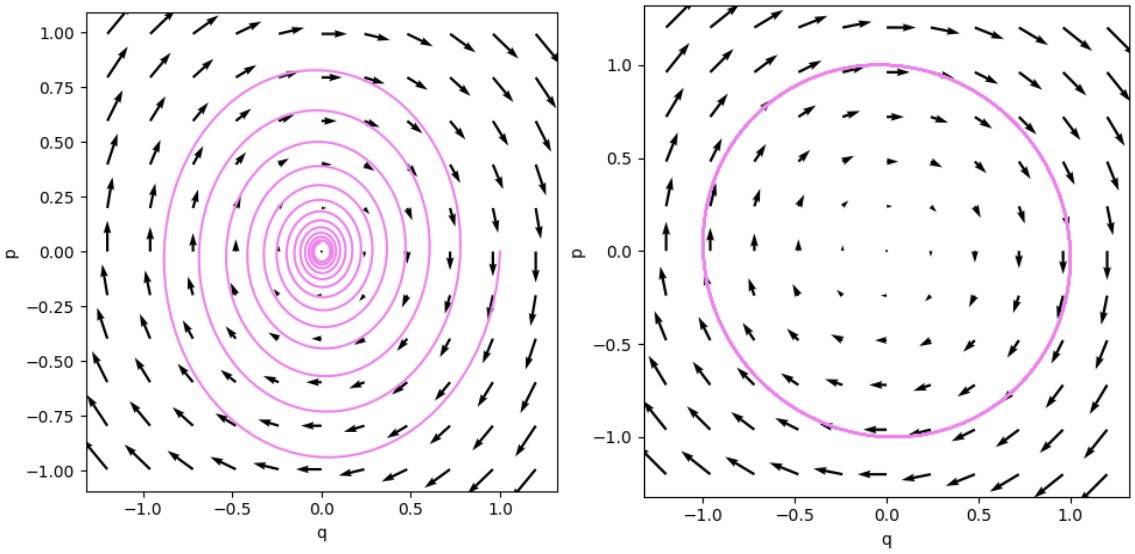


Figure 31: Phase portraits of mass-spring systems dynamics given by an implicit eulerian scheme (left) and a symplectic eulerian scheme (right)

Then, we generate data from a symplectic scheme and try to learn the dynamics i.e. establish a relation between (q, p) and $(\dot{q}, \dot{p})$. We compare a baseline NN $\mathcal{F}_\theta$ with θ its parameters and such that $\mathcal{F}_\theta(q, p) \sim (\dot{q}, \dot{p})$ against an hamiltonian NN (HNN) $\mathcal{H}_\theta$ such that $\mathcal{H}_\theta(q, p) \sim \mathcal{H}(q, p)$. For the baseline, dynamics are reconstructed with an RK4 scheme, and for HNN, they are reconstructed with an Euler symplectic scheme.

Let us observe a prediction with the same parameters as above and $\omega_0 = 1$ on figure 32. On the left graph, we observe trajectories predictions. Whilst the HNN prediction in blue overlaps the true trajectory in red, the baseline prediction in purple deviates from it in a spiral. These results are explained by the evolution of the energy of the system (i.e. the hamiltonian) as a function of the time on the right graph. The true energy in red remains constant, the baseline prediction's energy is not constant and increases, that is why it does not work. The HNN, in blue, roughly preserves the energy; it is sufficient for the HNN to work.

