Université de Strasbourg

Master Calcul Scientifique et Mathématiques de l'Information

Reduced Order Modeling For Highly Nonlinear Partial Differential Equations Using Physics Informed Neural Networks

Author
Vincent Italiano

Supervisors
Emmanuel Franck
Victor Michel-Dansac
Hubert Baty
Vincent Vigon

From 14/02/2022 to 19/08/2022

Contents

CHAPTER I

Introduction

1. Context

Inria is the French national research institute for digital science and technology. Established in 1967 at Rocquencourt, on a site that had previously been occupied by NATO, as the Institute for Research in IT and Automation (Iria), Inria's mission is to accelerate, through digital research and innovation, the construction of France's scientific, technological and industrial leadership in European dynamics (according to Inria's website). It is composed by 10 research centers throughout France and even one center in Chile. Its headquarter is located in Rocquencourt, near Versailles. Today, more than 3,900 researchers and engineers work at Inria which hosts over 200 projects teams, most of which are shared with major research universities.

My internship took place in one of those research teams, TONUS (TOkamaks and NUmerical Simulations), of the Nancy-Grand Est research center. The objective of this team is to deal with the mathematical and computing issues derived from kinetic and gyrokinetic modeling and simulations of collisionless plasmas. Together with the University of Strasbourg's Institute for Advanced Mathematical Research (IRMA), they work on fluid plasma models; collisions; reduced, multi-scale and asymptotic models; electromagnetism. And more recently, the research team has been exploring AI solutions for the problems arising from the latter subjects.

My internship was done under the supervision of Emmanuel Franck (Inria), Victor Michel-Dansac (Inria), Hubert Baty (Strasbourg's astronomical observatory), and Vincent Vigon (University of Strasbourg). We worked on the usage of deep learning methods to solve partial differential equations and the construction of reduced order method.

2. Motivation

Over the last decades, artificial neural networks have been used to solve problems in varied applied domains including image classification, handwriting recognition, speech recognition and translation, computer vision and many more [**LeCun et al., 2015**]. Despite the remarkable success in these areas, deep learning has not yet been widely used in the field of scientific computing. However, more recently, solving partial differential equations via deep learning has emerged with a new class of artificial neural networks known as Physics-Informed Neural Networks [**Raissi et al., 2017**].

To obtain the approximate solution of a partial differential equation via deep learning, a key step is to constrain the neural network to minimize the partial differential equation residual, and several approaches have been proposed to accomplish this. Compared to traditional mesh-based methods, such as the finite volume method or the finite element method, deep learning could be a mesh-free approach by taking advantage of automatic differentiation, and could break the curse of dimensionality. Another attractive feature of Physics-Informed Neural Networks is that it can be used to solve inverse problems with minimum change and have been further extended to solve integro-differential equations, stochastic differential equations [**Lu et al., 2021**] and more recently reduced-order modeling of nonlinear problems [**Chen et al., 2021**].

3. Research project

The research project of the internship concerns the application of deep learning methods to solve magnetohydrodynamic equations. These equations combine the Navier-Stokes equations for fluid dynamics with Maxwell's equations for electromagnetism to form a mathematical theory for plasmas. Magnetohydrodynamic helps us understand space plasmas in Earth and planetary magnetospheres, as well as the physics of the Sun, solar wind, and stellar atmospheres. In fusion research, magnetohydrodynamic is crucial to the understanding of plasma equilibria and their stability.

In particular, these partial differential equations are used by physicists to study the magnetic reconnection processes and are parameterized by the plasma resistivity η. The understanding of magnetic reconnection is a major goal of theoretical plasma physics in order to explain explosive events like solar/stellar flares, coronal mass ejections, and gamma-ray flares in pulsar winds. [**Baty, 2019**]

However, the study of the reconnection process requires repeated model evaluations over a potentially large range of parameter values, and high-fidelity simulations remain expensive. Therefore, the goal of the internship is to developp an efficient reduced-order modeling for the magnetohydrodynamic equations using the PINNs framework.

First of all, we applied the physics-informed neural networks to the 1-dimensional Burgers' equation and non-linear 2-dimensional partial differential equation. We learned how to solve partial differential equations using this framework and tested how all the parameters affect the resolution. We also tried different methods to improve the training, using importance sampling. Then we built a reduced model for the 1-dimensional Burgers equation with the viscosity as parameter. Finally, we focused on magnetohydrodynamics and we chose to solve a problem called the tilt instability. We managed to solve this problem with physics-informed neural networks but we needed to include a lot of simulation data in the training process and the research for good hyperparameters for the training was very long and tedious. Due to lack of time, we were not able to design a reduced model which works for the initial problem.

CHAPTER II

Physics informed neural networks

In this section, we introduce Physics informed neural networks (PINNS) [**Raissi et al., 2017**] and we detail the building blocks : neural networks architecture, training, loss, etc.. We then make a comparison between PINNs and FEM and we show how to use PINNs to solve the 1D Burger's equations and a 2-dimensional non-linear PDE.

1. What PINNs are

Physics-Informed Neural Networks (PINNs) are neural networks used to solve problems involving Partial Differential Equations. PINNs approximate PDE solutions by training a neural network to minimize a loss function; this loss contains terms reflecting the initial and boundary conditions and the PDE residual at selected points in the domain (called collocation point).

The PINN algorithm is essentially a mesh-free technique that finds PDE solutions by converting the problem of directly solving the governing equations into a loss function optimization problem. It works by integrating the mathematical model into the network and reinforcing the loss function with a residual term from the governing equation, which acts as a penalizing term to restrict the space of acceptable solutions [**Cuomo et al., 2022**].

1.1. Problem setup. We consider the following generic PDEs defined on a bounded continuous spatial domain $\Omega \in \mathbb{R}^d$ with boundary $\partial\Omega$:

$$\begin{aligned} \mathcal{N}[t, \mathbf{x}, u] &= 0 \quad \text{on } [0,T] \times \Omega \\ \mathcal{I}[\mathbf{x}, u] &= 0 \quad \text{on } \Omega \\ \mathcal{B}[t, \mathbf{x}, u] &= 0 \quad \text{on } [0,T] \times \partial\Omega \end{aligned} \tag{1}$$

where u represents the unknown solution, $\boldsymbol{x} \in \Omega$ is the spatial coordinate, t is the time, T is the time horizon, $\mathcal{N}$ is a general differential operator that may consist of time derivatives, spatial derivatives, and linear and nonlinear terms. Also, $\mathcal{I}$ and $\mathcal{B}$ denote, respectively, the initial and boundary conditions which can be, Dirichlet, Neumann, Robin, or periodic boundary conditions.

A PINN is a neural network $u_{\boldsymbol{\theta}}$, which approximates the solution of the PDE (1), where $\boldsymbol{\theta}$ is the set of model parameters.

1.2. Deep neural network architecture. Since [**Raissi et al., 2017**] and their original vanilla PINN, the majority of PINNs have used feed-forward neural networks (FFNN).

A feed-forward neural network, is a collection of neurons organized in layers that perform calculations sequentially through the layers. Neurons in adjacent layers are connected, whereas neurons inside a single layer are not linked (See Figure 1). Each neuron is a mathematical operation that applies an activation function to the weighted sum of its own inputs plus a bias factor.

Let $\mathcal{N}^L(\mathbf{x}) : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ be a L-layer neural network, with N_l neurons in the l-th layer ($N_0 = d_{in}, N_L = d_{out}$). Let us denote the weight matrix and bias vector in the l-th layer by $\mathbf{W}^l \in \mathbb{R}^{N_l \times N_{l-1}}$ and $\mathbf{b}^l \in \mathbb{R}^{N_l}$, respectively. Given a nonlinear activation function σ, which is applied element-wisely, the FNN is recursively defined as follows :

$$\begin{aligned} \text{input layer :} \quad & \mathcal{N}^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^{d_{in}} \\ \text{hidden layers :} \quad & \mathcal{N}^l(\mathbf{x}) = \sigma(\mathbf{W}^l \mathcal{N}^{l-1}(\mathbf{x}) + \mathbf{b}^l) \in \mathbb{R}^{N_l} \quad \text{for } 1 \le l \le L-1 \\ \text{output layer :} \quad & \mathcal{N}^L(\mathbf{x}) = \mathbf{W}^L \mathcal{N}^{L-1}(\mathbf{x}) + \mathbf{b}^L \in \mathbb{R}^{d_{out}} \end{aligned}$$

In this case, $\boldsymbol{\theta} = \{\mathbf{W}^l, \mathbf{b}^l\}_{1 \leq l \leq L}$ is the set of all weight matrices and bias vectors in the neural network.

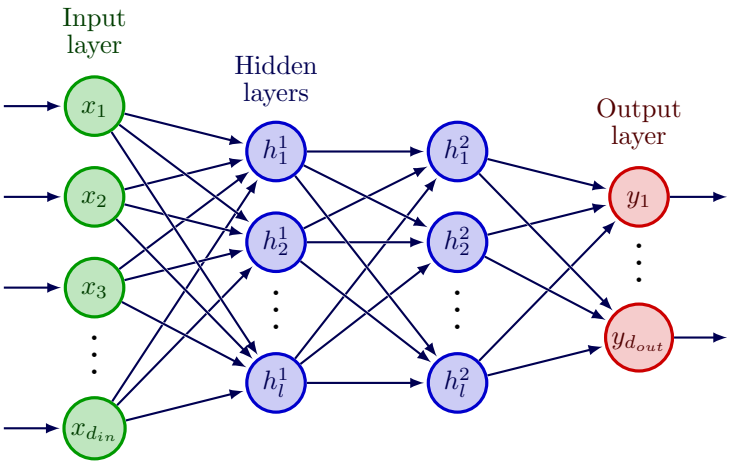

Figure 1. Feed forward neural network with 2 hidden layers.

Note that some researchers have experimented with different types of neural networks like CNN and RNN, and some research papers used PINNs with multiple fully connected networks blended together, e.g. to approximate specific equation of a larger mathematical model. (An architecture composed of five FFNNs is proposed by [**Haghighat et al., 2021**]. Instead of a single neural network across the entire domain, [**Moseley et al., 2021**] suggests using multiple neural networks, one for each subdomain.)

1.3. Training PINNs. Let us consider the following non-negative residuals, defined over the entire spatial and temporal domains

$$
\begin{aligned}
r_{\mathcal{N}}(\boldsymbol{\theta}) &= \int_{[0,T]\times\Omega} \mathcal{N}[t, \mathbf{x}, u_{\boldsymbol{\theta}}]^2 \mathrm{d}t\mathrm{d}\mathbf{x} \\
r_{\mathcal{I}}(\boldsymbol{\theta}) &= \int_{\Omega} \mathcal{I}[\mathbf{x}, u_{\boldsymbol{\theta}}]^2 \mathrm{d}\mathbf{x} \\
r_{\mathcal{B}}(\boldsymbol{\theta}) &= \int_{[0,T]\times\partial\Omega} \mathcal{B}[t, \mathbf{x}, u_{\boldsymbol{\theta}}]^2 \mathrm{d}t\mathrm{d}\mathbf{x}
\end{aligned}
$$

Here, $r_{\mathcal{I}}(\boldsymbol{\theta})$ and $r_{\mathcal{B}}(\boldsymbol{\theta})$ quantifies the misfit of the neural network prediction and the initial/boundary conditions and $r_{\mathcal{N}}(\boldsymbol{\theta})$ quantify how much the neural network is a solution to the PDE.

Therefore, the resolution of the PDE (1) is reduced to an optimization problem, where initial and boundary conditions can be viewed as constraints.

$$
\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, r_{\mathcal{N}}(\boldsymbol{\theta}) \quad s.t \quad r_{\mathcal{I}}(\boldsymbol{\theta}) = r_{\mathcal{B}}(\boldsymbol{\theta}) = 0 \tag{1}
$$

This constrained optimization can be reformulated as an unconstrained optimization with a modified loss function that also accommodates the constraints :

$$
\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta}) \tag{2}
$$

where $J(\boldsymbol{\theta})$ is defined by

$$
J(\boldsymbol{\theta}) = r_{\mathcal{N}}(\boldsymbol{\theta}) + \lambda_1 r_{\mathcal{I}}(\boldsymbol{\theta}) + \lambda_2 r_{\mathcal{B}}(\boldsymbol{\theta}) \tag{3}
$$

in which λ_1 and λ_2 are weight parameters used to adjust the relative importance of each residual term.

