Scientific machine learning: applications

J. Aghili[2],H. Barucq[3], L. Bois[12], E. Franck[1], F. Foucher[3],
V. Michel Dansac[12], L. Navoret[12],N. Victorion[3], V. Vigon[12]

Numerical Analysis and EDP Seminar of Rennes University

[1]Inria TONUS Nancy Grand Est, France
[2]IRMA, Strasbourg university, France
[2]INRIA Makutu Bordeaux Sud-Ouest

Outline

Deep learning for plasma physics modeling

Plasma Vlasov equation

- **Plasma**: hot ionized gas. Plasmas are sensitive to electric and magnetic field.
- **Model**: Vlasov equation

$$\partial_t f + \mathbf{v} \cdot \nabla_{\mathbf{x}} f + \mathbf{E}(\mathbf{x}) \nabla_{\mathbf{v}} f = \frac{1}{\epsilon}(M - f) \tag{1}$$

with $f(t, \mathbf{x}, \mathbf{v}) \in \mathbb{R}^+ \times \mathbb{R}^{2d}$, $\mathbf{E}(\mathbf{x})$ the electric field and

$$M = \frac{\rho(\mathbf{x})}{2\pi T(\mathbf{x})} e^{-\frac{(\mathbf{v}-\mathbf{u}(\mathbf{x}))^2}{2T(\mathbf{x})}}$$

with

$$\rho = \int_\Omega f d\mathbf{v}, \quad \rho\mathbf{u} = \int_\Omega f\mathbf{v} d\mathbf{v}, \quad \rho T = \int_\Omega f(\mathbf{v}-\mathbf{u})^2 d\mathbf{v}$$

- Very costly models to solve.
- **Reduced model**: fluid macroscopic model.

$$\int_\Omega (1) d\mathbf{v} \rightarrow \partial_t \rho + \nabla \cdot (\rho\mathbf{u}) = 0$$
$$\int_\Omega (1)\mathbf{v} d\mathbf{v} \rightarrow \partial_t \rho\mathbf{u} + \nabla \cdot (\rho\mathbf{u} \otimes \mathbf{u} + \rho T I_d + \Pi) = -\rho E$$
$$\int_\Omega (1)\frac{1}{2}\mathbf{v}^2 d\mathbf{v} \rightarrow \partial_t E + \nabla \cdot (E\mathbf{u} + p\mathbf{u} + \Pi\mathbf{u} + Q) = -\rho E\mathbf{u}$$

with the stress tensor and the heat flux:

$$\Pi + \rho T I_d = \int_\Omega f(\mathbf{v}-\mathbf{u}) \otimes (\mathbf{v}-\mathbf{u}) d\mathbf{v}, \quad Q = \int_\Omega f(\mathbf{v}-\mathbf{u})^3 d\mathbf{v}$$

Closure problem

- Euler equations:

$$\begin{cases} \partial_t \rho + \nabla \cdot (\rho \mathbf{u}) = 0 \\ \partial_t \rho \mathbf{u} + \nabla \cdot (\rho \mathbf{u} \otimes \mathbf{u} + \rho T I_d + \Pi) = -\rho E \\ \partial_t E + \nabla \cdot (E\mathbf{u} + p\mathbf{u} + \Pi \mathbf{u} + Q) = -\rho E \mathbf{u} \end{cases}$$

- but we have three equations and five unknowns. How solve the system ?
- In 1D, $\Pi = 0$ we need to determine only Q.

Closure

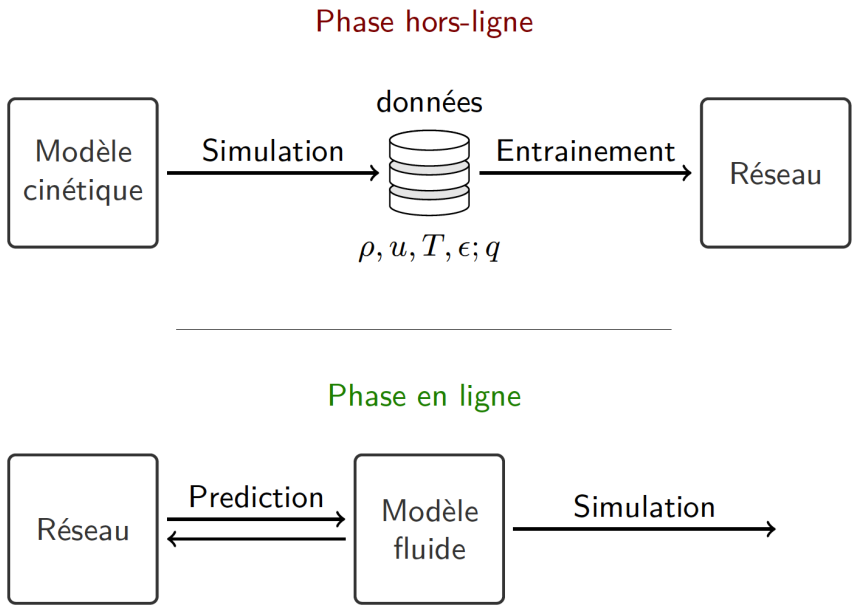
- Principle of closure:

$$\mathbf{\Pi}(f) \approx \hat{\mathbf{\Pi}}(\epsilon, \rho, \mathbf{u}, T), \quad \mathbf{Q}(f) \approx \hat{\mathbf{Q}}(\epsilon, \rho, \mathbf{u}, T)$$

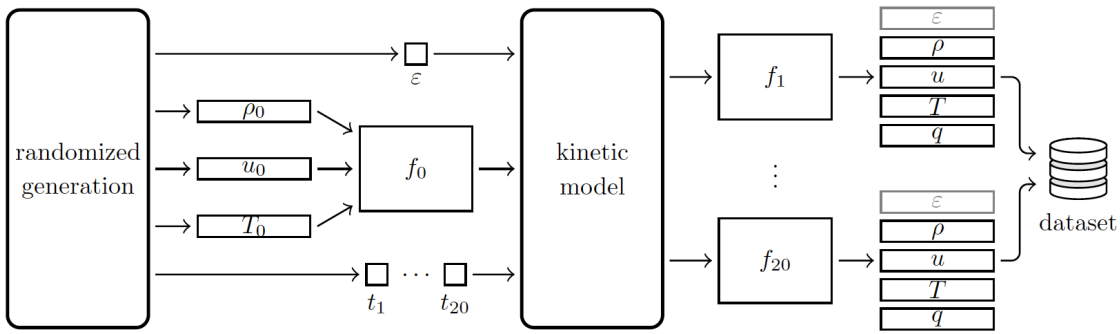
- **Strategy 1**: local closure obtain by asymptotic analysis for $\epsilon \ll 1$.
- **Strategy 2**: nonlocal closure which capture some specific effects associated with intermediate values of ϵ.
- **Strategy 3**: nonlocal/local closure obtained by supervised learning.

Learning closure I

- General priciple:



- Production of the data set



Learning closure II

Neural network

We use a CNN of the form:

$$q_h = CNN_\theta(\rho_h, u_h, T_h, \epsilon_h)$$

where we take as input the variables on all the mesh and gives the heat flux on all the mesh.

- Which neural network: Unet.

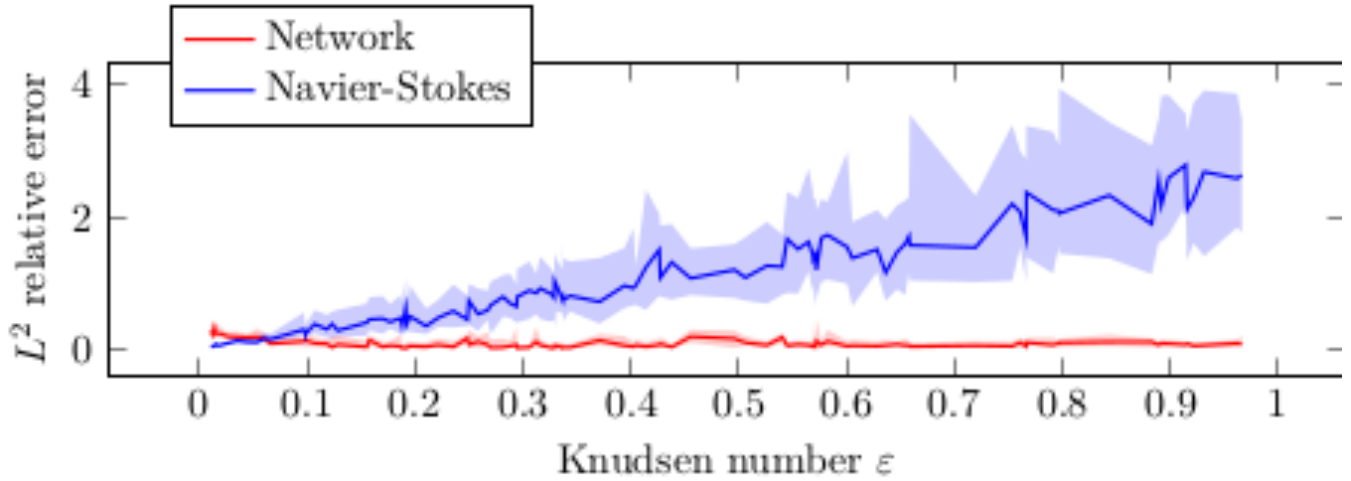
Hyper-paramètres	Valeur
taille de l'entrée	512
nombre de niveau (ℓ)	5
profondeur initiale (d)	4
taille des noyaux (p)	11
fonction d'activation	softplus