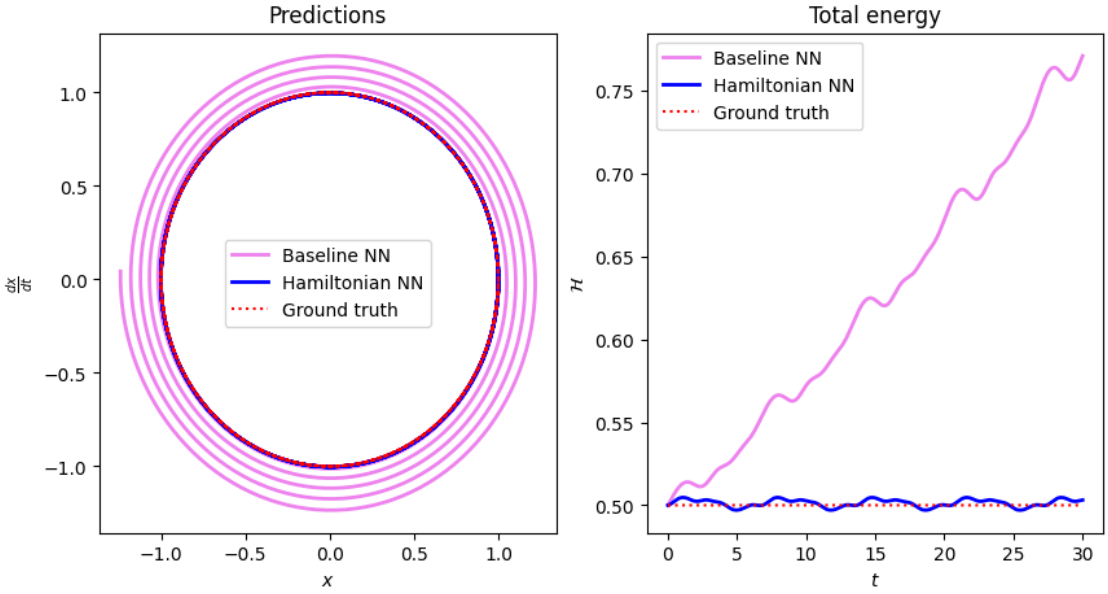


Figure 32: Trajectories comparison between $\mathcal{F}_\theta$ and $\mathcal{H}_\theta$ (left) and energies (right)

Remark. *In addition of providing unsatisfactory results over long time simulation, non-symplectic schemes tends to be unstable i.e. trajectories tends to infinity. Using a HNN and a non-symplectic scheme for prediction does not works either.*

In conclusion, it is important to preserve the symplectic structure as long as possible otherwise results are degraded.

A.5 Machine learning technical elements

In this annex, we discuss technical elements on machine learning for both AEs and HNNs. In addition, we present a mean to manage periodic data that is not yet fruitful. At last, we present how we draw samples from a particles distribution.

A.5.1 Initialisation, training process and hyperparameters

Let us begin with Neural Networks (NN) weights initialisation. We use random orthogonal initial weights matrix. They are determined by the QR decomposition of a matrix composed of random numbers drawn from a standard normal distribution. If the matrix has fewer rows than columns then it has orthogonal rows and conversely. Thus, the article [12] provides theoretical and experimental proofs of its efficiency against a Gaussian initialisation. They prove that it speeds up the convergence time and model performance. They also show that, for deep networks using an orthogonal initialisation, the number of units per layer needed to obtain an efficient convergence is independent from the network's depth as opposed to a Gaussian initialisation in which the width need is linearly dependent on the depth.

As for the optimisation algorithm or so called optimiser, we use AdamW: it is an Adam gradient descent with decoupled weight decays. Although Adam algorithm is very powerful, [13] provides experimental of the superiority of AdamW. They show an overall better performance as a better generalisation for AdamW compared to Adam. This algorithm relies on two hyperparameters, which have been chosen in the beginning and are now fixed, they the learning rate fixed at 5×10^{-4} and the weight decay fixed at 10^{-7}.

Furthermore, the activation tanh provides great results. An L^2 regularisation on weights in the HNN has proven to be effective. The idea is to reduce weights magnitudes by adding a term in the loss function which the L^2 norm of weights. We do not add any regularisation in the AE because doing otherwise deteriorates results.

Finally, we use mini-batches in the training process. In an usual training process, all training data goes through the NN, then we operate the gradient retro-propagation and weights update: this process is called an epoch. Epochs are repeated until the model is sufficiently trained. If necessary, data are passed in batches. In the mini-batch technique, data are eventually passed in batches and, in contrary to a classic epoch, the retro-propagation and weights update is done on every batch. We use an intermediate version and update weights every 20 steps. It speeds up the training process. In practice, we use batches of size 48, except for the parametrised initial condition where we use 256. Indeed, it should be adjusted according to the data set: with multiple simulations, a small batch size could give a model with too much variance.