To solve this unconstrained optimization problem, mini-batch Stochastic gradient descent (SGD) optimization algorithms are used. In each iteration of a mini-batch SGD algorithm, the gradient of loss function $\nabla_{\boldsymbol{\theta}} J$ is approximated through backpropagation using a batch of points of size m in the

input space, based on which the neural network parameters are updated. More precisely, we select according to a uniform distribution over the concerned domain

1. N_r collocation points $\{t_j^r, \mathbf{x}_j^r\}_{j=1}^{N_r}$ in $[0,T]\times\Omega$
2. N_0 initial points $\{\mathbf{x}_j^0\}_{j=1}^{N_0}$ in Ω
3. N_b boundary points $\{t_j^b, \mathbf{x}_j^b\}_{j=1}^{N_b}$ in $[0,T]\times\partial\Omega$

and at the ith iteration, we select a batch of m collocation points and compute the loss function as a Monte-Carlo estimation

$$J(\boldsymbol{\theta}) \simeq \frac{1}{m}\sum_{j=1}^{m}\mathcal{N}[t_j^r,\mathbf{x}_j^r,u_{\boldsymbol{\theta}}(t_j^r,\mathbf{x}_j^r)]^2 + \frac{\lambda_1}{N_0}\sum_{j=1}^{N_0}\mathcal{I}[\mathbf{x}_j^0,u_{\boldsymbol{\theta}}(0,\mathbf{x}_j^0)]^2 + \frac{\lambda_2}{N_b}\sum_{j=1}^{N_b}\mathcal{B}[t_j^b,\mathbf{x}_j^b,u_{\boldsymbol{\theta}}(t_j^b,\mathbf{x}_j^b)]^2 \tag{4}$$

At the ith iteration, the model parameters are updated according to

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \eta^i \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^i) \tag{5}$$

where η^i is the step size in the ith iteration.

Algorithm 1: PINNs training

Sample N_r collocation points from $[0,T]\times\Omega$, N_0 points from Ω and N_b points from $[0,T]\times\partial\Omega$
Specify optimizer hyper-parameters, and error tolerance ε.
while $J(\boldsymbol{\theta}^i) > \varepsilon$ **do**
  Randomly select a batch of m points out of the N_r collocation points according to a uniform distribution.
  Compute the loss function using equation 4.
  Take a descent step (Equation 5).
end

Moreover, to evaluate the loss function $J(\boldsymbol{\theta})$ at the collocation points, we need to compute the derivatives of the network's output with respect to its inputs. We use automatic differentiation as it is the best choice among the four possible methods for computing the derivatives: (1) hand-coded analytical derivative; (2) symbolic differentiation; (3) finite difference or other numerical approximations; (4) automatic differentiation.

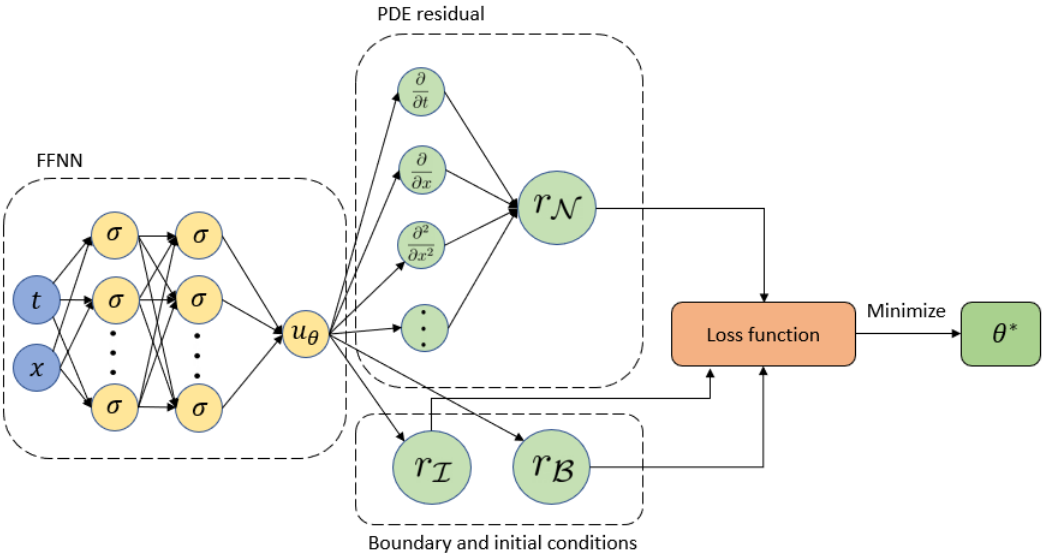

Figure 2. Schematic of a PINN for solving a PDE

2. Comparison between PINNs and other numerical methods

Here we compare the PINNs method to the other classical numerical methods [**Lu et al., 2021**], in particular to FEM and Finite Volume Method as they are the most frequently used.

- In classical numerical methods, the solution is approximated by a piecewise polynomial or constant function with point values to be determined, while in PINNs we construct a neural network parameterized by weights and biases.
- Classical numerical methods requires a mesh generation, whereas PINN is totally mesh-free, and we can use either a grid or random points.
- PINNs' computational cost does not grow with the number of grid points. Furthermore, the trained PINN network may be used to predict values on simulated grids of various resolutions without retraining.
- Classical numerical methods convert a PDE to an algebraic system (using the stiffness matrix and mass matrix in FEM for example), while PINNs embed the PDE and boundary conditions into the loss function.
- In the last step, an algebraic system is solved exactly by a linear solver, but the network in PINNs is learned by a gradient-based optimizer.
- PINNs approximate the function and its derivatives nonlinearly.

3. Example

In this section, we show how PINNs can solve PDEs with two examples: the 1-dimensional Burgers' equations, and a 2-dimensional PDE. The training is performed using TensorFlow on a NVIDIA Tesla V100-PCIE-16GB GPU.

3.1. Burgers equation. As a first example, let us consider the Burgers' equation. This equation arises in various areas of applied mathematics, including fluid mechanics, gas dynamics, and traffic flow.

We consider the one dimensional Burger's equation on the spatial domain $\Omega = [-1, 1]$ and $t \in [0, 1]$, along with Dirichlet boundary conditions :

$$\begin{cases} \partial_t u + \frac{1}{2}\partial_x u^2 = \nu \partial_{xx} u \quad \text{on } [0,1] \times \Omega \\ u(0, x) = \sin(\pi x) \quad \forall x \in \Omega \\ u(t, -1) = u(t, 1) = 0 \quad \forall t \in [0, 1] \end{cases}$$

where ν is the viscosity. It is known that, for small values of the viscosity parameter, Burgers' equation can lead to shock formation that is hard to solve using classical numerical methods. We compare the solution of the Burgers' equation obtained using PINNs to a reference solution computed using finite volume method.

For a viscosity value of $\nu = 10^{-3}$, we learn the solution by training all 3021 parameters of a 9-layer FFNN, with 20 neurons in each hidden layer and a hyperbolic tangent as activation function. We used $N_0 = N_b = 100$ randomly distributed initial and boundary data, and $N_r = 1,000$ collocation points inside the domain and we choose to minimize the loss functions using the Adam optimizer from Tensorflow. Figure 3 shows the solution of the Burgers' equation obtained using PINNs compared to the reference solution at different time instants $t = 0.25, 0.50, 0.75$. Figure 4 shows the loss decreasing during the training and the distribution of training points inside the domain.

For smaller viscosities, we need to increase the number of layers and neurons per layer, and also the number of collocation points and initial/boundary points to achieve a similar level of accuracy. Figure 5 shows the solution of the Burgers' equation obtained using PINNs compared to the reference solution at different time instants $t = 0.25, 0.50, 0.75$ for a viscosity $\nu = 10^{-4}$. We used a 16-layer FFNN, with 40 neurons in each hidden layer and a hyperbolic tangent as activation function. Also we used $N_0 = N_b = 10,000$ randomly distributed initial and boundary data, and $N_r = 30,000$.

We must underline that, unlike any classical numerical method for solving PDEs, this prediction is obtained without any sort of discretization of the spatio-temporal domain. The exact solution for this problem is analytically available.

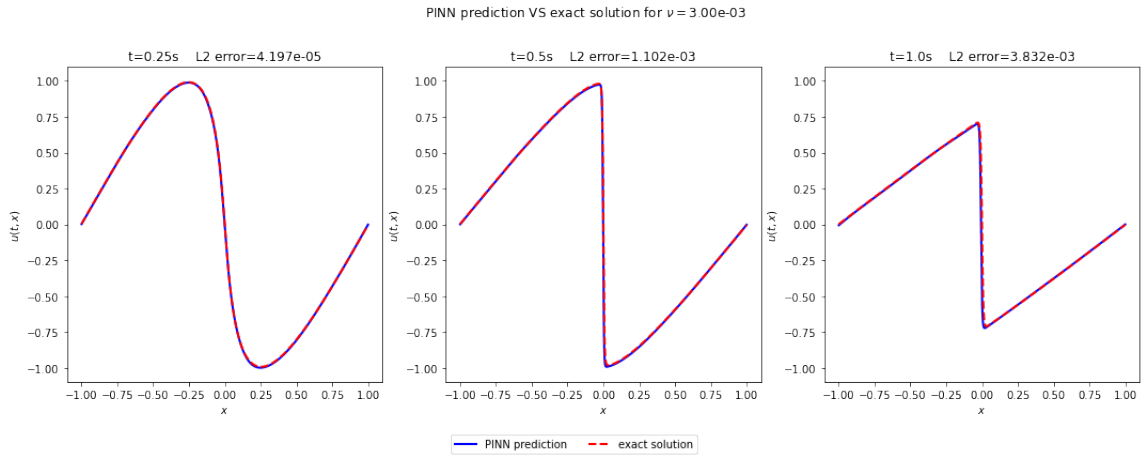

Figure 3. Burgers' equation: Comparison of the predicted and exact solutions corresponding to three temporal snapshots. $\nu = 10^{-3}$

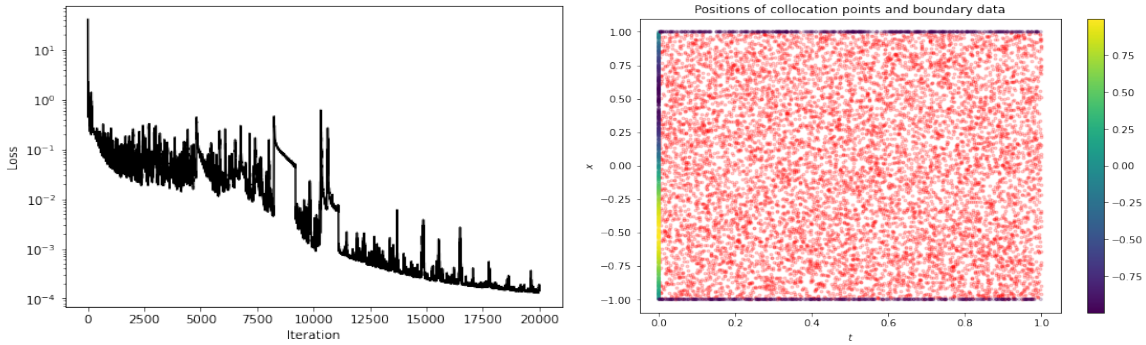

Figure 4. Loss history during training (left) distribution of training points. The colormap correspond to the values of the initial condition function and the boundary condition (right)