Nombre total de paramètres: $161\,937$

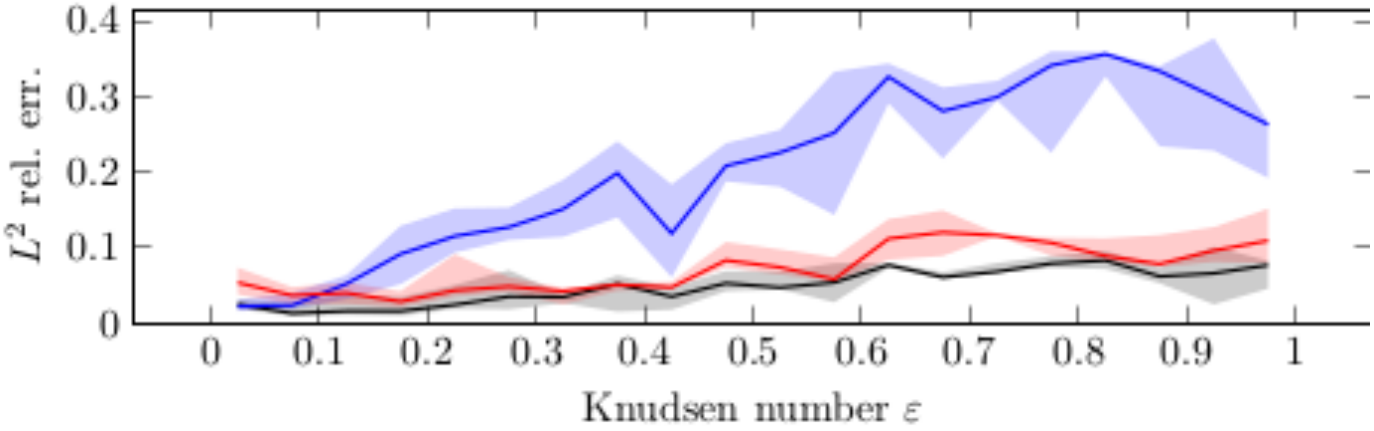
- **Training time**: around 30-40 minutes
- **How deal with other grid**: interpolation/projection to call the CNN on the training grid size.

Results I

- We compare q given by the NN, $q = -\epsilon \frac{3}{2} p \partial_x T$ of the Navier-Stokes closure and the q compute by a kinetic code.

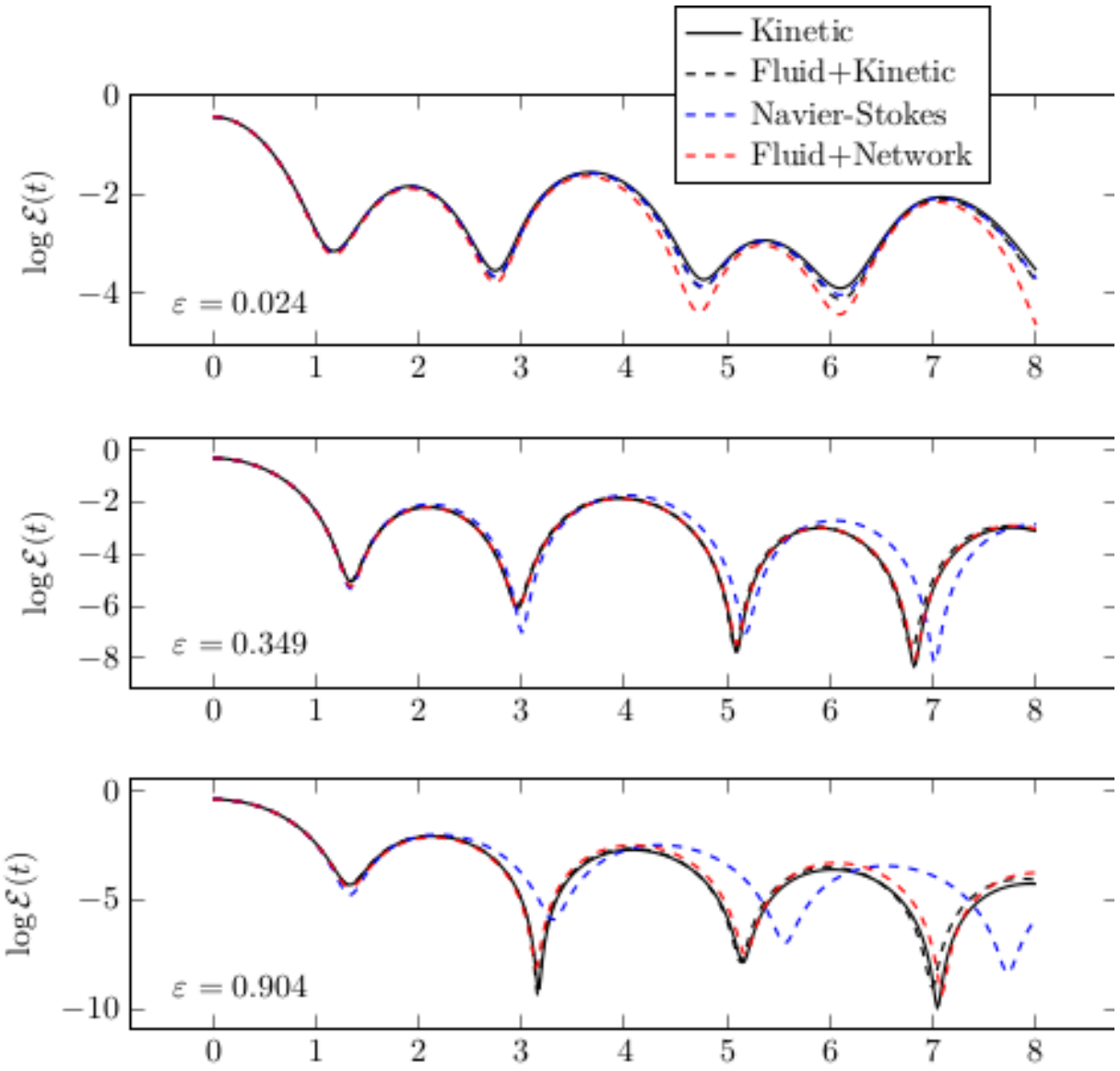


- We compare a kinetic code, fluid code with NN, with NS and with the q computed by the kinetic.



Results II

- Different simulations:



Neural operator and elliptic equations

Nonlinear elliptic problems and Newton's method

- We want to solve the following elliptic problem:

$$u - \alpha_0 \nabla \cdot (A(x,y)k(u)\nabla u) = f(x,y).$$

- It also corresponds to the implicit part of a diffusion equation.

Solver

- Finite difference or Finite element (here on structured meshes)
- Newton-Krylov method (Jacobian-free approximation + GMRES for linear part)

- After discretization, we solve the problem:

$$G_{A_h,f_h,\alpha_0}(\mathbf{u}_h) = 0$$

with $\mathbf{u}_h$ a discretization of $u(x)$.

Difficulties

- The more the equation is non-linear, the more the Newton convergence is difficult.
- The more $A(x,y)$ is anisotropic and $\alpha_0 \gg 1$, the more the condition number is large and the convergence (linear/nonlinear) is hard.

Initial guess and FNO

Convergence of Newton's method

The convergence depends on the initial guess. If the initial guess is too far from the solution, Newton's method converges slowly, or even does not converge.

Idea

Train a Fourier Neural Operator (FNO) to approximate the solution of the elliptic equation and use it as an initial guess.