Remark. *We do not aim to find optimal hyperparameters but working ones in order to provide experimental proof of operation of data driven methods for reduced order modelling.*

A.5.2 Autoencoders building tips

We discuss few construction strategies over different AE architectures.

Dense: We remind that an encoder is composed of multiple dense layers, the first one input dimension is $n \in \{N, 2N\}$ and the last one output dimension is $k \in \{K, 2K\}$ and conversely for the decoder. We construct the decoder as a reflection of the encoder so let us focus on the latter. The first layer has n units, the last has k units. As for number of unit per hidden layer, we can follow two strategies:

- we apply a steady decrease starting from the first layer with a step $(n-k-1)/(s+1)$ with s the number of steps,
- we apply a rough decrease: recursively, we divide the number of unit of the previous layer with an integer b and obtain the number of units of the next layer until we reach the reduced dimension.

In addition, we can repeat every layer to obtain a more complex network.

Light: As explained in section 3.3, a light AE architecture is defined by a set of integer pairs (l, m). As the decoder is a reflection of the decoder, we focus on the prior construction. As a consequence $l \geqslant m$. The closest m is to l, the more number of parameters the model has.

We have found some experimental tips on the selection of (l, m) pairs. It is better for l not to be greater than a fifth of the previous layer output dimension, ideally a tenth for the first layers. It is better to keep m at least as great as half of l to have enough parameters.

A.5.3 On periodic data augmentation

One of the particularities of AEs is that they process continuous data, thereby when given periodic data i.e. with discontinuities, they fail to correctly reconstruct the latter. We find a ingenious approach to make this data continuous while keeping the information that it is periodic: an augmentation.

We consider an entry $\begin{pmatrix} x \\ v \end{pmatrix} \in \mathbb{R}^{2N}$ where x is $L-$periodic. We proceed to an augmentation on positions $x \mapsto (x_c, x_s) \in \mathbb{R}^{2N}$ with:

$$\begin{cases} x_c = R\cos\left(\dfrac{2\pi}{L}x\right), \\ x_s = R\sin\left(\dfrac{2\pi}{L}x\right). \end{cases}$$

where $R \in \mathbb{R}_+^*$ is a radius. Instead of having x in the input, the AE would have (x_c, x_s). To un-augment an output, we use:

$$x = \frac{L}{2\pi}\text{atan2}\left(x_s, x_c\right).$$

Remark. *The radius is chosen according to the NN activation function: we want that the range of entry values is located where the activation function vary in order to avoid dead units layers whole sole purpose is to rescale entry data. For instance with tanh, we would like inputs in a range [-0.2,0.2], more or less to have more active layers. In consequence, we could choose* $R = 0.2$.

It begs the question on how to deal with velocities. Either we leave it unchanged, or we also augment it using $\dot{x} = v$; in the latter case $v \mapsto (v_c, v_s) \in \mathbb{R}^{2N}$ with:

$$\begin{cases} v_c = \dot{x}_c = -\dfrac{2\pi}{L}x_s v, \\ v_s = \dot{x}_s = \dfrac{2\pi}{L}x_c v. \end{cases}$$

As for the un-augmentation:

$$v = s\frac{L}{2\pi R}\sqrt{v_c^2 + v_s^2} \text{ with } \begin{cases} \text{sgn}(v_s)\text{sgn}(x_c) & \text{if } x_c > \epsilon \\ \text{sgn}(v_c)\text{sgn}(x_s) & \text{else} \end{cases}$$

where sgn(.) is the sign function and ϵ a threshold.

Remark. *To keep a reasonable input range in sgn(.), we could choose* $\epsilon = R/2$.