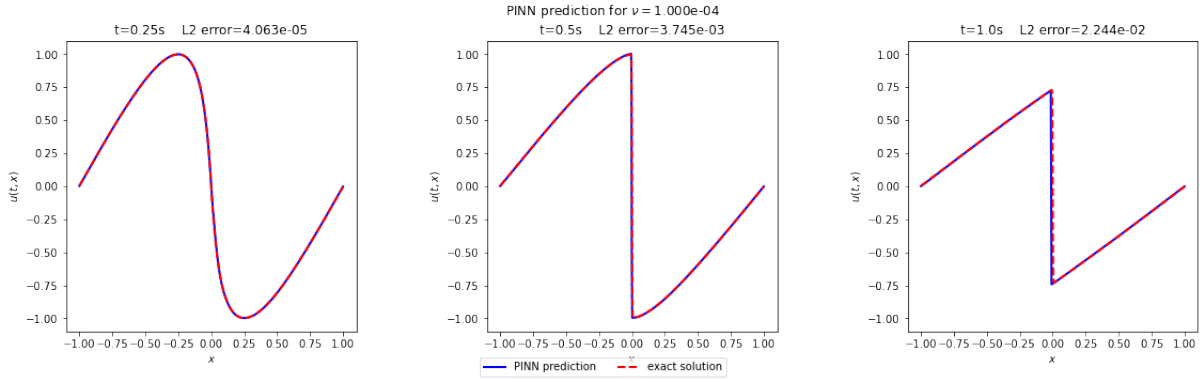

Figure 5. Burgers' equation: Comparison of the predicted and exact solutions corresponding to three temporal snapshots. $\nu = 10^{-4}$

3.2. 2D case. Next, we consider the following viscous 2-dimensional Burgers' equation on the spatial domain $\Omega = [0,1] \times [0,1]$ and $t \in [0, 1.5]$.

$$\begin{cases} \partial_t u + \frac{1}{2}\partial_x u^2 + \partial_y u = \nu \Delta u \quad \text{sur } [0,1.5] \times \Omega \\ u(0,x,y) = 1.5 - 2x \quad \forall (x,y) \in [0,1] \times [0,1] \\ u(t,0,y) = 1.5, u(t,1,y) = -0.5, u(t,x,0) = 1.5 - 2x \end{cases} \tag{6}$$

where ν is the viscosity. Note that the quasi-linear term involving shocks is one dimensional.

Figure 6 shows the solution of equation (6) obtained using PINNs at different time instants $t = 0, 0.3, 0.6, 1.5$ for a viscosity $\nu = 10^{-3}$. We used a 8-layer FFNN, with 150 neurons in each hidden layer and a hyperbolic tangent as activation function. Also, we used $N_0 = N_b = 10,000$ randomly distributed initial and boundary data, and $N_r = 30,000$.

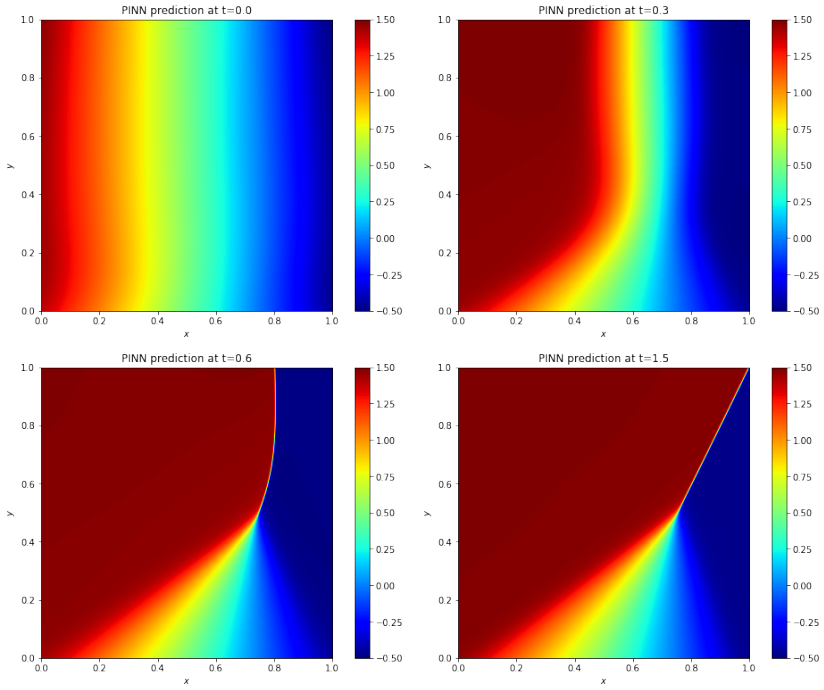

Figure 6. Predicted solutions of equation (6) corresponding to four temporal snapshots. $\nu = 10^{-3}$

We can see that a shock layer is formed, beginning at $(x = 0.75, y = 0.5)$ as a consequence of the steepening of the slope solution when propagating upward and rightward, that has been initiated at the bottom layer. As time advances, a final steady-state solution having an oblique shock layer between the previous point and the top-right corner is obtained.

To check the validity of the PINNs solution, we have a reference solution, computed using finite element method. Figure 8 compares the solution of equation (6) obtained using PINNs and the reference solution for two time instants $t = 0.8, 1.5$. It also shows the quadratic error between the two solutions. Figure 7 compares vertical slices of the solution obtained using PINNs and the reference solution at final time $t = 1.5$, for four different values of y.

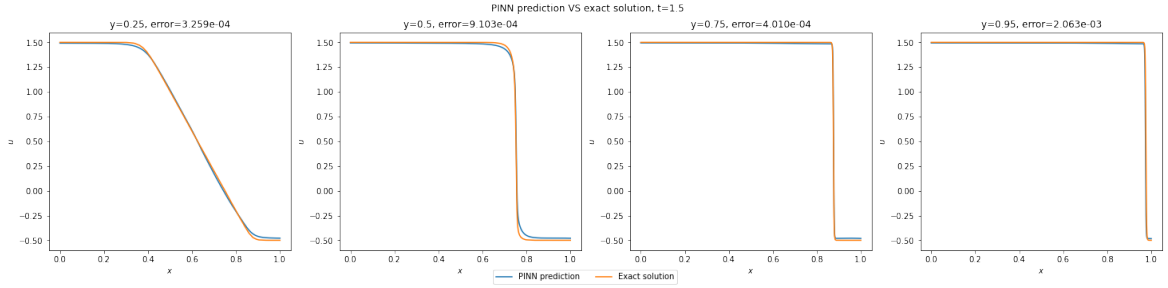

Figure 7. Comparison of vertical slices of the predicted and exact solutions at final time $t = 1.5$, for four different values of y.

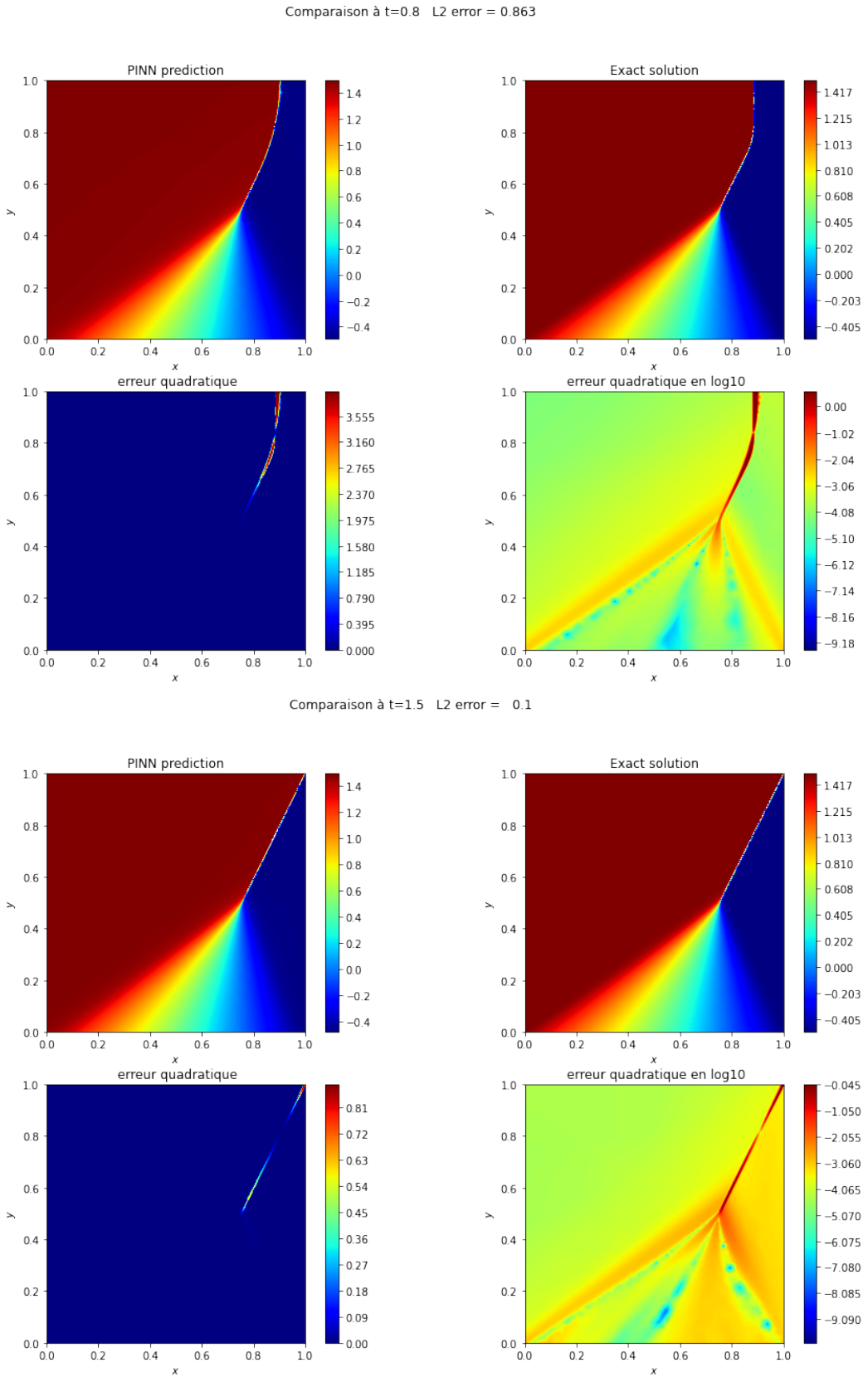

Figure 8. Comparison of the predicted and exact solutions corresponding to two temporal snapshots.

CHAPTER III

Reduced model for Burgers' equation

The first part of the intership was dedicated to the Burgers' equation. First we tried to improve PINNs training using importance sampling and looked for good criteria to select collocation points during training. Then we built a reduced model for the Burgers' equation where the viscosity is considered as a parameter.

1. Importance sampling

In this section, we present the work done to improve PINNs training using importance sampling, We propose to improve the distribution of residual points during the training process, conceptually similar to FEM refinement methods [**Nabian et al., 2021**] [**Lu et al., 2021**].

1.1. Importance sampling. As we discussed in section 1.3, the training is performed using Monte-Carlo method to approximate the loss function $J(\boldsymbol{\theta})$ and the collocation points are uniformly selected in the domain. This works well for most cases, but it may not be efficient for some PDEs that exhibit solutions with steep gradients [**Krishnapriyan et al., 2021**]. For example, the Burgers' equation: intuitively we should put more points near the sharp front to capture the discontinuity well.

In [**Nabian et al., 2021**], they draw training samples from an alternative sampling distribution based on the residual and we will follow their work. They show that training of PINNs can be accelerated using an importance sampling approach where training samples are obtained from a distribution proportional to the loss function. The idea is that we will add more residual points in the locations where the PDE residual is large. We also tried to sample training points from a distribution proportional to the entropy of the solution and compare the two approaches. (see Figure 1)

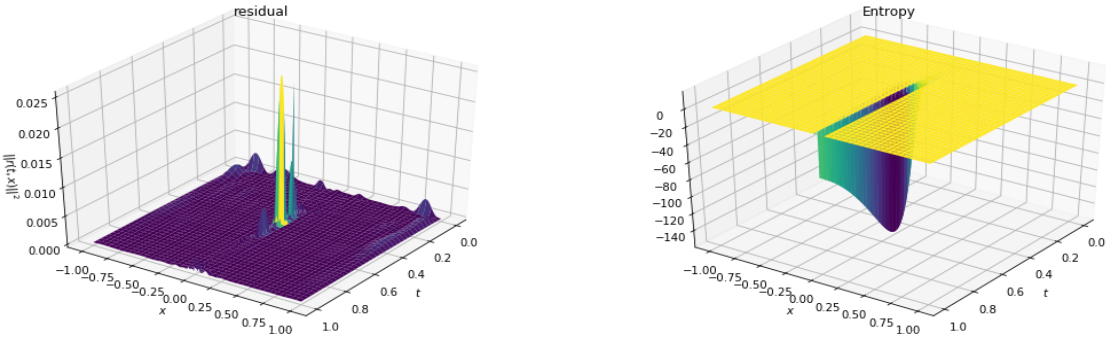

Figure 1. Surface plot of the PDE residual (left) and the entropy (right) associated to a PINN solution. The two distributions put more points near the shock.