- We keep the convergence properties of the scheme, and we hope to accelerate Newton's method.
- Algorithm:
  1. Fix a mesh, fix α_0 and $k(\cdot)$,
  2. Randomly generate many data u_h^i, A_h^i (random Fourier coefficients, sum of random Gaussian functions),
  3. Compute the right-hand side associated with f_h^i,
  4. Train the neural network $G_\theta^+(A_h^i, f_h^i)$, by minimizing

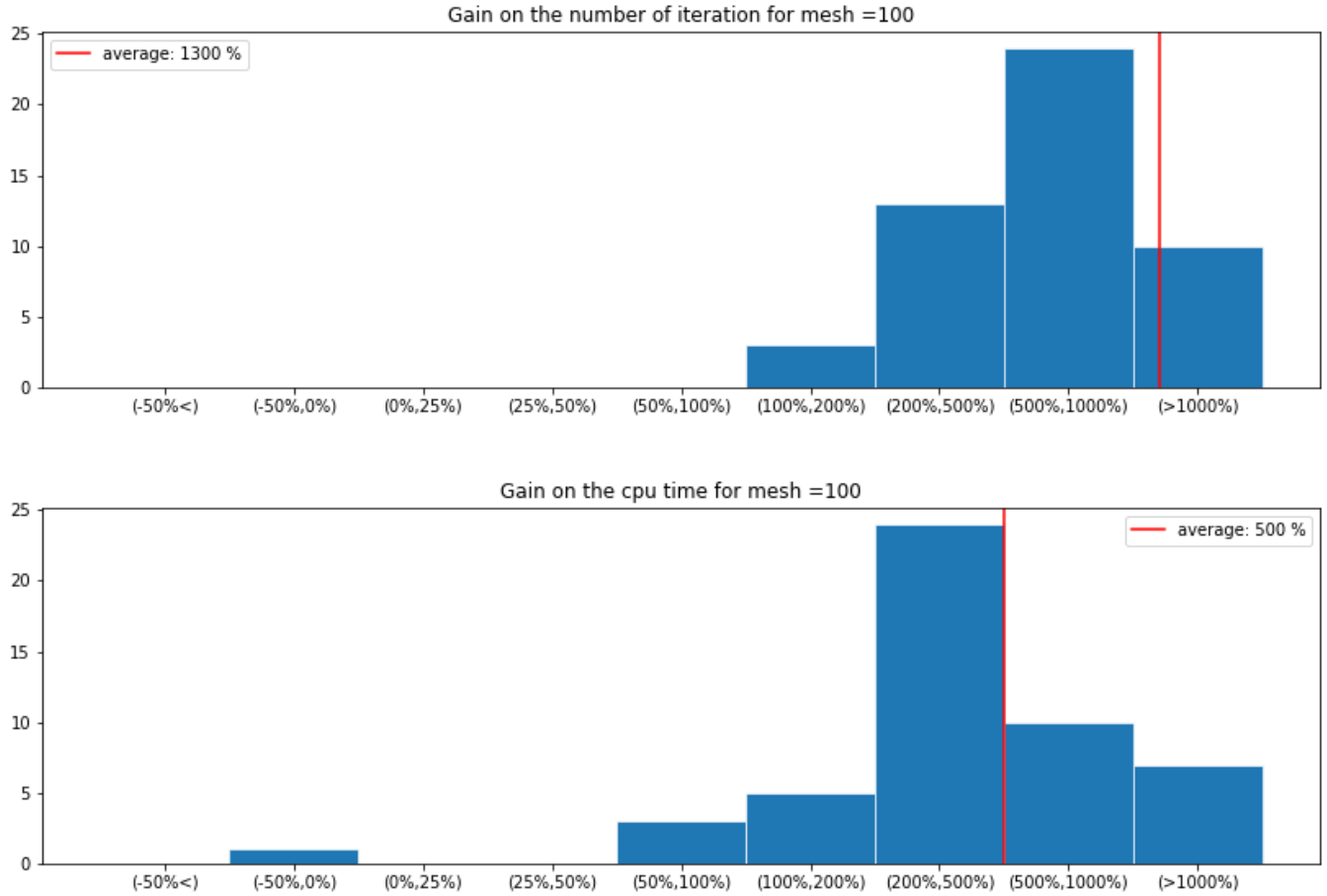
$$J(\theta) = \omega \sum_{i=1}^{n} \| G_\theta^+(A_h^i, f_h^i) - u_h^i \| + (1-\omega) \sum_{i=1}^{n} \| G_{A_h^i, f_h^i, \alpha_0}(G_\theta^+(A_h^i, f_h^i)) \|.$$

Results I

- We consider the equation

$$u(x) - \partial_x(2\alpha(x)u^4\partial u) = f$$

- We learn only with the data loss and the residue loss.
- Ratio number of iterations (top) and CPU time (bottom) for classical/NN on 100 cells

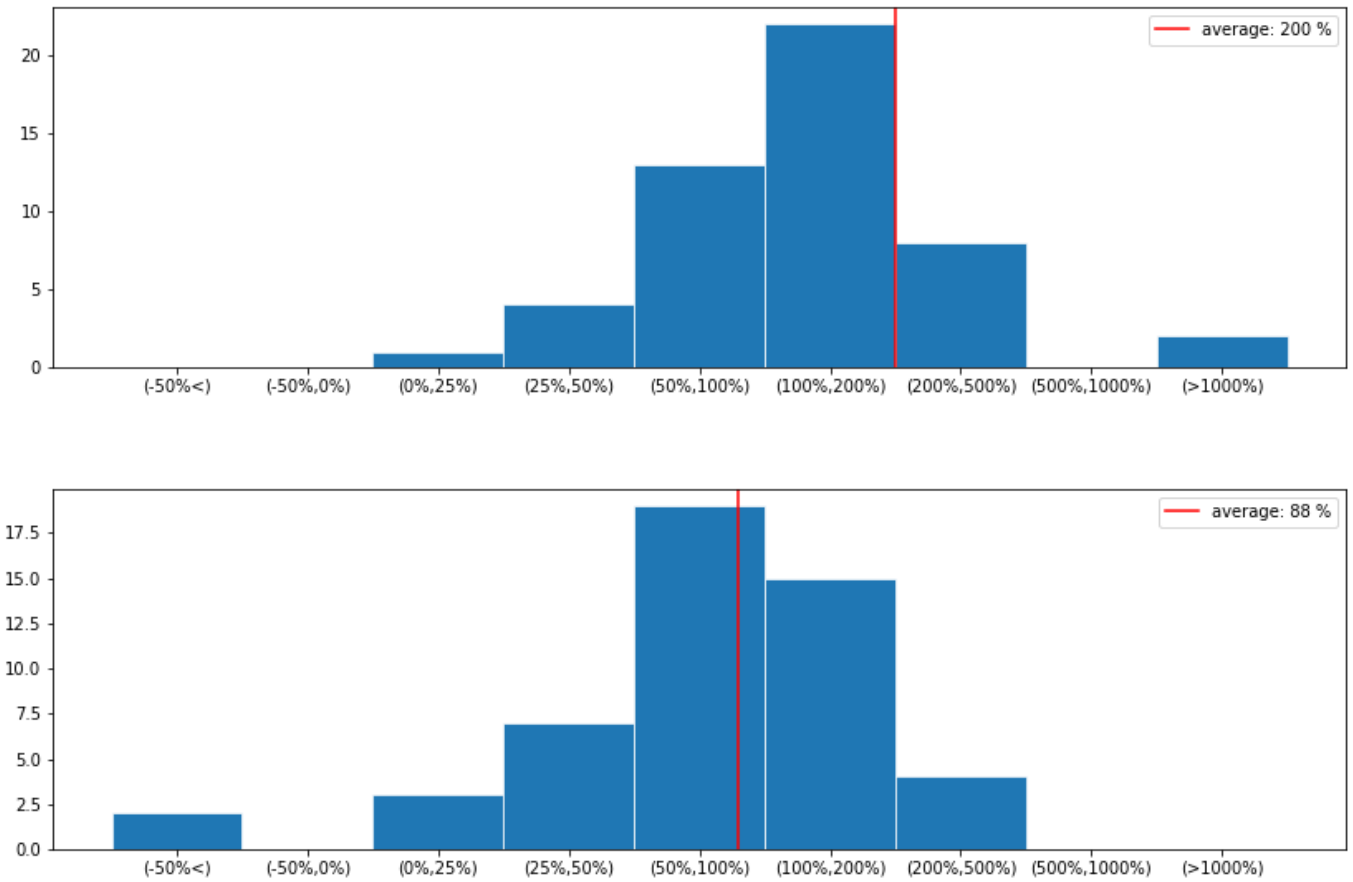


Results I

- We consider the equation

$$u(x) - \partial_x(2\alpha(x)u^4\partial u) = f$$

- We learn only with the data loss and the residue loss.
- Ratio number of iterations (top) and CPU time (bottom) for classical/NN on 200 cells