The choice to augment v or not can condition the AE choice. Indeed, the subsequent HNN shall take inputs of even size $2K$, the first half related to positions and the other half related to velocities because it is based on Hamilton's equations. As a consequence, we distinguish two cases:

- we solely augment positions: an AE input is $\begin{pmatrix} x_c & x_s & v \end{pmatrix}^T \in \mathbb{R}^{3N}$. Due to the even size of the latter, we must choose an AE that manage positions and velocities independently: a separated AE. The first would manage $\begin{pmatrix} x_c & x_s \end{pmatrix}^T \in \mathbb{R}^{2N} \mapsto \overline{x} \in \mathbb{R}^K$ and back, the second would manage $v \in \mathbb{R}^N \mapsto \overline{v} \in \mathbb{R}^K$ and back. Note that AEs have different input size.
- We augment both positions and velocities: an AE input is $\begin{pmatrix} x_c & x_s & v_c & v_s \end{pmatrix}^T \in \mathbb{R}^{4N}$. As the size of the positions entry is the same as the velocities entry, we can choose any AE,

In practice with a full augmentation, the error between an input (x_c, x_s) and its output has a major impact on the product of sgn(.) in the un-augmentation. It results in an important error when x_c or x_s is close to zero. Therefore, we would rather not augment velocities.

As periodic boundary conditions are common plasma simulation, It would be interesting to have a functioning process AE and HNN with such conditions. Despite AE achieve great performance on these boundary conditions, HNN does not achieve to learn the reduced dynamics. This is why, although it has been the subject of research, there is no correct results.

A.5.4 Initial conditions drawing

An HNN works with a range of initial conditions, it does not support redrawing nor particles swapping. As a consequence, we must find a convenient mean to draw initial conditions from a given distribution without any randomness. In fact, as we use an inverse method to draw samples from a distribution, it is sufficient to draw uniform samples. Then, we apply the inverse cumulative distribution function to the latter and we obtain an initial condition.

To draw a sample over $[0,1]^2$, we use a low-discrepancy sequence as defined and explained in [14] and called an Hammersley sequence. We note N the number of samples and b a prime number set to 2. A 2D Hammersley sequence is $(e_b(n), \frac{n}{N})_{n\in\{0,\cdots,N-1\}}$ where $e_b(n) \in [0,1]$ is the $b-$ary expansion of n. Any integer n can be expressed in a basis b in the form of a $b-$ary representation:

$$n = \sum_{k=0}^{K} d_k(n) b^k$$

where $0 \leqslant d_k(n) \leqslant b$ is the k^{th} digit of n $b-$ary representation. For instance, any integer can be represented in binary: 5 is expressed as 101 because its binary representation is $2^2 + 2^0$. With this representation, we define the $b-$ary expansion of n:

$$e_b(n) = \sum_{0}^{K} d_k(n) b^{-1-k}.$$

Hammersley sequence is a low-discrepancy sequence, in other worlds it fills well the space. Let us look at an example with $N = 1200$ on figure 33:

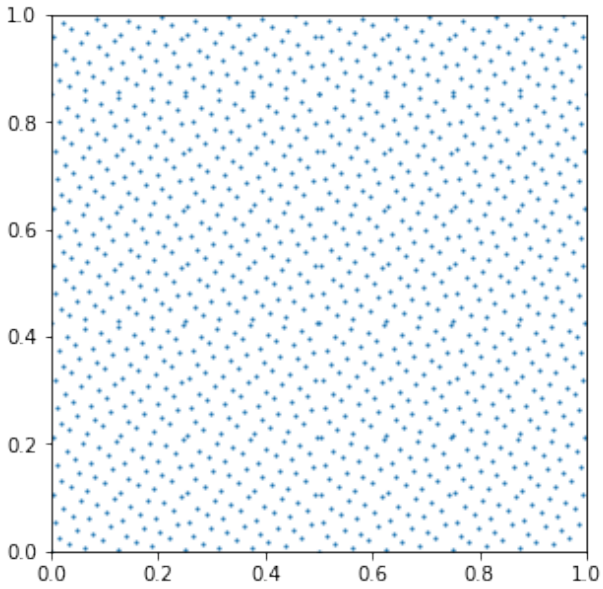


Figure 33: An Hammersley sequence

B Internship related appendices

In this annex, we focus on the six months research internship itself. It can be considered as a preparatory work before my PhD studies starting next year. We begin with its specific objectives that sketch out the road-map, then we present a synthetic logbook with issues encountered and choices we made. We continue with a part on resources used and finish with a general conclusion.