This was effectively done by calculating the sampling distributions at the ith training step according to

$$q_j^i = \frac{J(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}^i)}{\sum_{j=1}^{N} J(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}^i)} \quad \forall j = 1, \dots, N_r \tag{7}$$

when using the residual, where $J(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}^i) = \mathcal{N}[t_j^r, \mathbf{x}_j^r, u_{\boldsymbol{\theta}^i}(t_j^r, \mathbf{x}_j^r)]^2$. In the case of Burgers' equation, we have

$$J(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}) = \left[\frac{\partial u_{\boldsymbol{\theta}}}{\partial t}(t_j^r, \mathbf{x}_j^r) + \frac{1}{2}\frac{\partial u_{\boldsymbol{\theta}}^2}{\partial x}(t_j^r, \mathbf{x}_j^r) - \nu\frac{\partial^2 u_{\boldsymbol{\theta}}}{\partial x^2}(t_j^r, \mathbf{x}_j^r)\right]^2 \tag{8}$$

And,

$$q_j^i = \frac{\mathcal{S}(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}^i)}{\sum_{j=1}^{N} \mathcal{S}(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}^i)} \quad \forall j = 1, \dots, N_r \tag{9}$$

when using the entropy. Here $\mathcal{S}$ denotes the entropy function associated to the solution and, for the Burgers' equation, we have

$$\mathcal{S}(t_j^r, \mathbf{x}_j^r; \boldsymbol{\theta}) = \left[\frac{1}{2}\frac{\partial u_{\boldsymbol{\theta}}^2}{\partial t}(t_j^r, \mathbf{x}_j^r) + \frac{1}{3}\frac{\partial u_{\boldsymbol{\theta}}^3}{\partial x}(t_j^r, \mathbf{x}_j^r) - \nu\frac{\partial^2 u_{\boldsymbol{\theta}}^2}{\partial x^2}(t_j^r, \mathbf{x}_j^r)\right]^2 \tag{10}$$

Then we sample collocation points indices from a multinomial with probabilities $\mathbf{p}^i = \{q_1^i, \dots, q_{N_r}^i\}$ which represent the probability of each point in $\{t_j^r, \mathbf{x}_j^r\}_{j=1}^{N_r}$ to be selected during the training step.

Although evaluation of the loss function for the entire set of collocation points in each iteration can be very expensive, we use a piece-wise constant approximation to the loss function to overcome this problem. That is, instead of evaluating the loss function at every collocation point, we evaluate the loss only at a subset of points, hereinafter referred to as "seeds", denoted by $\{t_s, \mathbf{x}_s\}_{s=1}^{S}$, with $S < N_r$. Next, using a nearest neighbor search algorithm (KD-Tree for example), for each collocation point, we identified the nearest seed, and set the loss value at that collocation point equal to the loss at the nearest seed. This is equivalent to generating a Voronoi tesselation using the seeds, and using a constant approximation for loss within each Voronoi cell.

Moreover, compared to [**Nabian et al., 2021**], where they used very smooth PDEs, we want to apply the importance sampling method to the Burgers' equation. We still need to sample some collocation points uniformly otherwise almost all the points would be along the shock at each training step leading to a bad training.

Algorithm 2 summarizes the steps for the proposed importance sampling method.

Algorithm 2: Efficient training of PINNs using importance sampling

Sample N_r collocation points from $[0, T] \times \Omega$, S seeds from $[0, T] \times \Omega$, N_0 points from Ω and N_b points from $[0, T] \times \partial\Omega$.
For each collocation point, find the nearest seed.
Specify optimizer hyper-parameters, batch size m, ratio of uniformly sampled points α and error tolerance ε.
while $J(\boldsymbol{\theta}^{(i)}) > \varepsilon$ **do**
  Compute the loss value at each seed.
  Compute the probability vector $\mathbf{p}^{(i)} = \{q_1^i, \dots, q_{N_r}^i\}$ according to the selected criterion.
  Select $(1-\alpha)m$ collocation points out of the N_r according to the multinomial law of parameter $\mathbf{p}^{(i)}$.
  Select αm collocation points out of the N_r according to uniform distribution.
  Take a descent step.
end

1.2. Results. Here we compare the performance of the importance sampling method using residual and entropy with the classical uniform method.

First of all, we note that the parameter α must be at least equal to 10%, otherwise almost all the points would be along the shock at each training step leading to a bad training. Higher values for α do not lead to much better results. From now on we will always choose $\alpha = 10\%$.

Next, we tested with a challenging viscosity $\nu = 10^{-4}$. Table 1 shows the minimal loss value obtained using the different algorithms, for different values of N_r. For each simulation we train for 20,000 iterations a FFNN with 8 hidden layers and 64 neurons per layer. We used the following hyperparameters: batch size is set to 3,000, learning rate is initially set to 10^{-4} and follows an exponential decay of factor 0.98 every 200 step.

We can see that importance sampling is in general better than uniform sampling and importance sampling with entropy performs better than with residual when the number of collocation points is large. However, it is challenging, in general, to design a good distribution to sample collocation points

	uniform sampling	IS using residual	IS using entropy
10,000 points	9.44^{-2}	4.31^{-2}	7.87^{-2}
20,000 points	7.04^{-3}	8.51^{-3}	2.15^{-2}
40,000 points	6.53^{-3}	3.05^{-3}	1.47^{-3}
100,000 points	4.75^{-3}	7.48^{-4}	5.32^{-4}

Table 1. Best loss reach during training. $\nu = 10^{-4}$

for problems whose solution is unknown (e.g when we do not know the entropy function), thus the residual method is prefered.

2. Reduced model

In this section we build a reduced order model coupled with PINNs to solve the Burgers' equation with viscosity as a parameter. As a result, we reach an accuracy approaching 10^{-4} using a single PINN to predict the solution of the Burgers' equation for viscosities in the interval $[10^{-4}, 10^{-2}]$.

The advantage of PINNs over traditionnal numerical method is that the computational cost does not grow that much with the PDEs dimensions and the number of grid points. We exploit these properties to construct a reduced model using PINN : parametrized PDEs are viewed as higher dimensional PDEs where the parameters are also variables.

2.1. Results. We adapt the previous PINNs setup to the new problem, adding 1 input to the model, corresponding to the viscosity value. The new model is a 16-layer FFNN, with 256 neurons in each hidden layer and a hyperbolic tangent as activation function. Also we used $N_0 = N_b = 20,00$ randomly distributed initial and boundary data and $60,000$ collocation points. The model is trained for viscosities in the interval $[10^{-4}, 10^{-2}]$ using the following hyperparameters : batch size is set to $3,000$, learning rate is initially set to 10^{-3} and follows an exponential decay of factor 0.97 every 200 step. We train for $20,000$ step. Figure 2 shows the loss decreasing during the training.

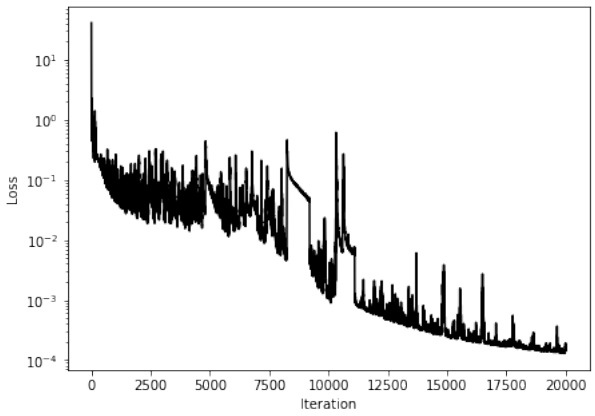

Figure 2. Training loss of the PINN reduced model

Figure 3 shows the predicted solution of the Burgers' equation for different viscosity values on the same plot, at different times $t = 0, 0.25, 0.5, 1$. Figure 4 compares the model prediction to the solution computed using Finite Volume Method, for different time (horizontal) and different viscosities (vertical).