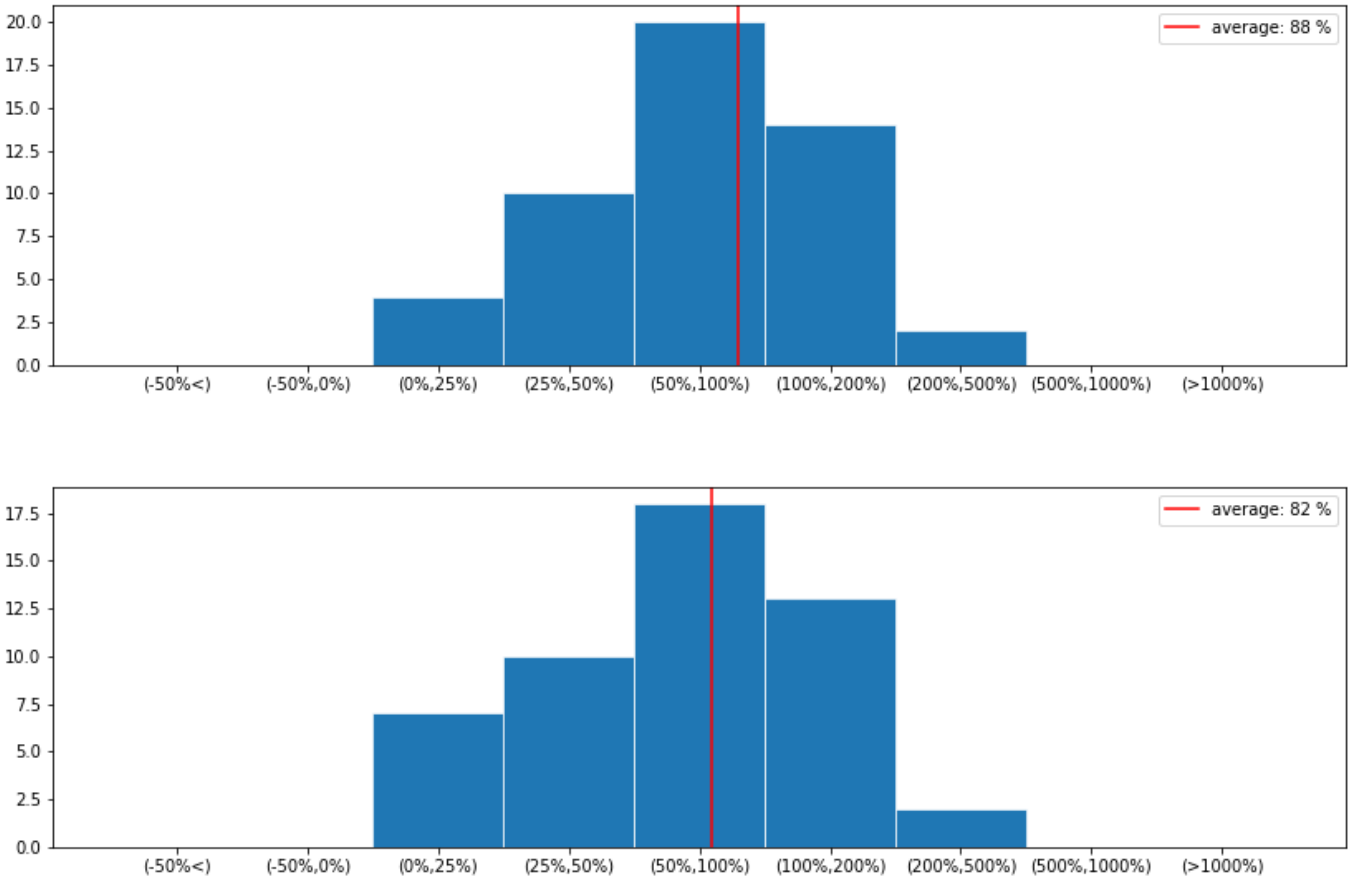


Results I

- We consider the equation

$$u(x) - \partial_x(2\alpha(x)u^4\partial u) = f$$

- We learn only with the data loss and the residue loss.
- Ratio number of iterations (top) and CPU time (bottom) for classical/NN on 400 cells

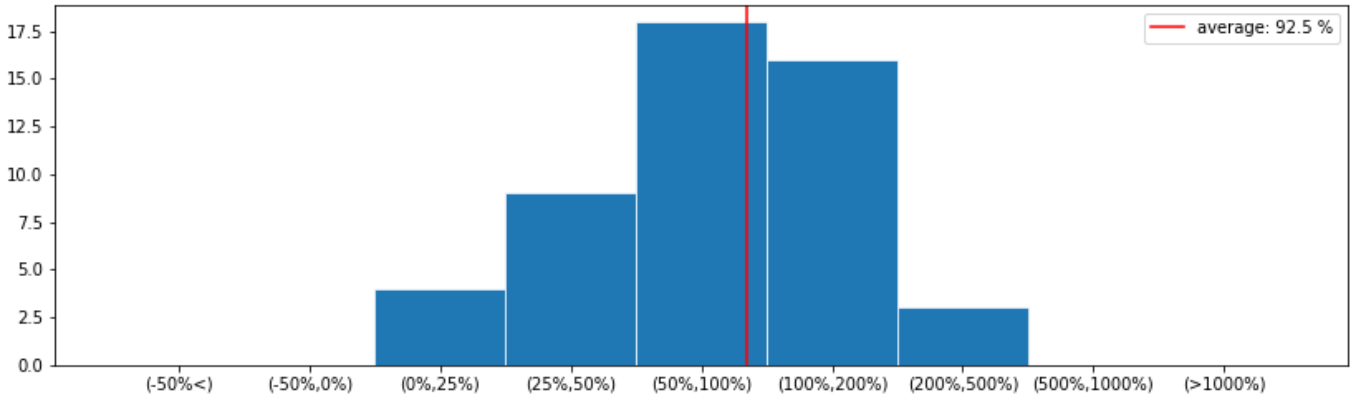


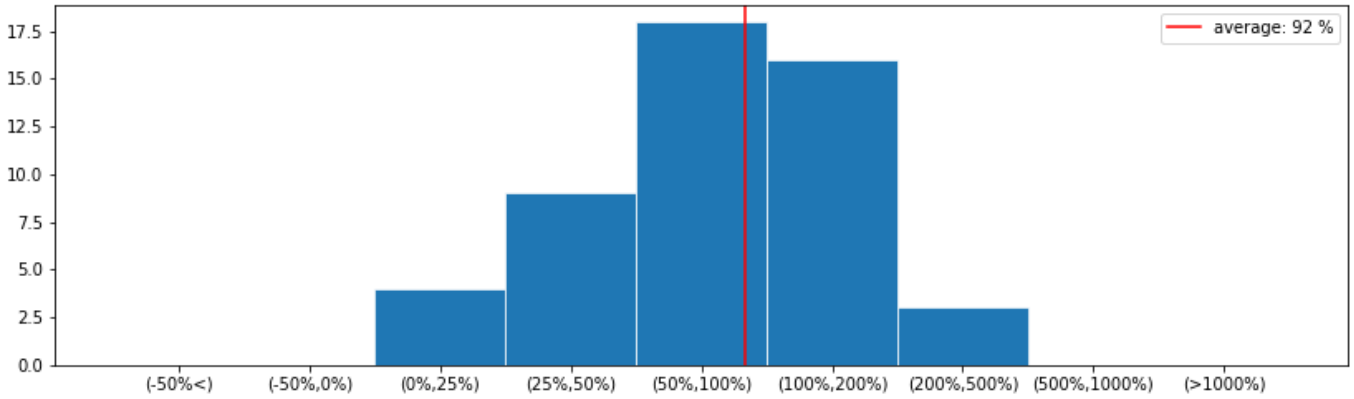
Results I

- We consider the equation

$$u(x) - \partial_x(2\alpha(x)u^4\partial u) = f$$

- We learn only with the data loss and the residue loss.
- Ratio (nb iter Newton/nb iter Newton +NN) on 600 cells





Results II

- What happens when we increase α_0 to get a stronger non-linearity?
- We only compare the average results:

mesh	$\alpha_0 = 2$ (40 sim)	$\alpha_0 = 5$ (25 sim)	$\alpha_0 = 8$ (25 sim)
100 cells	+500%	+1800%	+5000%
200 cells	+88%	+230%	+620%
400 cells	+82%	+150%	+220%
600 cells	+92%	+220%	+250%

Table: Comparison of the mean "gain" for different values of α_0.

- **Fails**: on all the tests, we have 0% of fail (our method being less efficient than the classical one) for the iteration criterion, and around 2% of fail on the CPU time criterion.
- On more refined meshes, the gain is smaller (the network acts only at the beginning of the convergence).
- More the system is nonlinear more the method is efficient.

Enhanced numerical methods with PINNs

Pinns vs Finite element

- Parametric problem:

$$\begin{cases} -\Delta u = \alpha \sin(2\pi x) + \beta \sin(4\pi x) + \gamma \sin(8\pi x) \\ u = 0 \end{cases}$$

- Solving that with PINNs we obtain an error comparable to 80 cells finite elements.
- **Problems**: no guarantees and in a few time some fails. **Advantages**: after the training, very fast to have a solution for new (α, β, γ).

Idea

See the PINNs as a prior to the solution and plug in it the numerical method (here FE) to increase the accuracy.