B.1 Specific objectives

The beginning of this internship is dedicated to completely finishing the project that preceded it which is named "Learning Hamiltonian dynamics with neural networks" [7]. We studied data driven and Hamiltonian mechanics based methods to learn simple systems dynamics.

Then, we are able to get into the main subject. We remind that we study the following process: given an initial plasma state in the physical high dimension space, we compress an initial condition into low dimension reduced variables that follow a reduced model. We simulate this reduced model then we decompress reduced variables into a high dimension final state or any other output of interest. This process is depicted on figure 34. As shown in red, we can add data driven methods e.g. Neural Networks (NN) at two key points: the projection from a high dimension to a low dimension space and back and to model the reduced dynamics. Note that these methods need a large data set, building it is one of our first goal.

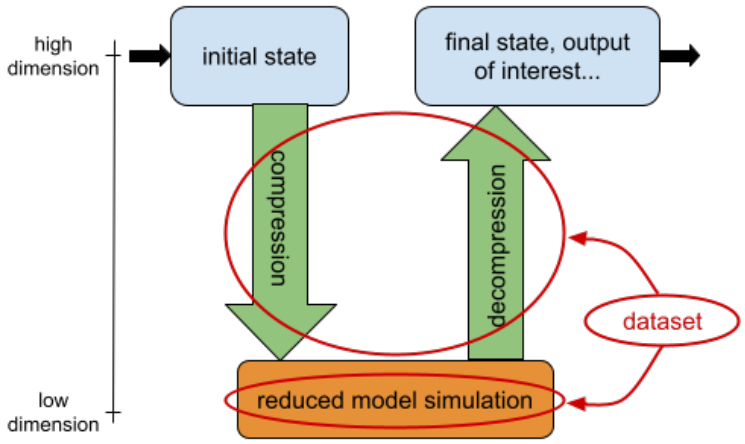


Figure 34: Process of interest

Some major objectives can be identified in chronological order:

- we start with a reference method from article [5]. First steps are to be able to generate data with a Particle In Cell (PIC) algorithm (see section 2.4.1) and to reproduce the method from this article,
- we add a first NN called an Hamiltonian NN (HNN) to replace the reduced model,
- we replace the given reduction method with an other NN called an autoencoder (AE). Going from there, we generalise our results as much as possible with self-induced electric field, different boundary conditions...

B.2 Logbook

We present an indicative and concise timeline of this internship that takes place from the 15^{th} of February 2021 to the 2^{nd} of August 2021 (instead of the 13^{th} thanks to vacations). We focus on important decisions and major issues encountered. It does not include bibliographic research and implementation related issues.