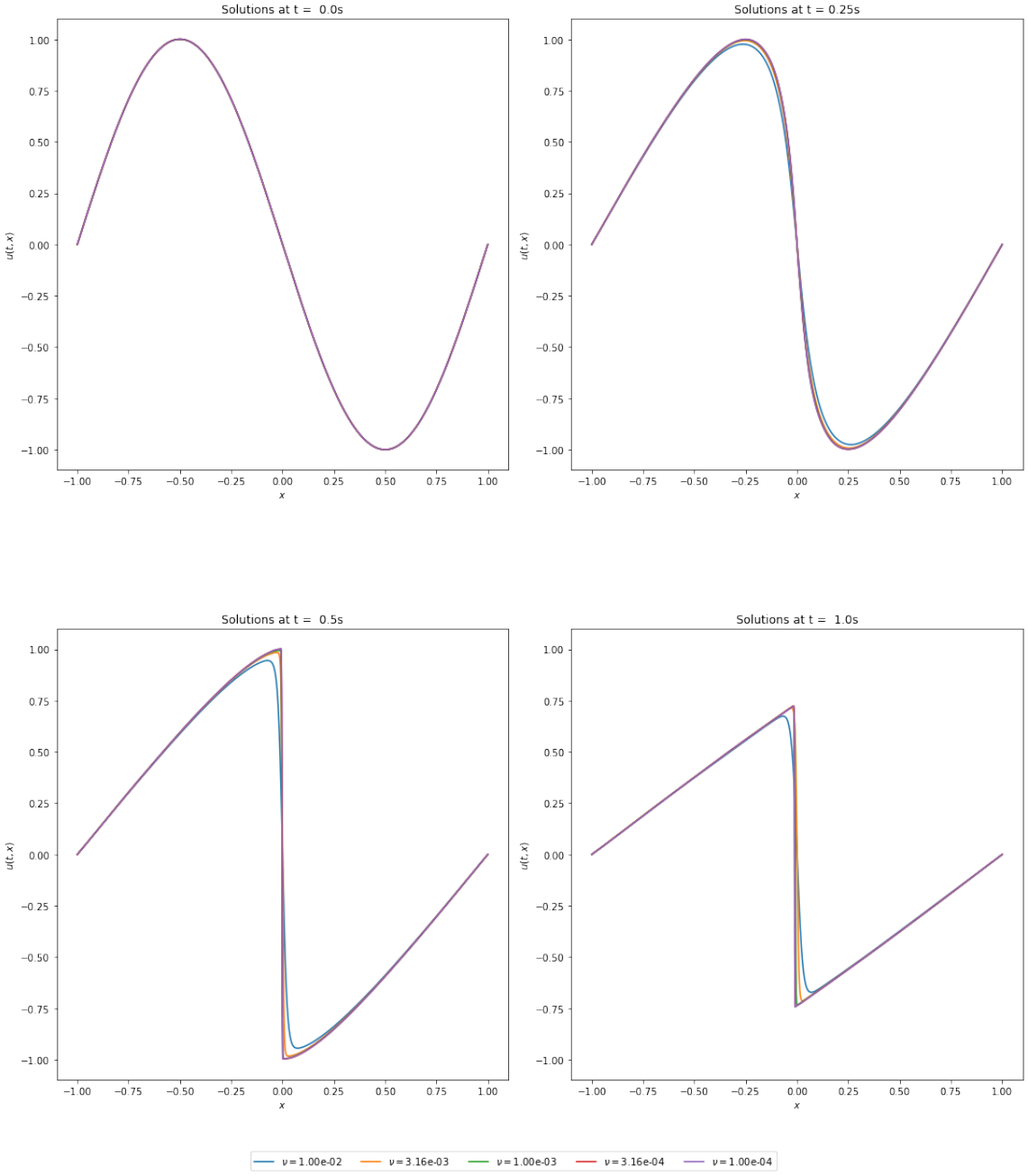

Figure 3. PINN prediction for different time and different viscosities

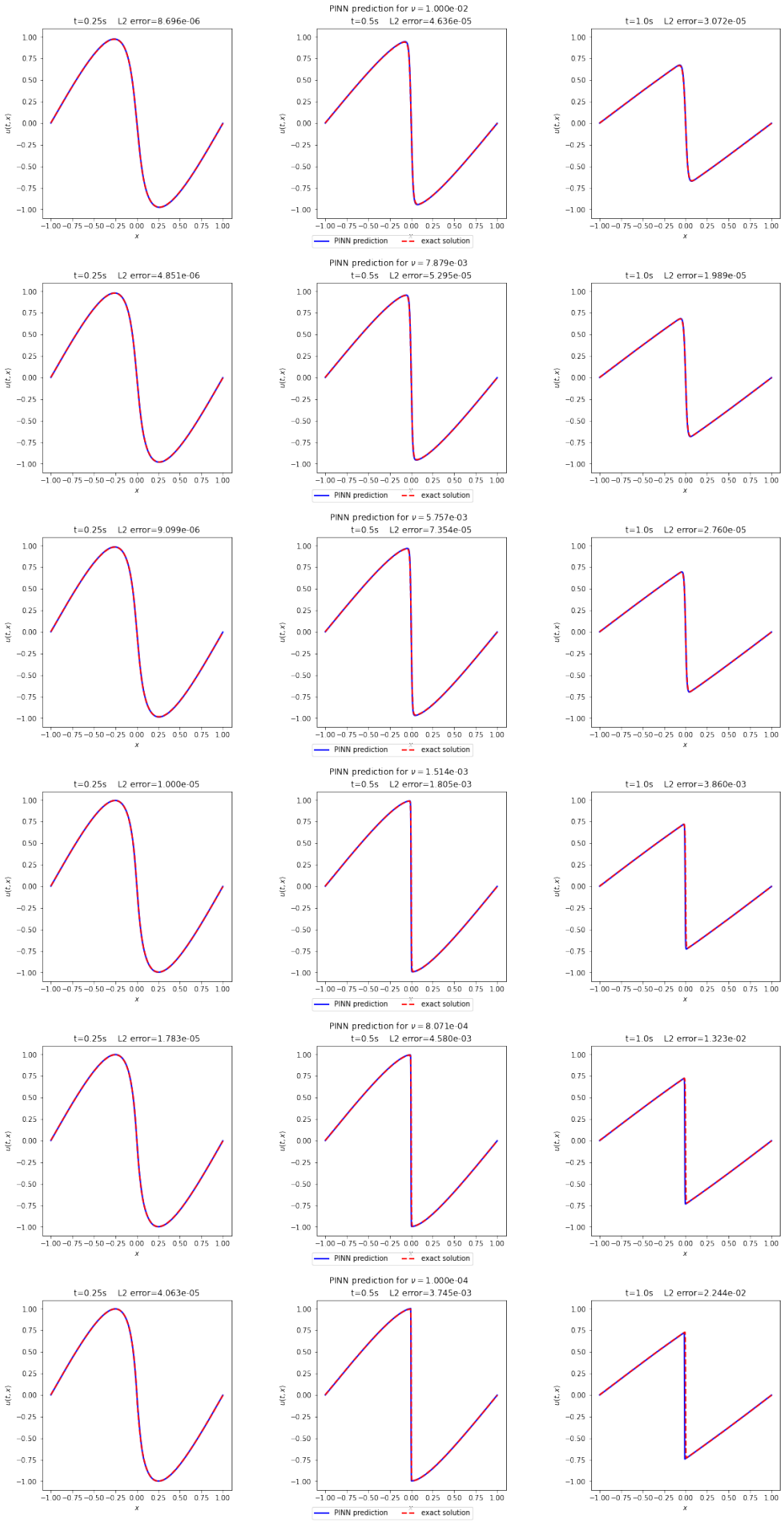

Figure 4. Solutions of Burgers' equation for different viscosity values, using a single neural network

CHAPTER IV

Reduced order model for magnetohydrodynamics

In the second part of the intership, we apply the PINNs framework to magnetohydrodynamics. First, we introduce the magnetohydrodynamic and its governing equations, then we present the tilt instability configuration, and we present the results obtained using PINNs to solve this problem.

1. What is magnetohydrodynamics

Magnetohydrodynamics (MHD) is concerned with the flow of electrically conducting fluids in the presence of magnetic fields, either externally applied or generated within the fluid by inductive action. The basic equations of MHD have been proposed by the Swedish physicist Hannes Alfven (1908-1995), who received the Nobel Prize in Physics in 1970 for *"fundamental work and discoveries in magnetohydrodynamics with fruitful applications in different parts of plasma physics"*.

Figure 1. Hannes Alfven (1908-1995)

MHD combines the Navier-Stokes equations for fluid dynamics with Maxwell's equations for electromagnetism to form a new mathematical theory for plasmas. The main concept behind MHD is that magnetic fields can induce currents in a moving conductive fluid, which in turn creates forces on the fluid and influences the magnetic field itself. MHD helps us understand space plasmas in Earth and planetary magnetospheres, as well as the physics of the Sun, solar wind, and stellar atmospheres (Figure 2). In fusion research, MHD is crucial to the understanding of plasma equilibria and their stability.

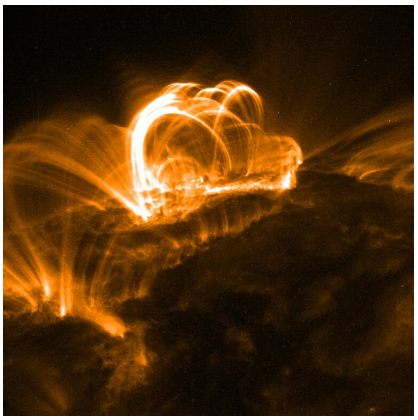

Figure 2. Solar flare, which can release particles and radiations.
Source : http://www-solar.mcs.st-and.ac.uk/~herbert/Before_2018/

1.1. Equations of MHD. The equations of magnetohydrodynamics are a reduction of the equations of fluid mechanics coupled with Maxwell's equations. Magnetohydrodynamics is composed of four equations, which consist of the basic conservation laws of mass, momentum and energy together with the induction equation for the magnetic field. [Baty, 2022]

The first equation of MHD is the continuity equation, which expresses the conservation of mass density

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \tag{a}$$

where ρ is the mass density and $\mathbf{V}$ the flow velocity.

The second equation quantifies how the magnetic field acts on the hydrodynamic component.

$$\rho \frac{\partial \mathbf{V}}{\partial t} + \rho(\mathbf{V} \cdot \nabla)\mathbf{V} = \mathbf{J} \times \mathbf{B} - \nabla P + \mu \Delta \mathbf{V} \tag{b}$$

Here $\mathbf{B}$ is the magnetic field (subjected to the relation $\nabla \cdot \mathbf{B} = 0$), $\mathbf{J} = (\nabla \times \mathbf{B})/\mu_0$ the current density, P is the fluid pressure, $\mathbf{g}$ the gravitational acceleration and μ the dynamic viscosity coefficient of the fluid.

The other way around, the third equation quantifies how the flow velocity acts on the magnetic field.

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{V} \times \mathbf{B}) - \nabla \times (\mathbf{J}/\sigma) \tag{c}$$

where σ is the electrical conductivity coefficient. This equation comes from the Maxwell-Faraday equation combining with Ohm's law of conducting fluid.

The last equation is an equation of state, needed to close the set of relations. Neglecting the Joule heating, we can consider the following equation

$$\frac{\partial P}{\partial t} + \mathbf{V} \cdot \nabla P = -\Gamma P \nabla \cdot \mathbf{V} \tag{c}$$

where Γ is the heat capacity ratio.

Finally, MHD is governed by the following set of equations,

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0 \tag{11}$$

$$\rho \frac{\partial \mathbf{V}}{\partial t} + \rho(\mathbf{V} \cdot \nabla)\mathbf{V} = \mathbf{J} \times \mathbf{B} - \nabla P + \mu \Delta \mathbf{V} \tag{12}$$

$$\frac{\partial \mathbf{B}}{\partial t} = \nabla \times (\mathbf{V} \times \mathbf{B}) - \nabla \times (\mathbf{J}/\sigma) \tag{13}$$

$$\frac{\partial P}{\partial t} + \mathbf{V} \cdot \nabla P = -\Gamma P \nabla \cdot \mathbf{V} \tag{14}$$

with $\mathbf{J} = (\nabla \times \mathbf{B})/\mu_0$ and $\nabla \cdot \mathbf{B} = 0$.

1.2. Reduced MHD. Reduced magnetohydrodynamics is an incompressible fluid model of plasma behavior that is simpler than a full MHD model. It has the advantage of being computationally more efficient for many problems for which it is applicable. It is a good approximation when the magnetic field is almost uniform and unidirectional : $\mathbf{B} \simeq B_z \mathbf{e}_z$. This situation may occur in many situations like loops in the solar atmosphere, Tokamaks, Magnetic clouds/flux ropes in the solar wind.

In this case, it can be shown that the plasma is incompressible : $\nabla \cdot \mathbf{V} = 0$ and its dynamics is constrained to a 2-dimensional plane: i.e $\mathbf{V} = V_x \mathbf{e}_x + V_y \mathbf{e}_y$. Thus the full MHD equations are reduced to simplified equations in 2 dimensions, reducing the numbers of scalar variables from 8 to 4 but also leads to a better numerical handling.

1.2.1. *Flow-vorticy formulation.* We first consider reduced MHD equations written in flux-vorticity $(\psi - \omega)$ scalar variables,

$$\frac{\partial \omega}{\partial t} + (\mathbf{V}\cdot\nabla)\omega = \frac{1}{\rho}(\mathbf{B}\cdot\nabla)J + \nu\Delta\omega \tag{15}$$

$$\frac{\partial \psi}{\partial t} + (\mathbf{V}\cdot\nabla)\psi = \eta\Delta\psi \tag{16}$$

$$\omega = -\Delta\phi \tag{17}$$

$$\mu_0 J = -\Delta\psi \tag{18}$$

Where we have introduced the two stream functions, $\phi(x,y)$ and $\psi(x,y)$, defined as $\mathbf{V} = \nabla\phi\times\mathbf{e}_z$ and $\mathbf{B} = \nabla\psi\times\mathbf{e}_z$ ($\mathbf{e}_z$ being the unit vector perpendicular to the xOy simulation plane) and ω and J are the z components of the vorticity and the current density vectors, as $\boldsymbol{\omega} = \nabla\times\mathbf{V}$ and $\mathbf{J} = (\nabla\times\mathbf{B})/\mu_0$. The coefficient $\eta = 1/(\mu_0\sigma)$ is the magnetic resistivity. Note that the thermal pressure gradient is naturally absent from our set of equations.

1.2.2. *Current-vorticity formulation.* Another formulation using current-vorticity $(J-\omega)$ exists, and is given by,

$$\frac{\partial \omega}{\partial t} + (\mathbf{V}\cdot\nabla)\omega = \frac{1}{\rho}(\mathbf{B}\cdot\nabla)J + \nu\Delta\omega \tag{19}$$

$$\frac{\partial J}{\partial t} + (\mathbf{V}\cdot\nabla)J = (\mathbf{B}\cdot\nabla)\omega + \eta\Delta J + g(\phi,\psi) \tag{20}$$

$$\omega = -\Delta\phi \tag{21}$$

$$\mu_0 J = -\Delta\psi \tag{22}$$

with $g(\phi,\psi) = 2[\frac{\partial\psi}{\partial x}, \frac{\partial\phi}{\partial x}] + 2[\frac{\partial\psi}{\partial y}, \frac{\partial\phi}{\partial y}]$. This second formulation has advantages over the first one, as the two evolution equations are more symmetric, thus facilitating the numerical matrices calculus for linear solvers and there is no numerical instabilities due to the evaluation of the third order spatial derivative term $(\mathbf{B}\cdot\nabla)J$ as J is itself deduced from $\mu_0 J = -\Delta\psi$.