- Strong form $\rightarrow$ weak form:

$$-\Delta u = f \qquad \rightarrow \qquad \int_\Omega \nabla u \cdot \nabla v dx = \int_\Omega f v dx, \quad v \in H_0^1(\Omega)$$

- We take $u = u_h = \sum_{j=0}^{N} \alpha_j \phi_j(x)$ and $v = \phi_i(x)$ and plug it in the weak form to obtain

$$A\alpha = b, \quad A_{ij} = \int_\Omega \nabla \phi_j \cdot \nabla \phi_i, \quad b_i = \int_\Omega f \phi_i$$

Finite element

- Definition of **finite element**: triplet $\{K, P, \Sigma\}$ where
  - K is compact Lipschitz subset of $\mathbb{R}^d$,
  - $P \subset V_K$, with V_K is a vectorial space of function $\phi : K \to \mathbb{R}^d$
  - Σ is a set of q linear forms acting on P such that the linear mapping:

$$G : P \to (\sigma_1(\phi), ..., \sigma_q(\phi)) \in \mathbb{R}^q$$

  is bijective. The linear forms $\{\sigma_1, ..., \sigma_q\}$ are called **local degrees of freedom**
- A finite element is P-unisolvant if each element of P is uniquely determined by the local degrees of freedom.
- The basis $\{\phi_1, ..., \phi_q\}$ of P which satisfy

$$\sigma_i(\phi_j) = \delta_{ij}, \quad 1 \leq ij, j \leq q$$

  is called **local shape function**.
- The local interpolation operator $\mathcal{I}_K$ is defined by:

$$\forall f \in V_K, \qquad \mathcal{I}_K f(x) = \sum_{i=1}^{q} \sigma_i(f)\phi_i(x)$$

- Imposing the continuity between **the local shape function** we obtain the **global shape functions** which form a basis of $V_h \in H_1(\Omega)$

Enhanced Finite element

Classical P_1 element

We choose the value at the node as DoF:

- $K = [x_1, x_2]$
- $\mathbb{P}_1([x_1, x_2])$
- $\Sigma_{\theta,K} = \{\bar{\sigma}_1, \bar{\sigma}_2\}$.
- The local shape functions:

$$\phi_1(x) = \frac{x_2 - x}{x_2 - x_1}, \quad \phi_2(x) \frac{x - x_1}{x_2 - x_1}$$

Modified P_1 element

We consider $u_\theta(x; \boldsymbol{\mu}) \in C^1(\mathbb{R})$.

- $K = [x_1, x_2]$
- $P_{\theta,K} = \left\{\bar{\phi} = \phi u_\theta, \quad \phi \in \mathbb{P}_1([x_1, x_2])\right\}$
- $\Sigma_{\theta,K} = \{\bar{\sigma}_1, \bar{\sigma}_2\}$ with $\forall \bar{\phi} \in P_{\theta,K}, \quad \bar{\sigma}_i(f) = \frac{f(x_i)}{u_\theta(x_i)}, \quad 1 \leq i \leq 2$
- The local shape functions:

$$\bar{\phi}_1(x) = \phi_1(x) u_\theta(x), \quad \bar{\phi}_2 = \phi_2(x) u_\theta(x)$$

with ϕ_1, ϕ_2 the P_1 local shape functions.

Theory

- Discrete problem:

$$\begin{cases} \text{Seek } u_h \in V_h \in H^1(\Omega), & \text{such that} \\ a(u_h, v_h) = b(v_h), \forall v_h \in V_h \end{cases}$$

with a_h a discrete bilinear form, b_h a discrete linear form, $V_h \in H^1(\Omega)$

- The Cea lemma allow to obtain the following estimation of the error:

$$\| u - u_h \|_{H^m} \leq \frac{M}{\alpha} \| u - v_h \|_{H^m} \leq C \| u - \mathcal{I}_h u \|_{H^m} \quad \forall v_h \in V_h$$

since the interpolator $\mathcal{I}_h u \in V_h$.

Proposition

$$\mathcal{I}_h(u) = \bar{\mathcal{I}}_h(\frac{u}{u_\theta(x)}) u_\theta(x)$$

with $\bar{\mathcal{I}}_h$ the classical P_1 interpolation. So

$$\|u - \mathcal{I}_h(u)\|_{H^m} \leq \| \frac{u}{u_\theta} - \bar{\mathcal{I}}(\frac{u}{u_\theta(x)}) \|_{H^m} \|u_\theta(x)\|_{H^m}$$

- We use the results of the classical interpolation to conclude.

Convergence

$$\|u - \mathcal{I}_h(u)\|_{H^m} \leq C h^{2-m} \left(| \frac{u}{u_\theta(x)} |_{H^2} \frac{\|u_\theta(x)\|_{H^m}}{\|u\|_{H^m}} \right) \|u\|_{H^m}.$$

NN-enhanced finite element solver: results

- First test:

$$-\partial_{xx} u = \alpha \sin(2\pi x) + \beta \sin(4\pi x) + \gamma \sin(8\pi x)$$

We train with $(a, b, c) \in [0, 1]^3$ and test with $(a, b, c) \in [0, 1.2]^3$.

Results with 20 cells:

Data set	inside training set	outside training set	inside/outside
Average gain vs FE	101	28.4	76.8
Average gain vs PINNS	6.7	7	6.8

- Second test:

$$v\partial_x u - \frac{v}{Pe}\partial_{xx} u = r$$

We train with $r \in [1.5, 2.0]$, $v \in [1.5, 2.0]$ and $Pe \in [10, 120]$,
and we test with $r \in [1.5, 2.2]$, $v \in [1.5, 2.2]$ and $Pe \in [10, 150]$.

Results with 20 cells:

Data set:	inside training set	outside training set	inside/outside
Average gain	110	25	81.6

- The method converges with second-order accuracy.
- **Next step, with Inria Pau**: 2D extension, extension to time problems.

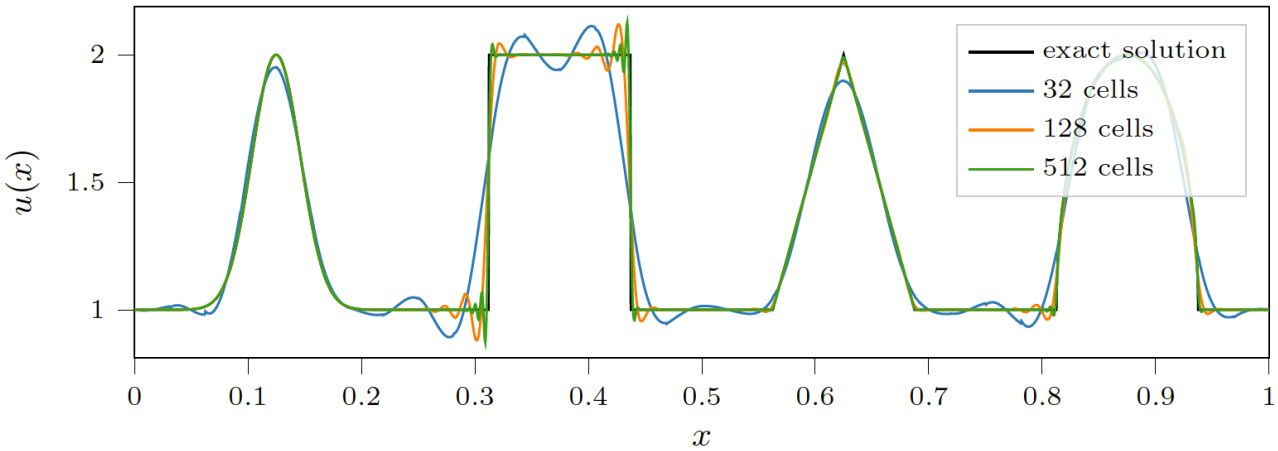
Application of differentiable physics

General problem

- We want to solve general hyperbolic PDEs:

$$\partial_t \mathbf{U} + \partial_x \mathbf{F}(\mathbf{U}) = 0$$

- High order method (MUSCL, HO finite volumes or DG) generate oscillations around areas with strong gradients or shock waves: Gibbs phenomenon.
- Example on the advection equation:



- **Solutions**: slope limiting, artificial viscosity, filtering, etc.

Goal

Design slope limiting for MUSCL or artificial viscosity for DG using neural networks.

Artificial viscosity problem for DG

- We have a DG scheme, written under the form

$$\partial_t^{rk} \mathbf{U}_h + \partial_x^{DG} \mathbf{F}(\mathbf{U}_h) = 0.$$

- **Artificial viscosity method**: add a diffusion operator, which acts on the oscillations.
- Modified scheme:

$$\partial_t^{rk} \mathbf{U}_h + \partial_x^{DG} \mathbf{F}(\mathbf{U}_h) = \partial_x^{DG} (\mathbf{D}(\mathbf{U}_h) \partial_x^{DG} \mathbb{U}_h).$$

- **How to construct D?**
- Derivative-based approach:

$$D(\mathbf{U}_h) = \lambda_{max} h |\partial_x^{DG} \mathbf{U}_h)|$$

- MDH approach: we reconstruct the modes within the cells, and apply viscosity to decrease the highest modes.
- Other approaches: MDA, entropy-based, etc.
- How to use neural networks? Approach from J. Hesthaven: compute the best viscosity on many test cases, and learn this viscosity with a NN.
- The NN interpolates between known viscosities.
  - There is no new viscosity model,
  - and we cannot use this method to tune a scheme where we do not have a prior viscosity model.

Differentiable physics approach I

Tool

We propose to use differentiable physics (control optimal approach) to design new types of viscosity model.

- Formalism of optimal control and RL.
- We define a NN $D_\theta(\mathbf{U}_h(t))$ with $\mathbf{U}_h(t)$ the discrete solution.
- We define a value function:

$$V_\theta^T(\mathbf{U}_0) = \int_0^T C(\mathbf{U}_h(t))dt,$$

with C a cost function and $\mathbf{U}_0 = \mathbf{U}_h(0)$ an initial condition.