- *15^{th} February:* start of the internship: we properly end the previous project "Learning Hamiltonian dynamics with neural networks".
- *2^{nd} March:* we start the main subject. To simulate a plasma, we implement a PIC algorithm with the help of an example. We study a reference article [5]: they use a Proper Symplectic Decomposition (PSD) for reduced modelling of the Vlasov-Poisson model with solely an external electric field in a periodic space domain. We want to reproduce their results.
- *9^{th} March:* reference results are reproduced. For starters, we would like to learn the reduced dynamics given by [5] using an HNN as in [7]. In addition, we want to properly lay the theoretical background of the PSD: with the linear reduction $u = A\overline{u}$, we can demonstrate that the reduced model is still hamiltonian.
- *23^{rd} March:* the theoretical background has been laid, we would like to extent it to any reduction $u = A(\overline{u})$ in preparation of AEs. The HNN does not work as is with periodic boundary conditions. Knowing that the electric field if solely external, it is sufficient to map particles position to a domain of length the period, or simply consider a non-periodic domain, to obtain satisfactory results. Thus, the physical hamiltonian is separable and we make the hypothesis that the reduced hamiltonian is also separable in the general case: we want to work with two different networks in the HNN (one to model each part of the hamiltonian).
- *6^{th} April:* the separable HNN provides great results with only an external electric field. The demonstration on the hamiltonian-preserving reduced model in the general case $u = A(\overline{u})$ is unfinished. We might miss a trick as the one from [5] when they define the symplectic inverse. Nonetheless, we continue on reduced modelling: we want to vary a physical parameter in an interval, make a data set for various values of this parameter and test if the reduced model is still valid for any value of the latter parameter in the interval, including values unknown to the model. To do so, we use a cosinusoidal electric field and vary its amplitude as in [5]. In parallel, we start to implement a first AE to model $u \mapsto \overline{u}$ and vice-versa and get rid of the linear reduction from the PSD that does not work for a self-consistent field.
- *13^{th} April:* we notice that the HNN does not work when we draw new initial conditions with the same distribution nor when we permute particles. Nevertheless, the combination of PSD and HNN works well with the varying parameter and for about 10^3 particles in a non-periodic domain. We notice that, in the HNN, only the gradient of the hamiltonian is used to predict the dynamics: should it learn the hamiltonian or only its gradient? We look for a new way of drawing initial conditions and start adding a self-consistent electric field to our model.
- *20^{th} April:* we remove any randomness from initial conditions (more details in annex A.5). We verify that the PSD does not work for a self-consistent field. With such fields, we distinguish two cases whether the particles are confined by an electric field or not. As for the periodic boundary condition, we cannot use the same trick as before because the electric field depends on particles positions. Thus, the Poisson problem is not well defined, it results in a possibly faulty field solver. We need to work on it. From now on, we always work with a self-consistent electric field sometimes with an external field (such as a confinement).

- *4th May:* The Poisson problem is correctly defined and the field solver is well implemented for different boundary conditions (Dirichlet, Neumann and periodic). The combination of a PSD and a HNN quite works when the behaviour of the plasma is mostly linear (no confinement and short time of simulation). In other cases, it does not works. We want to build a pipeline for AE and HNN routines. We have a new idea to keep on periodic domains: we project data on a circle for compression and back for decompression (it is called augmentation, more details in annex A.5).

- *11th May:* with a greater number of particles e.g. 10^4, a standard AE becomes too heavy (i.e. too much weights or parameters). We have to work on a light AE (LAE). In addition, we still need to work on the pipeline to automate our work.

- *8th June:* AE+HNN works well with self-consistent and confining electric fields. Nevertheless, results are getting worse in long time run. Augmenting periodic data starts to work, learning from it has not been tested yet.

- *22nd June:* Augmentation does not work with HNN. LAE provides great results in long time run with a great number of particles. We want to test two new structures for the autoencoder: a separated one in which we use one AE/LAE for position and another for velocities, and a mono-block: we use one AE/LAE for positions and we derive velocities using the fact that it is the time derivative of position. We would like to find ways to obtain the non-discernibility of particles in our networks: we take a look at [15, 16, 17]. We give up on the periodic case.

- *6th July:* We make a review of every AE structure with its pros and cons, depending on the number of particles and the time of simulation. We want to parametrize the initial condition again, for instance with a beta law on positions.

- *13th July:* Some parts of the report has already been written, it is time to write the whole report and conclude an in-depth analysis of our results.

B.3 Resources

The code is developed in Python. The machine learning part is implemented using Tensorflow. Concerning its writing and execution, I use VScode on my personal computer in cunjunction with a Google Drive folder. It allows a real-time synchronisation on the cloud and the execution of the code through Google Colab (a free online service that allow to execute Python code for students and researchers) and a Jupyter notebook. Therefore, a distant computer with an efficient GPU is available for machine learning. Code development and execution is divided into two parts: written libraries are stored in a `./SRC` folder. Next to it, we find several notebooks for execution and results analysis. You can find different versions of the project, they correspond to various major changes that have taken place to adapt it to our needs.