2. The tilt instability

The reduced MHD problem we worked on during the internship is the tilt instability between two repelling currents which leads to reconnection. Magnetic reconnection is a fundamental process in laboratory and space plasmas, which allows the conversion of magnetic energy into bulk flow and heating. The understanding of magnetic reconnection is a major goal of theoretical plasma physics in order to explain explosive events like solar/stellar flares, coronal mass ejections, and gamma-ray flares in pulsar winds.

In this section we will present the problem setup following [**Baty, 2019**].

2.1. Problem setup. The initial magnetic field configuration for tilt instability is a dipole current structure. It consists of two oppositely directed currents which tend to repel, embedded in a constant magnetic field (see Figure 3). The boundaries are taken to be relatively far from the centre, in order to have a weak effect on the central dynamics so we choose the spatial domain $\Omega = [-3,3]^2$.

The initial equilibrium is defined by the following magnetic flux distribution,

$$\psi_e(x,y) = \begin{cases} \left(\frac{1}{r} - r\right)\frac{y}{r} & r > 1 \\ -\frac{2}{kJ_0(k)}J_1(kr)\frac{y}{r} & r \le 1 \end{cases} \tag{23}$$

and the corresponding current density

$$J_e(x,y) = \begin{cases} 0 & r > 1 \\ -\frac{2k}{J_0(k)}J_1(kr)\frac{y}{r} & r \le 1 \end{cases} \tag{24}$$

where $r = \sqrt{x^2 + y^2}$, J_0 and J_1 are Bessel functions of order 0 and 1 respectively, and k is the first non-zero root of J_1 i.e $k \simeq 3.83170597$.

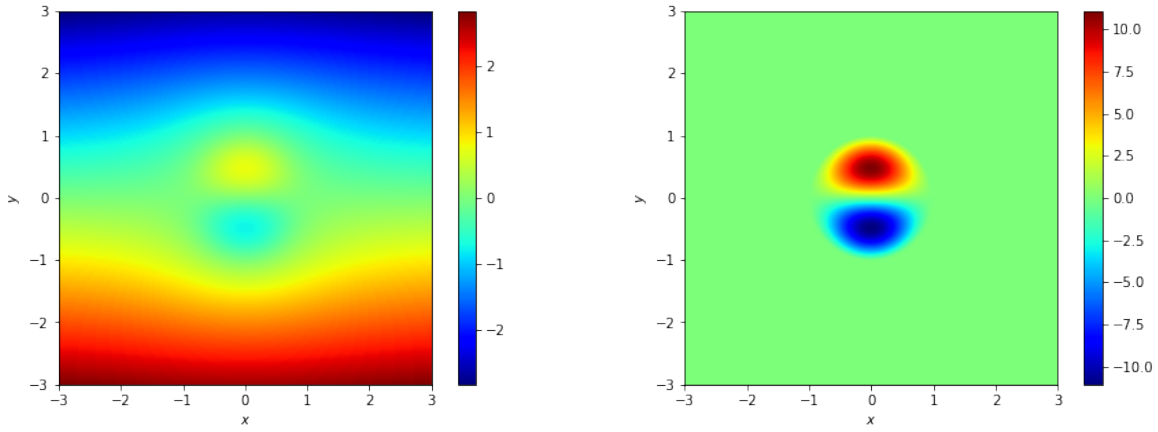

Figure 3. Initial configuration for the tilt instability showing the dipole current density ψ_e (left panel) and the density current J_e (right panel).

Moreover, an initial zero stream function is assumed $\phi_e(x, y) = 0$, with zero initial vorticity $\omega_e(x, y) = 0$ and we assume $\eta = \nu$ and $\rho = \mu_0 = 1$. Since we consider an incompressible reduced MHD model, thermal pressure is absent and we are not concerned by such choice. The initial current structure and current density are reported in Figure 3.

2.2. Typical evolution of the tilt instability. During the early time evolution of the system corresponding to the tilt instability, the pair of oppositely directed currents tend to repel one another giving rise to a rotation (see Figure 5 for a case with $\eta = 3 \times 10^{-3}$) during the linear phase. The sense of rotation of the structure (anticlockwise in Figure 5) is not predetermined and depends only on the numerical noise. This rotation causes two new regions of current density taking the form of two bananas.

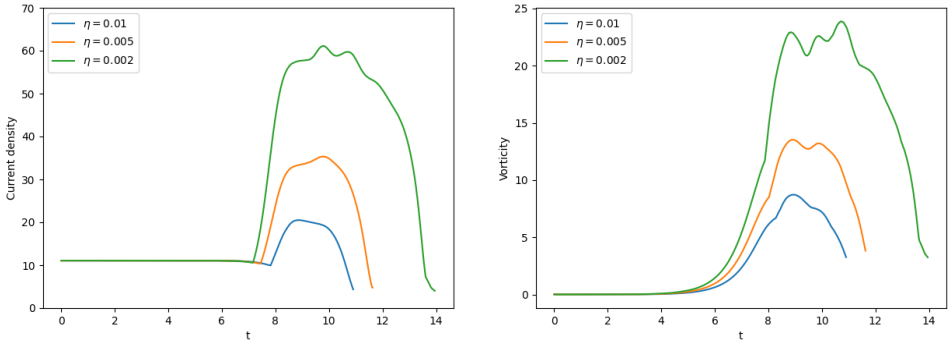

Figure 4. Time history of the maximum current density amplitude (left panel) and maximum vorticity amplitude (right panel) for three resistivity values $\eta = 0.01, 0.005$ and 0.002.

The time history of the maximum current density amplitude and the maximum vorticity amplitude (taken over the whole domain) is shown to increase exponentially in time over the equilibrium value, as illustrated in Figure 4 for three different viscosity values $\eta = 0.01, 0.005$ and 0.002. A steady-state is obtained with a nearly constant current density structure subsequently forming, driving thus the reconnection process.

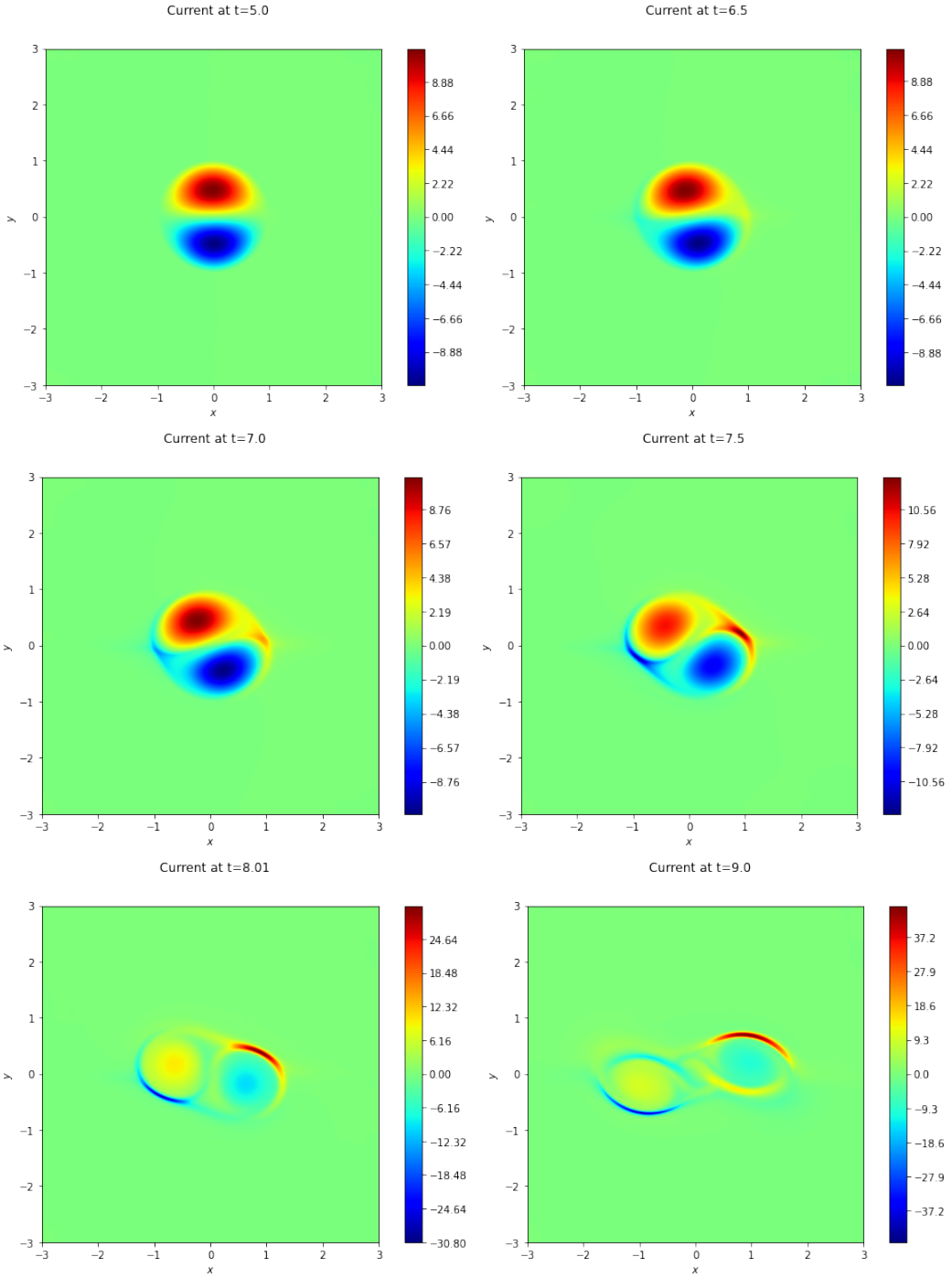

Figure 5. Current density at different times during the development of the tilt instability for $\eta = 3 \times 10^{-3}$.

3. Numerical results

In this section, we describe the resolution of the previous tilt problem using PINNs. The objective was to reproduce the evolution of the system with PINNs. Using a reference solution given by H. Baty (Strasbourg's astronomical observatory), based on finite elements, to check the validity of our solution. Unfortunately, it took a lot of time to tune the model's hyperparameters and make the PINNs work so we did not have time to design a reduced model that works.

3.1. Model design. First of all, we decided to modify the PINNs loss from Equation 4 and remove the boundary and initial conditions residual, and add a data misfit term instead. The new loss writes

$$J(\boldsymbol{\theta}) \simeq \frac{1}{m}\sum_{j=1}^{m}\mathcal{N}[t_j^r, \mathbf{x}_j^r, u_{\boldsymbol{\theta}}(t_j^r, \mathbf{x}_j^r)]^2 + \underbrace{\frac{\lambda}{N_d}\sum_{j=1}^{N_d}\left[u_j^d - u_{\boldsymbol{\theta}}(t_j^d, \mathbf{x}_j^d)\right]^2}_{=J_{\text{data}}(\boldsymbol{\theta})} \tag{25}$$

where $\{t_j^d, \mathbf{x}_j^d\}_{j=1}^{N_d}$ is the set of training data points, $\{u_j^d\}_{j=1}^{N_d}$ represents the values that the exact solution take on these points and λ is a weight parameter used to adjust the relative importance of each term. The first term on the right hand side of the equation is still the residual of the PDE, while the second one now quantifies how much the PINNs fits the data. We use this trick because after many attempts, the PINNs was unable to trigger the instability, so this is a way to start the simulation at an advanced time in the system evolution. But this trick also helps to guide the training by imposing the PINNs to take specific value at some points. In practice, we have several data sets from a high-fidelity simulation at multiple time step.