This report is written in LaTeXmarkup language with an online editor called Overleaf. It manages compilation and permits an easy access on any computer with an internet connection.

For my supervision, we have a weekly video conference on Google Meet or Big Blue Button. If necessary, we can organise intermediates video conferences between meetings. Written exchanges are done by e-mail. My supervisors also have an access to the Google Drive folder which contains the code and any written information such as demonstrations or reports.

Concerning my working conditions: it is mostly teleworking. I also have a shared office at the faculty with all necessary equipment.

B.4 Analysis and conclusion

To begin the analysis, we take a look at a limited resource: time. Then, we discuss about some difficulties encountered, and we conclude.

We distinguish four time-consuming activities:

- bibliographic research: reading courses, lectures and articles, taking notes,
- code implementation: writing, debugging and optimising the code,
- results analysis: run the code, set correct parameters and analyse outputs,
- reports writing: anything from small reports on a specific subject (a demonstration, a part of theory...) to the whole report.

The figure 35 gives an overall estimate of time consumption of the above-mentioned activities. To resume it, most of the time is spent working on the code. On the one hand, I have to write an error-free code with a decent speed of execution. It can be tricky considering I rely solely on Tensorflow for the calculation part (i.e. apart from drawing initial conditions and generating graphical outputs). Indeed, the scope of possible operations is limited compared to famous libraries such as Numpy, especially with a GPU execution. For instance, an accelerated function with the decorator `@tf.function` has to work solely with tensors, it makes sometimes the use conditional arguments or slicing tricky. On the other hand, we have to analyse results: it means searching for working parameters and hyper-parameters for the machine learning part, and when the results are bad, it must be determined whether it is because of a code error, inadequate parameters or a faulty assumption.

Then, the theoretical and bibliographic part is also important. The theory has to be known to make correct assumptions, to support practical results with a theoretical and demonstrated background. Thus, it is essential to build our work on researches from other people. We should not reinvent what already exists but advance research. However, theories are vast, there is a risk of spending too much time on information that is not relevant to the internship.

At last, writing a report can be time consuming. Indeed, writing in LaTeXmarkup language demands time and most and foremost, it has to be synthetic, which is an exercise in itself.

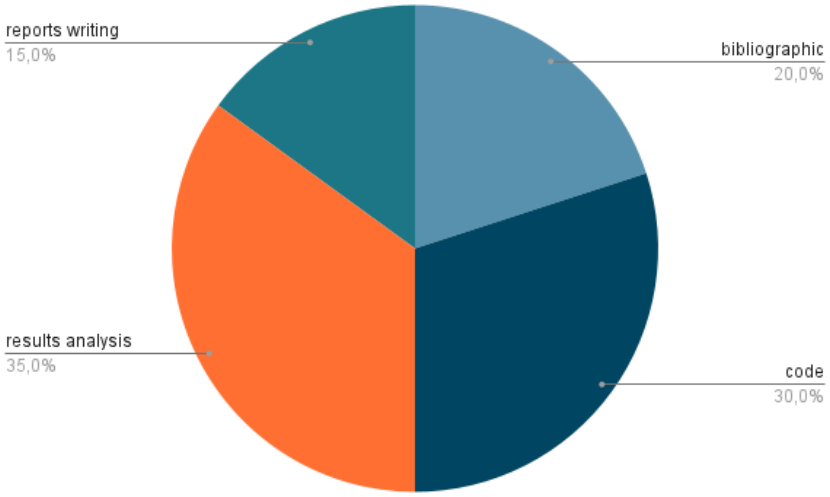


Figure 35: Time consumption

As for the main difficulties we faced, it is worth mentioning the challenge of making a modular and resilient code. Indeed, as the project progresses, needs evolve and the code has to follow these changes in order to produce results in an efficient manner. To avoid too much rewriting, implementation must be modular. This can be difficult because future needs are not known in advance. In addition, one must keep in mind constraints imposed by Tensorflow, especially in accelerated GPU execution.

The other major difficulty is rather theoretical. I have an educational background in applied mathematics which comes in useful in computing, numerical analysis and physics. However, this know-how is quite limited concerning theory. I realise that there is a knowledge gap to fill.

As a conclusion, this internship is a great experience to finish my Master. Despite particular working conditions, it allowed me to tackle a subject I am passionate about. It is also a gateway to my future PhD studies which will be close to the latter. The idea is to become familiar with the latter and the world of research, as well as stacking up useful knowledge. It is also a first experience on a real research subject, where the path is not marked out and some theories are yet to be discovered. This long time internship gives the opportunity to confront to the resulting problems such as the maintenance of an efficient and modular calculation code, the choices and assumptions that have to be made, some of which lead to dead ends. This is where the supervision is crucial, in order to benefit from the knowledge and the experience of seasoned researchers. Knowledge and know-how are important, but their enforcement is difficult without a sufficient experience.

References

[1] Eric Sonnendrücker. *Numerical Methods for the Vlasov-Maxwell equations*. Springer, 2015.

[2] Ernst Hairer. *Geometric Numerical Integration, Lecture 2: Symplectic integrators*. Jan. 2010.

[3] Particle In Cell Consulting LLC. *The Electrostatic Particle In Cell (ES-PIC) Method*. https://www.particleincell.com/2010/es-pic-method/. consulted on the 5th of march 2021. 2010.

[4] Régine Barthelmé. "Le problème de conservation de la charge dans le couplage des équations de Vlasov et de Maxwell." Thèse. 7, rue René Descartes 67084 Strasbourg Cedex (FRANCE): Université Louis Pasteur et CNRS (UMR 7501), 2005.

[5] Tomasz M. Tyranowski and Michael Kraus. "Symplectic model reduction methods for the Vlasov equation". In: (2019). arXiv: 1910.06026 [physics.comp-ph].

[6] Kookjin Lee and Kevin Carlberg. *Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders*. 2019. arXiv: 1812.08373 [cs.NA].

[7] Guillaume Steimer, Laurent Navoret, Emmanuel Franck, and Vincent Vigon. *Project report: Learning Hamiltonian dynamics with neural networks*. 2020.

[8] Sam Greydanus, Misko Dzamba, and Jason Yosinski. *Hamiltonian Neural Networks*. https://greydanus.github.io/2019/05/15/hamiltonian-nns/.

[9] Giovanni Manfredi. "Non-relativistic limits of Maxwell's equations". In: *European Journal of Physics* 34.4 (Apr. 2013), pp. 859–871. ISSN: 1361-6404. DOI: 10.1088/0143-0807/34/4/859. URL: http://dx.doi.org/10.1088/0143-0807/34/4/859.

[10] *Wikipedia: Helmholtz decomposition*. https://en.wikipedia.org/wiki/Helmholtz_decomposition. consulted on the 27th of july 2021.

[11] Eric Sonnendrücker. *Lecture notes in Computational plasma physics*. 2017.

[12] Wei Hu, Lechao Xiao, and Jeffrey Pennington. *Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks*. 2020. arXiv: 2001.05992 [cs.LG].

[13] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[14] *Wikipedia: Low-discrepancy sequence*. https://en.wikipedia.org/wiki/Low-discrepancy_sequence. consulted on the 13rd of April 2021.

[15] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan Salakhutdinov, and Alexander Smola. *Deep Sets*. 2018. arXiv: 1703.06114 [cs.LG].

[16] Huilin Qu and Loukas Gouskos. "Jet tagging via particle clouds". In: *Physical Review D* 101.5 (Mar. 2020). ISSN: 2470-0029. DOI: 10.1103/physrevd.101.056019. URL: http://dx.doi.org/10.1103/PhysRevD.101.056019.

[17] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. *FoldingNet: Point Cloud Auto-encoder via Deep Grid Deformation*. 2018. arXiv: 1712.07262 [cs.CV].