Moreover, we saw in a previous section that the scalar variables involved in the tilt instability, that is ω, ϕ, J and ψ, can take a wide range of values, typically from 10^{-3} to 10^2 for a small viscosity. In principle, using sufficiently wide networks would allow the PINNs to associate different parts of each layer to a specific output. However, the wide range of values taken by the solution will result in a bad approximate and, after many attempts, we find that using separate networks facilitates the training. Therefore, we used two independent FFNNs blended together as in [**Haghighat et al., 2021**]. The first one is used to approximate the hydrodynamic variables ω and ϕ while the second one to approximate the magnetic variables J and ψ. Also, we remark that the choice of a specific formulation for reduced MHD have no incidence on the training, so we use the flow-vorticity formulation.

Finally, assuming that we use D data snapshots, we have to modify the loss in Equation 25, and add many parameters to ponderate the loss between the different scalar variables misfits for each data snapshot. The final data loss J_{data} writes,

$$J_{\text{data}}(\boldsymbol{\theta}) = \sum_{d=0}^{D}\Big(\frac{\lambda_d^\omega}{N_d}\sum_{j=1}^{N_d}\left[\omega_j^d - \hat{\omega}(t_j^d, \mathbf{x}_j^d)\right]^2 + \frac{\lambda_d^\phi}{N_d}\sum_{j=1}^{N_d}\left[\phi^d - \hat{\phi}(t_j^d, \mathbf{x}_j^d)\right]^2 \tag{26}$$

$$+ \frac{\lambda_d^J}{N_d}\sum_{j=1}^{N_d}\left[J_j^d - \hat{J}(t_j^d, \mathbf{x}_j^d)\right]^2 + \frac{\lambda_d^\psi}{N_d}\sum_{j=1}^{N_d}\left[\psi^d - \hat{\psi}(t_j^d, \mathbf{x}_j^d)\right]^2\Big) \tag{27}$$

Where terms with a hat are the PINNs prediction, and the lambdas are weights parameters used to adjust the relative importance of each term, which adds to the 4 other weights corresponding to the four PDEs equations to reach a total of $4 \times (D + 1)$ hyperparameters to adjust for the training process.

3.2. Results. We trained two networks with the same architecture : 16-layers FFNN with 256 neurons in each hidden layer. The first one corresponding to the hydrodynamic variables use the ReLu activation function while the other, corresponding to the magnetic variables use the hyperbolic tangent function. We trained the PINNS with the following hyperparameters : batch size is set to 3,000, the learning rates for the two optimizers are the same, and were initially set to 10^{-3} and follows an exponential decay of factor 0.98 every 200 step. Also we set the resistivity $\eta = 3 \times 10^{-3}$, used

$10,000,000$ collocation points and 10 different data snapshots regularly spaced in time , corresponding to the beginning of the exponential croissance phase ($t = 4$) until the reconnection process ($t = 10$).

The tuning of all the hyperparameters, in particular the weights parameters between the different terms in the loss, was a long and tedious task. Indeed, the PDE system we look is highly non-linear, the functions we want to approximate take complex shapes that evolve a lot, and take wide range of values. But we finally manage to obtain good approximations.

Figure 6 compares PINNs prediction to the real solution data. It shows the time history of the maximum current density amplitude and maximum vorticity amplitude on the left hand side, and the value of kinetic and magnetic energy of the system on the right hand side, computed using the formula

$$E_{\text{cin}}(t) = \frac{1}{2}\int_{\Omega} |\nabla\phi(t,\mathbf{x})|^2 \mathrm{d}\mathbf{x} \quad \text{and} \quad E_{\text{mag}}(t) = \frac{1}{2}\int_{\Omega} |\nabla\psi(t,\mathbf{x})|^2 \mathrm{d}\mathbf{x} \tag{28}$$

We can see that the PINNs prediction is able to predict the evolution of the system with a relatively good accuracy on J and ω, but for the two others, ϕ and ψ, involved in the kinetic and magnetic energy, the approximate seems to have a lower accuracy.

Figures 7, 8, 9 and 10 show the PINNs prediction for the different scalar fields compared to the real solutions at different time step, $t = 5, 6, 7.5$ and 9. These figures confirm that the PINNs have no difficulty to approximate the main variables (ω and J) but face greater difficulties to catch the dynamic of the other variables (ϕ and ψ).

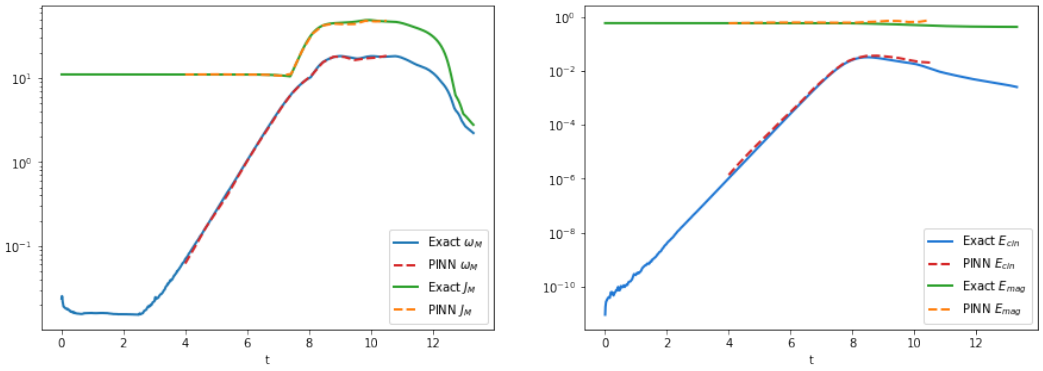

Figure 6. Comparison of the time history of the maximum current density amplitude and maximum vorticity amplitude (left panel) and the kinetic and magnetic energy (right panel) between the PINNs prediction and the data.

To conclude, PINNs can solve complex problems, like the tilt problem, without any mesh generation. This is a big advantage of PINNs over classical numerical methods. Indeed, in [**Baty, 2019**], author had to implement a highly adaptive characteristic-Galerkin scheme that use a finite element discretization using triangles with quadratic basis functions on an unstructured grid.

However, we see that PINNs struggle with multi-scale problem like the tilt problem. In these situation, tuning the many training hyperparameters is essential otherwise the PINNs will not be able to traing properly, but this a hard task, especially when training can take hours.

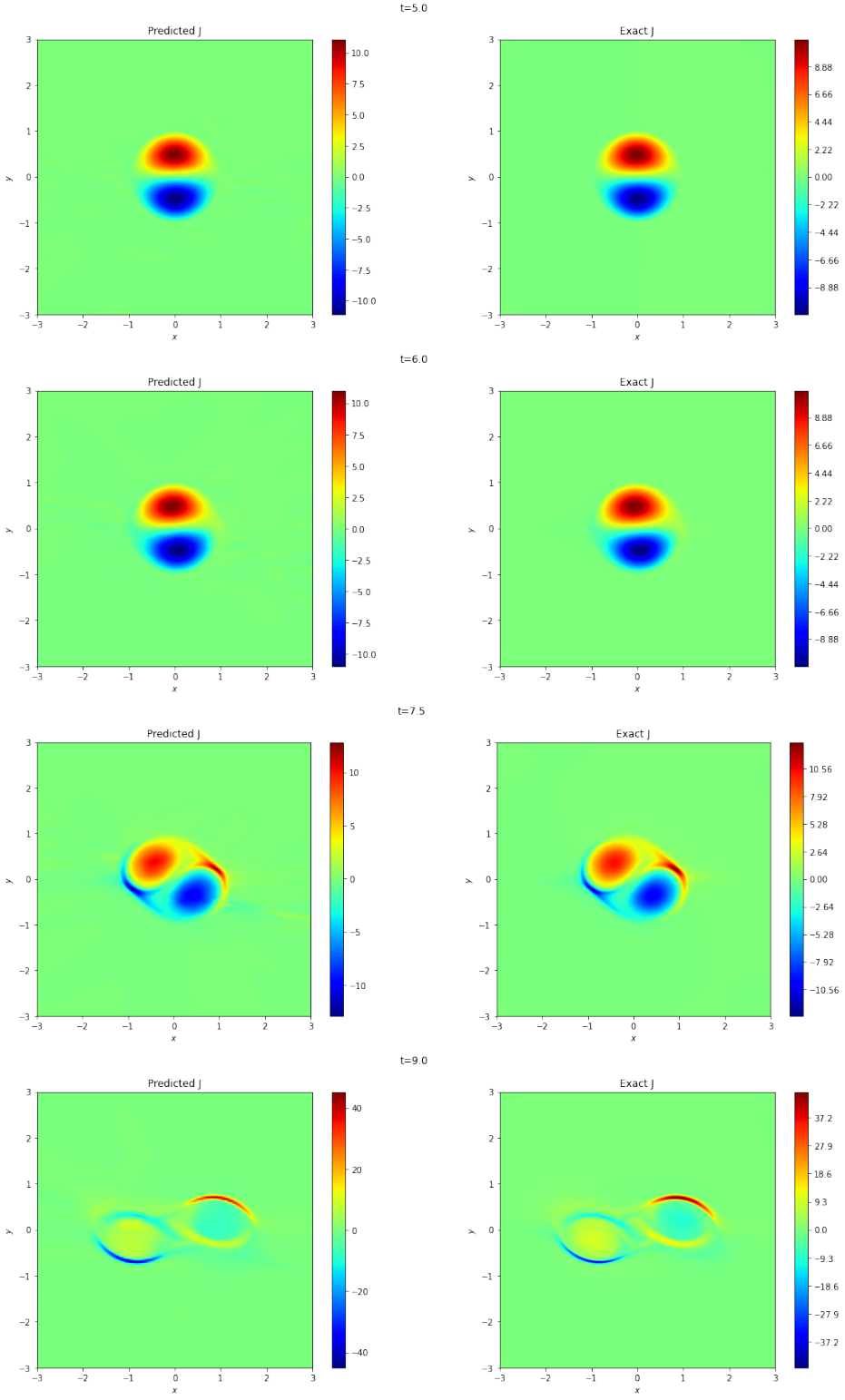

Figure 7. Current density predicted by PINNs compared to the exact solution at different times. $\eta = 3 \times 10^{-3}$.

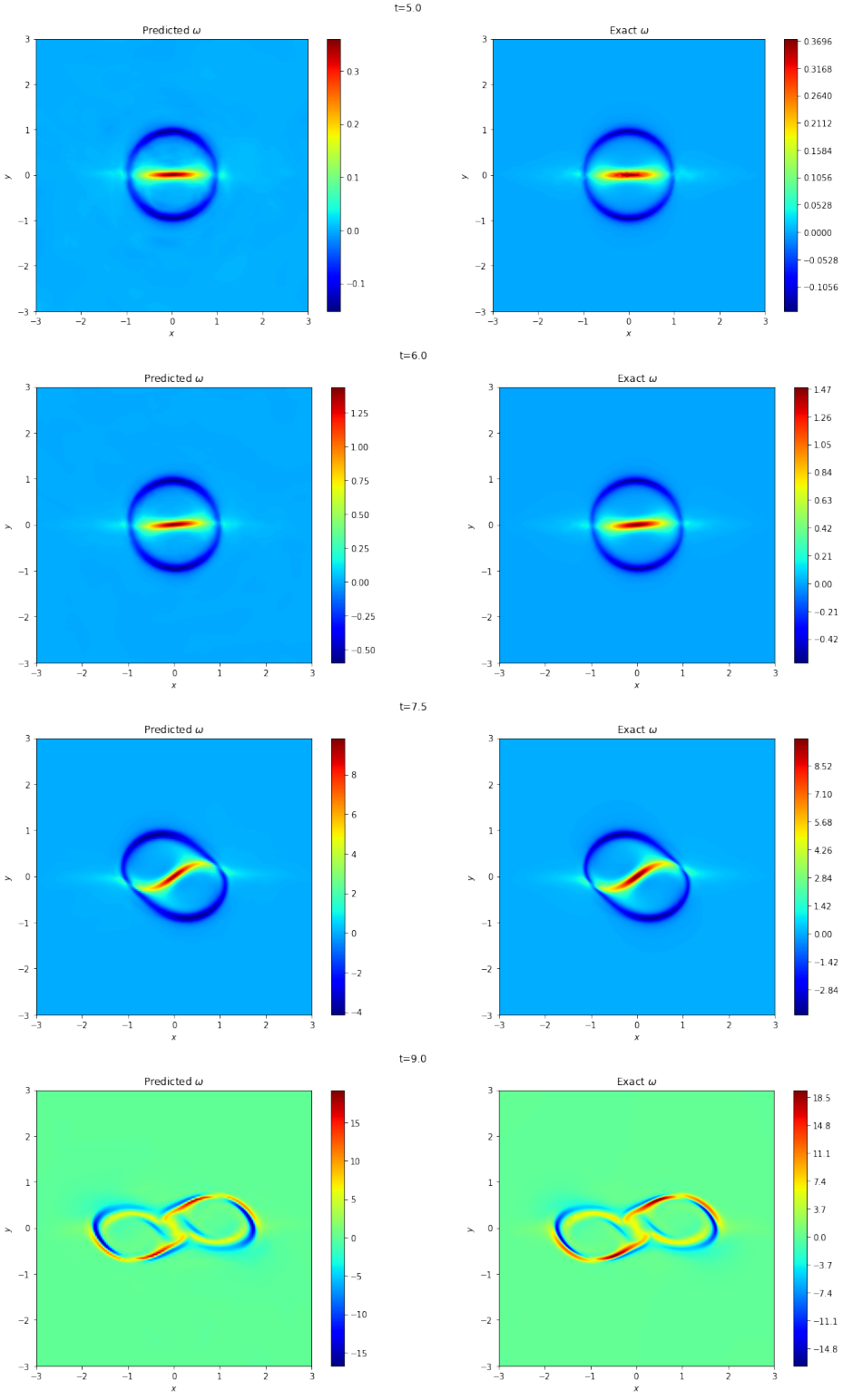

Figure 8. vorticity predicted by PINNs compared to the exact solution at different times. $\eta = 3 \times 10^{-3}$.

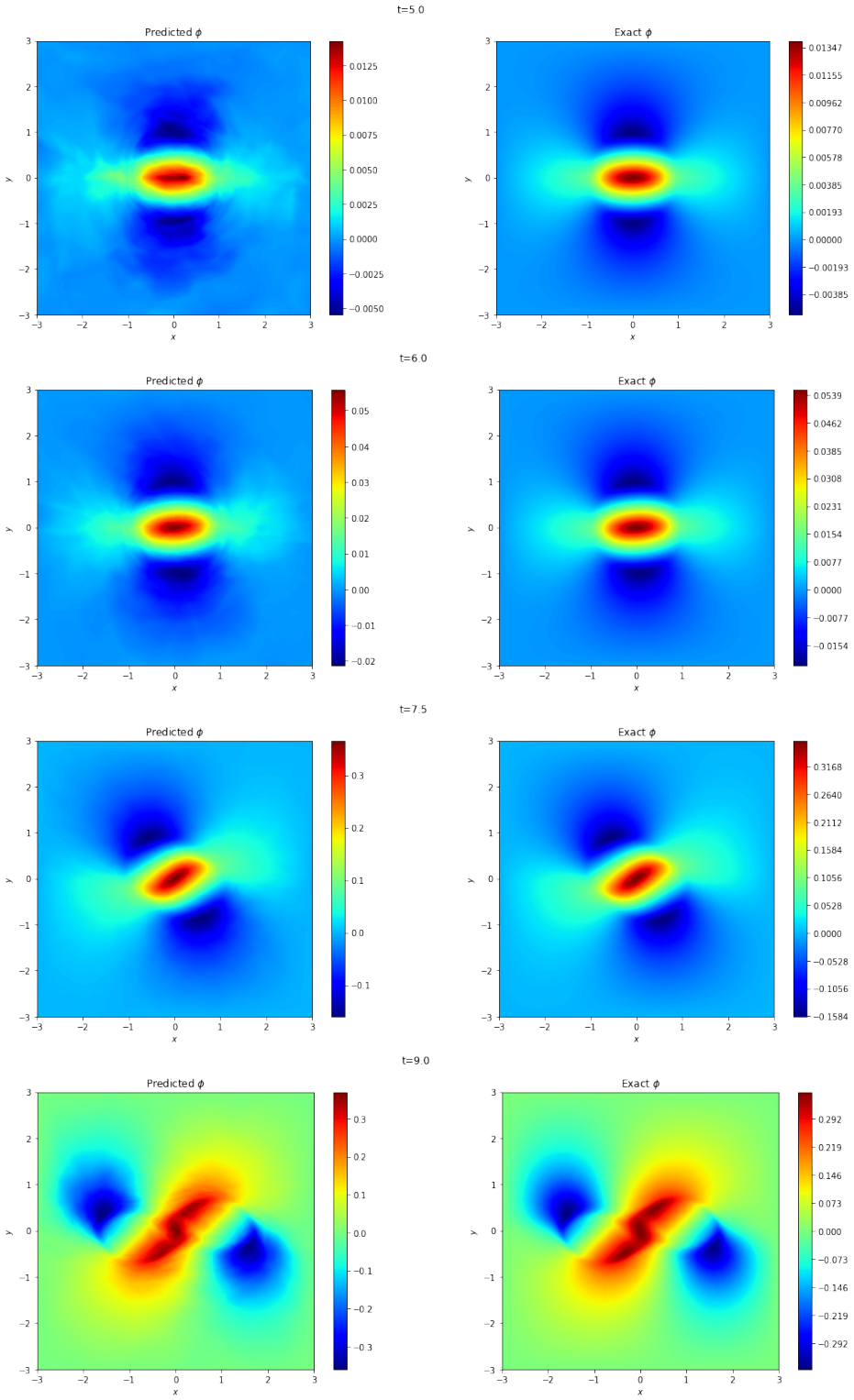

Figure 9. ϕ function predicted by PINNs compared to the exact solution at different times. $\eta = 3 \times 10^{-3}$.

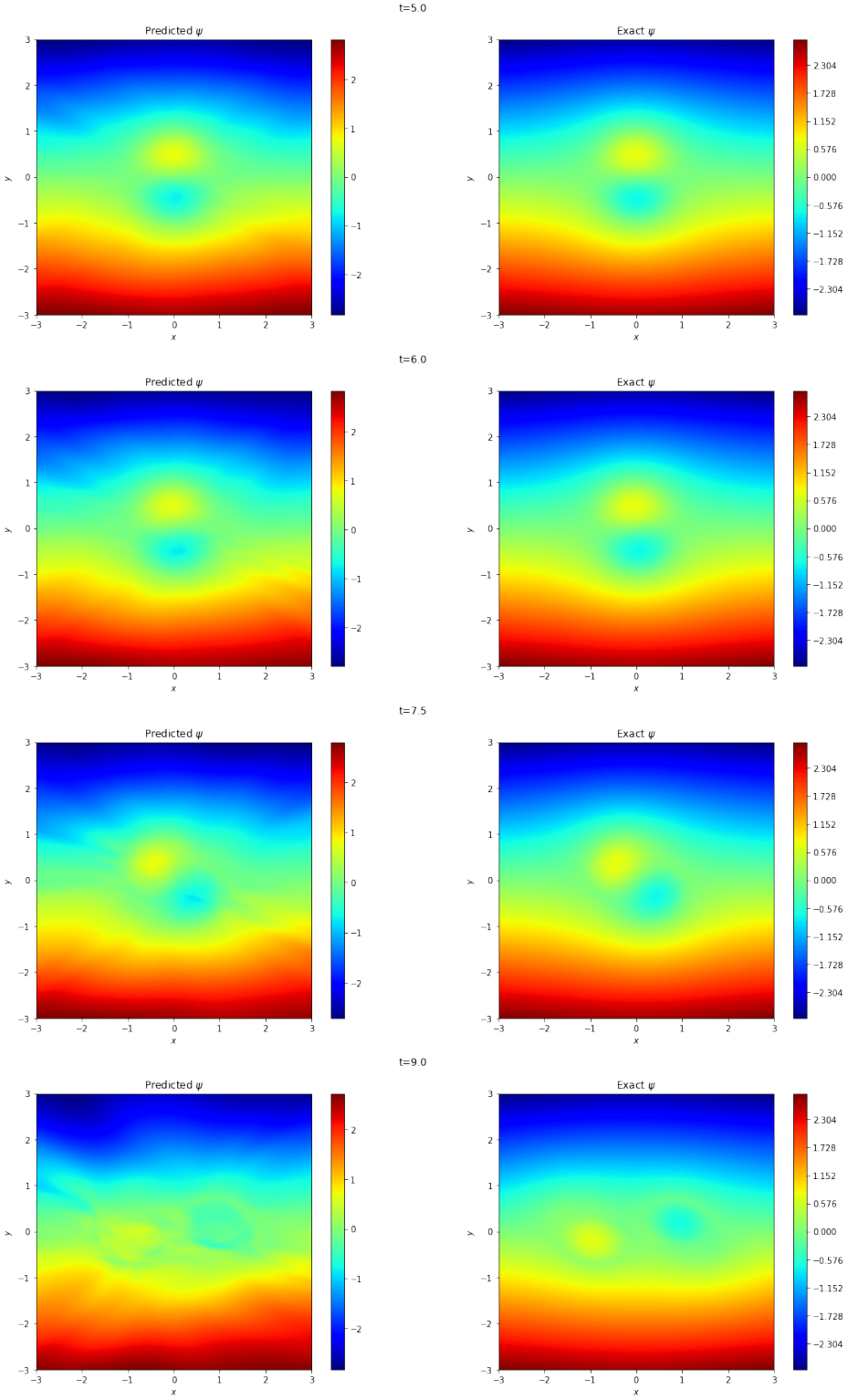

Figure 10. ψ function predicted by PINNs compared to the exact solution at different times. $\eta = 3 \times 10^{-3}$.

Bibliography

[Baty, 2019] Baty, H. (2019). FINMHD: An adaptive finite-element code for magnetic reconnection and formation of plasmoid chains in magnetohydrodynamics. https://arxiv.org/abs/1904.11173.

[Baty, 2022] Baty, H. (2022). Reconnexion magnétique et instabilités magnétohydrodynamiques dans les plasmas. https://hal.archives-ouvertes.fr/hal-03711796. working paper or preprint.

[Chen et al., 2021] Chen, W., Wang, Q., Hesthaven, J. S., and Zhang, C. (2021). Physics-informed machine learning for reduced-order modeling of nonlinear problems. https://www.sciencedirect.com/science/article/pii/S0021999121005611.

[Cuomo et al., 2022] Cuomo, S., di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., and Piccialli, F. (2022). Scientific machine learning through physics-informed neural networks: Where we are and what's next. https://arxiv.org/abs/2201.05624.

[Haghighat et al., 2021] Haghighat, E., Raissi, M., Moure, A., Gomez, H., and Juanes, R. (2021). A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. https://www.sciencedirect.com/science/article/pii/S0045782521000773.

[Krishnapriyan et al., 2021] Krishnapriyan, A. S., Gholami, A., Zhe, S., Kirby, R. M., and Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. https://arxiv.org/abs/2109.01050.

[LeCun et al., 2015] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. https://www.nature.com/articles/nature14539.

[Lu et al., 2021] Lu, L., Meng, X., Mao, Z., and Karniadakis, G. (2021). Deepxde: A deep learning library for solving differential equations. https://epubs.siam.org/doi/pdf/10.1137/19M1274067.

[Moseley et al., 2021] Moseley, B., Markham, A., and Nissen-Meyer, T. (2021). Finite basis physics-informed neural networks (fbpinns): a scalable domain decomposition approach for solving differential equations. https://arxiv.org/abs/2107.07871.

[Nabian et al., 2021] Nabian, M. A., Gladstone, R. J., and Meidani, H. (2021). Efficient training of physics-informed neural networks via importance sampling. https://arxiv.org/abs/2104.12325.

[Raissi et al., 2017] Raissi, M., Perdikaris, P., and Karniadakis, G. E. (2017). Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. https://arxiv.org/abs/1711.10561.

[Rao et al., 2020] Rao, C., Sun, H., and Liu, Y. (2020). Physics informed deep learning for computational elastodynamics without labeled data. https://arxiv.org/abs/2006.08472.