Goal

Our objective to find a solution of the minimization problem:

$$\min_\theta \int_{U_0} V_\theta(\mathbf{U}_0)d\mathbb{P}(\mathbf{U}_0)d\mathbf{U}_0 \tag{2}$$

with $\mathbb{P}(\mathbf{U}_0)$ a probability law of initial data on $\mathbf{U}_0$.

Differentiable physics approach II

- After Monte-Carlo discretization, we obtain the minimization problem:

$$\min_{\theta} J(\theta) = \min_{\theta} \sum_{i=1}^{n_{\text{data}}} V_\theta^T(\mathbf{U}_{i,0}).$$

- We provide an approximation in time of the value function:

$$V_\theta^T(\mathbf{U}_0) = \Delta t \sum_{t=1}^{T} C(\mathbf{U}_h^t)$$

- The transition between two time steps is given by $\mathbf{U}_h^{n+1} = S_h(\mathbf{U}_h^n, D_\theta(\mathbf{U}_h^n))$ with our scheme. As a consequence, we have:

$$V_\theta^T(\mathbf{U}_0) = C(\mathbf{U}_0) + C(S_h(\mathbf{U}_0, D_\theta(\mathbf{U}_0))) + C(S_h(S_h(\mathbf{U}_0, D_\theta(\mathbf{U}_0)), D_\theta(S_h(\mathbf{U}_0, D_\theta(\mathbf{U}_0))))) + \dots,$$

- As previously mentioned in the paradigm of differential physics, we can compute by automatic differentiation:

$$\nabla_\theta V_\theta^T(\mathbf{U}_0)$$

- We solve the minimization problem on $J(\theta)$ using a gradient method, with

$$\nabla_\theta J(\theta) = \sum_{i=1}^{m} \nabla_\theta V_\theta^T(\mathbf{U}_{i,0})$$

Differentiable physics approach III

- To complete the algorithm, the NN and loss function still have to be defined.

Neural network

A ResNet convolution neural network (without coarsening operator) with q channels (polynomial order q); once trained, it can be used on arbitrary uniform grids, by sliding the convolution window.

Loss function

The cost function $C()$ is composed of three parts:

- L^2 error compared to a MUSCL solution on a fine grid:

$$C_{\text{error}}(\mathbf{U}_h^n) = h_{FV} \sum_{i=1}^{n} \|\Pi_{FV}(\mathbf{U}_h^n)_i - \mathbf{U}_{i,ref}\|_2^2 ,$$

- L^1 error on the Laplacian compared to the Laplacian of the reference solution

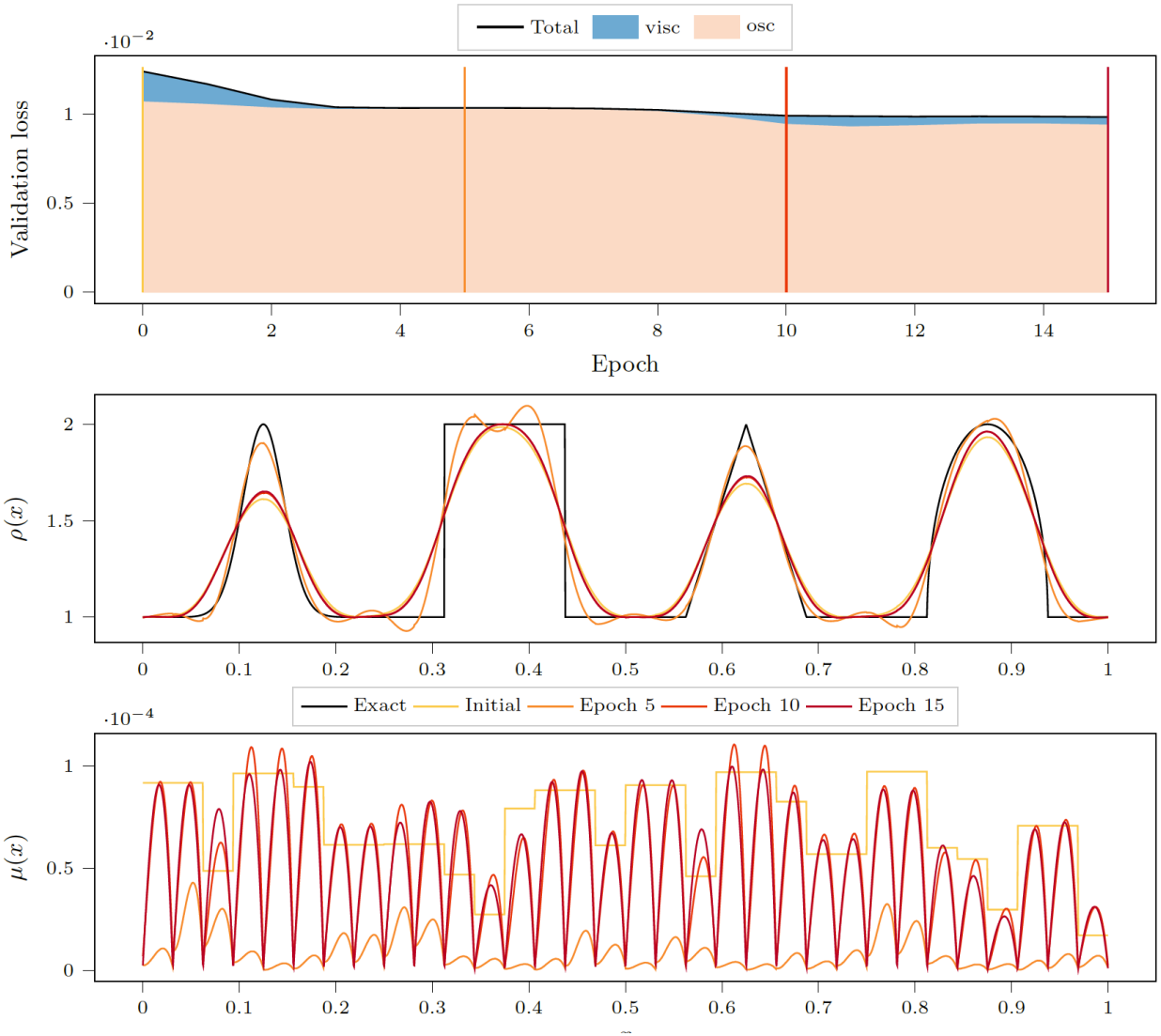
$$C_{\text{osc}}(\mathbf{U}_h^n) = h_{fv} \sum_{i=1}^{n} \left\| D_{xx}^{fv}(\Pi_{fv}(\mathbf{U}_h^n))_i - D_{xx}^{fv}\mathbf{U}_{j,ref} \right\|_1 .$$

- L^2 norm of D_θ:

$$C_{\text{vis}}(\mathbf{U}_h^n) = \|D_\theta(\mathbf{U}_h^n)\|_2^2 .$$

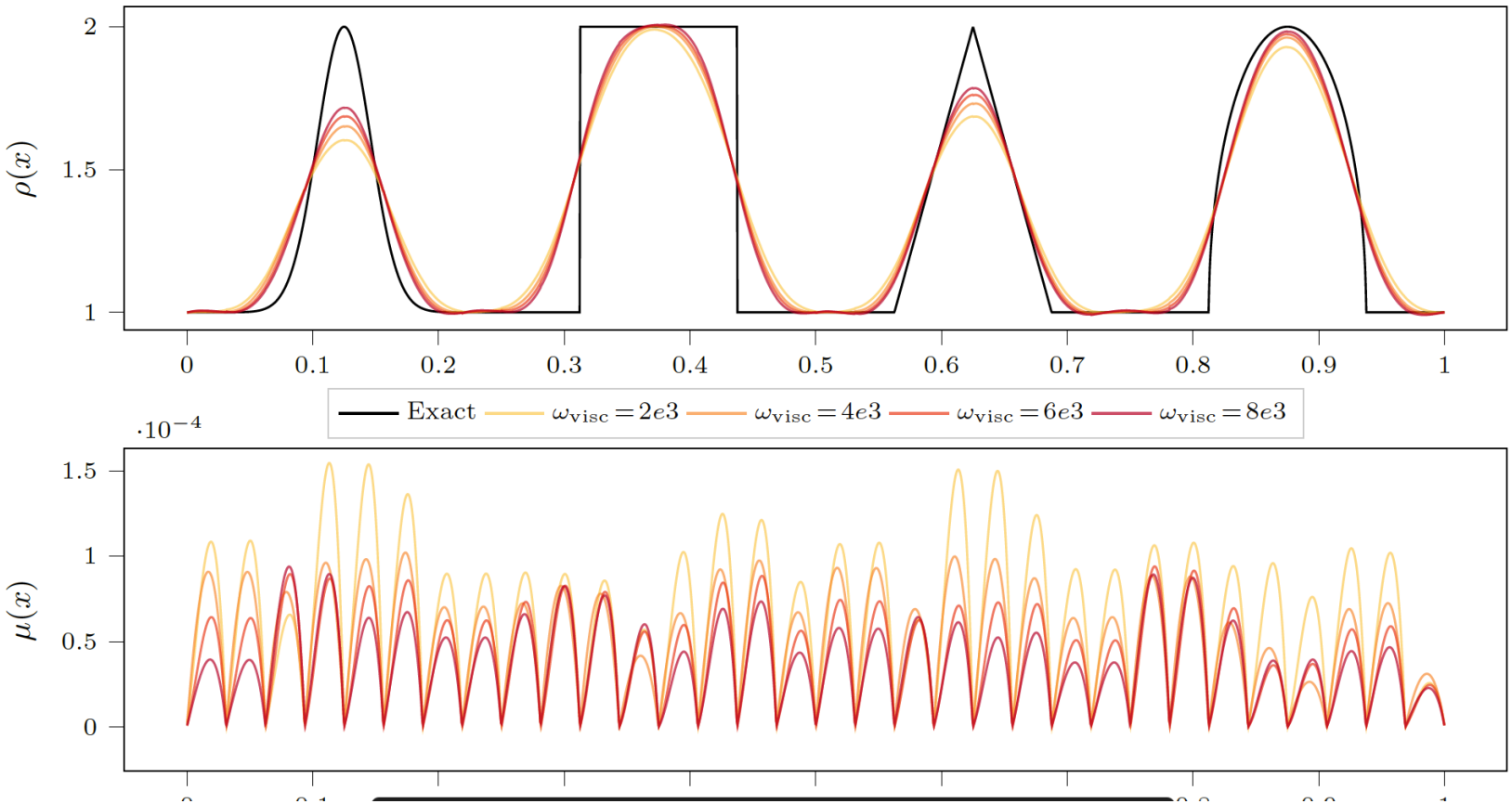
Results I

- We make a training with the loss "oscillation" and "viscosity".
- How the NN learn:



Results II

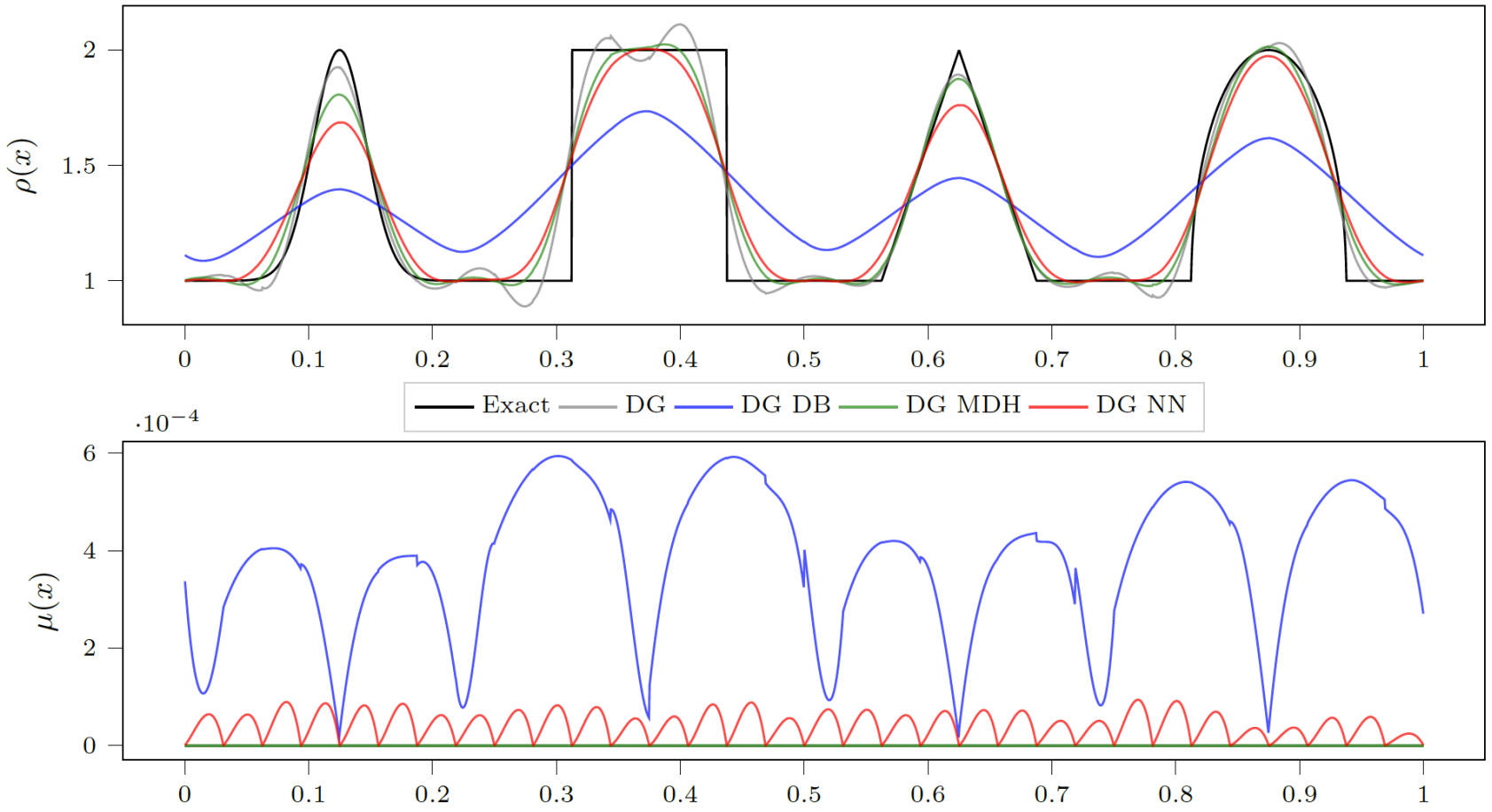
- We compare the training for different ratio loss.
- We fixe the weight of the oscillation loss.



- The final result is mainly related to this ratio.
- The train stability around a error which depends of this ratio

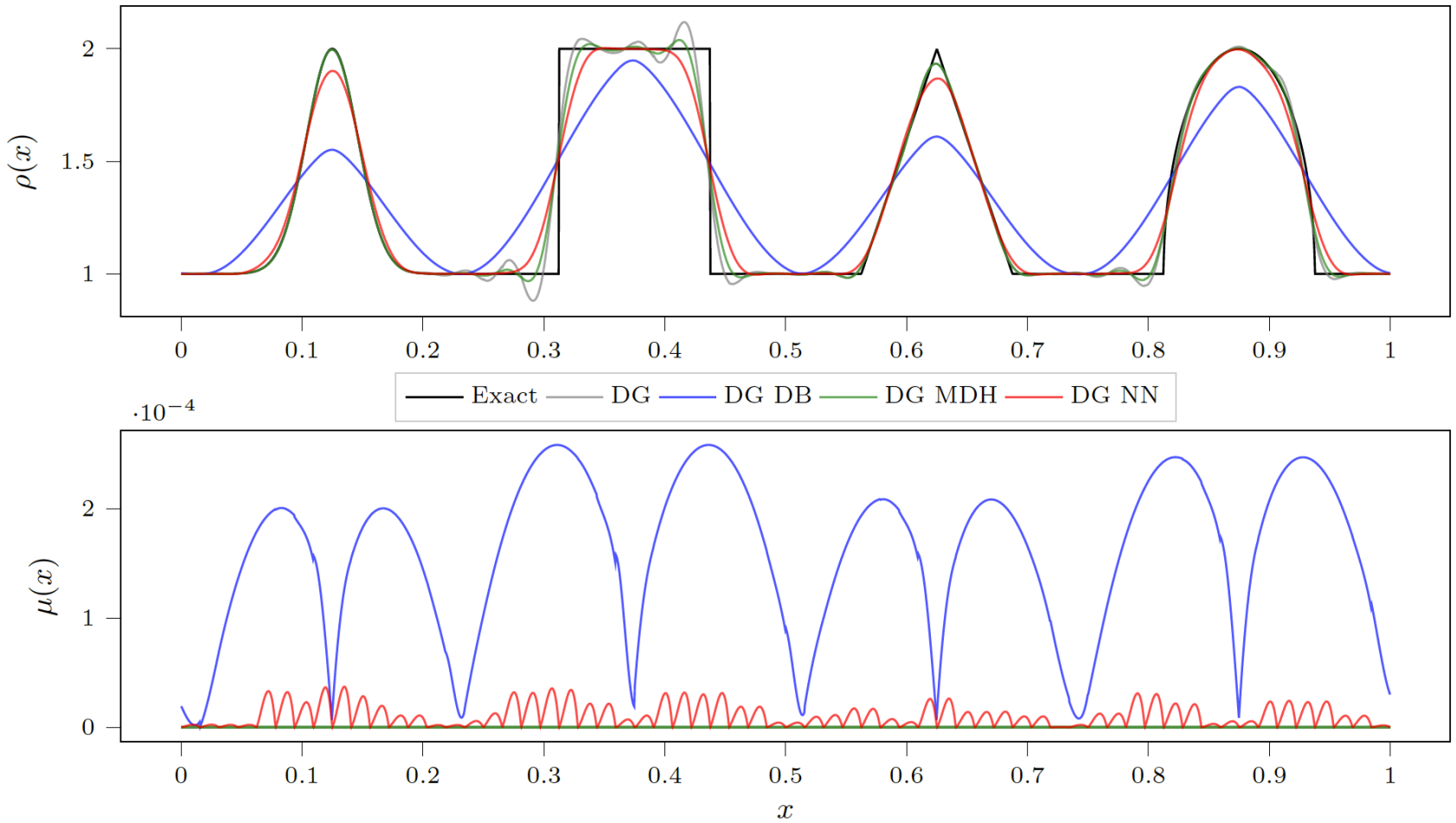
Results III

- We solve $\partial_t \rho + \partial_x \rho = 0$ with periodic boundary condition with $T_f = 2$ (long time).
- Comparison between differents viscosities:



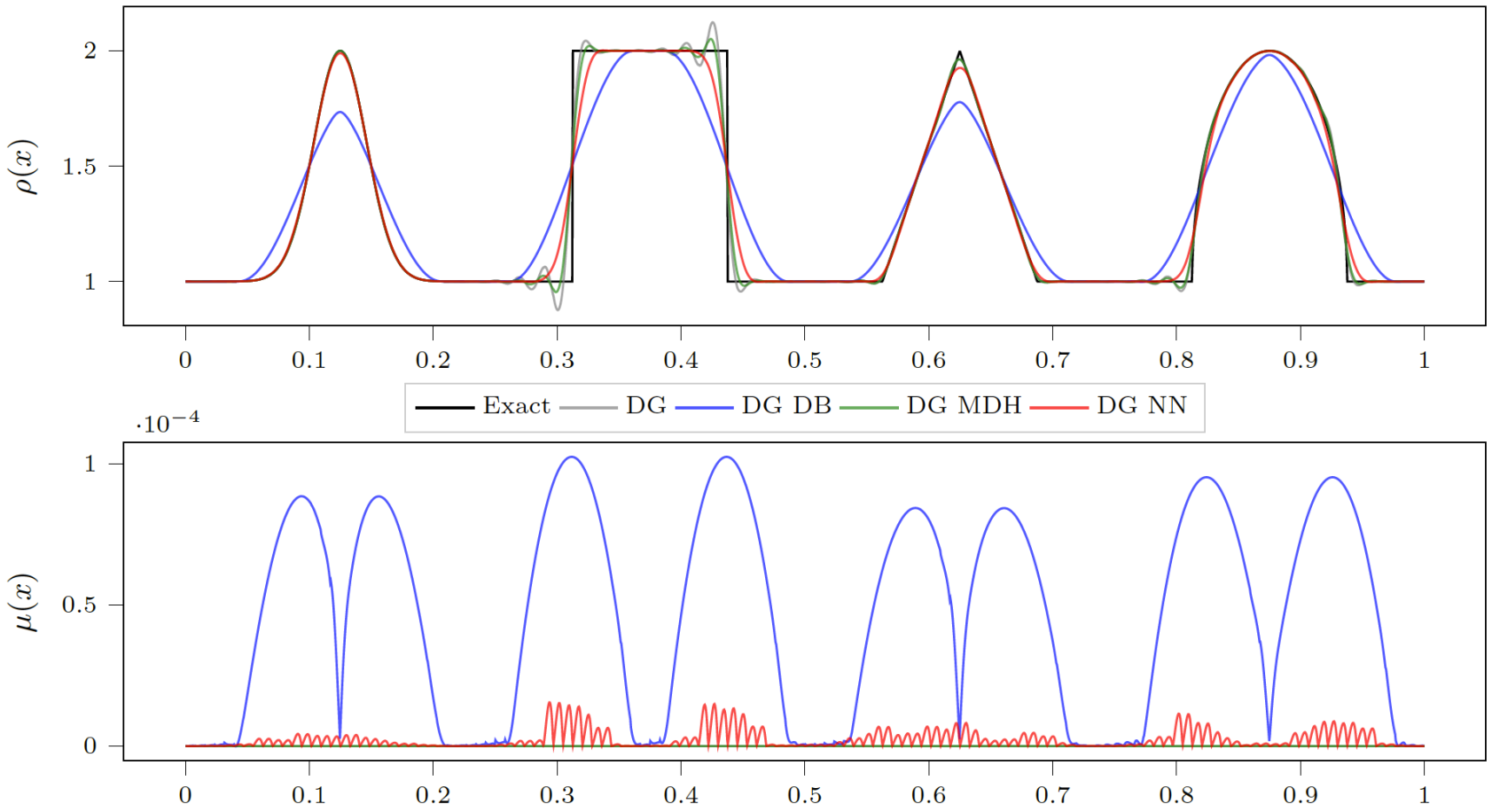
Results III

- We solve $\partial_t \rho + \partial_x \rho = 0$ with periodic boundary condition with $T_f = 2$ (long time).
- Comparison between differents viscosities:



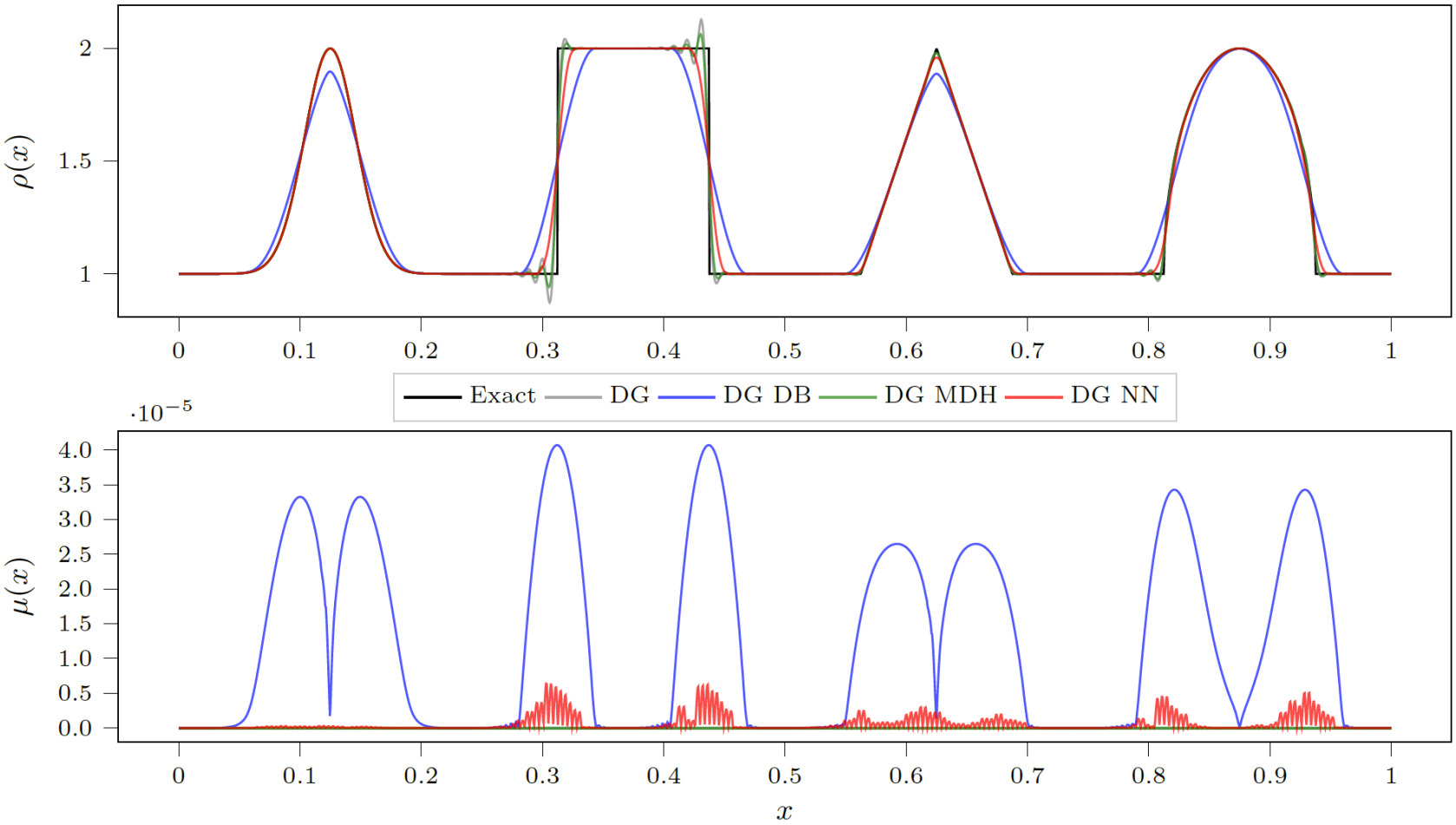
Results III

- We solve $\partial_t \rho + \partial_x \rho = 0$ with periodic boundary condition with $T_f = 2$ (long time).
- Comparison between differents viscosities:



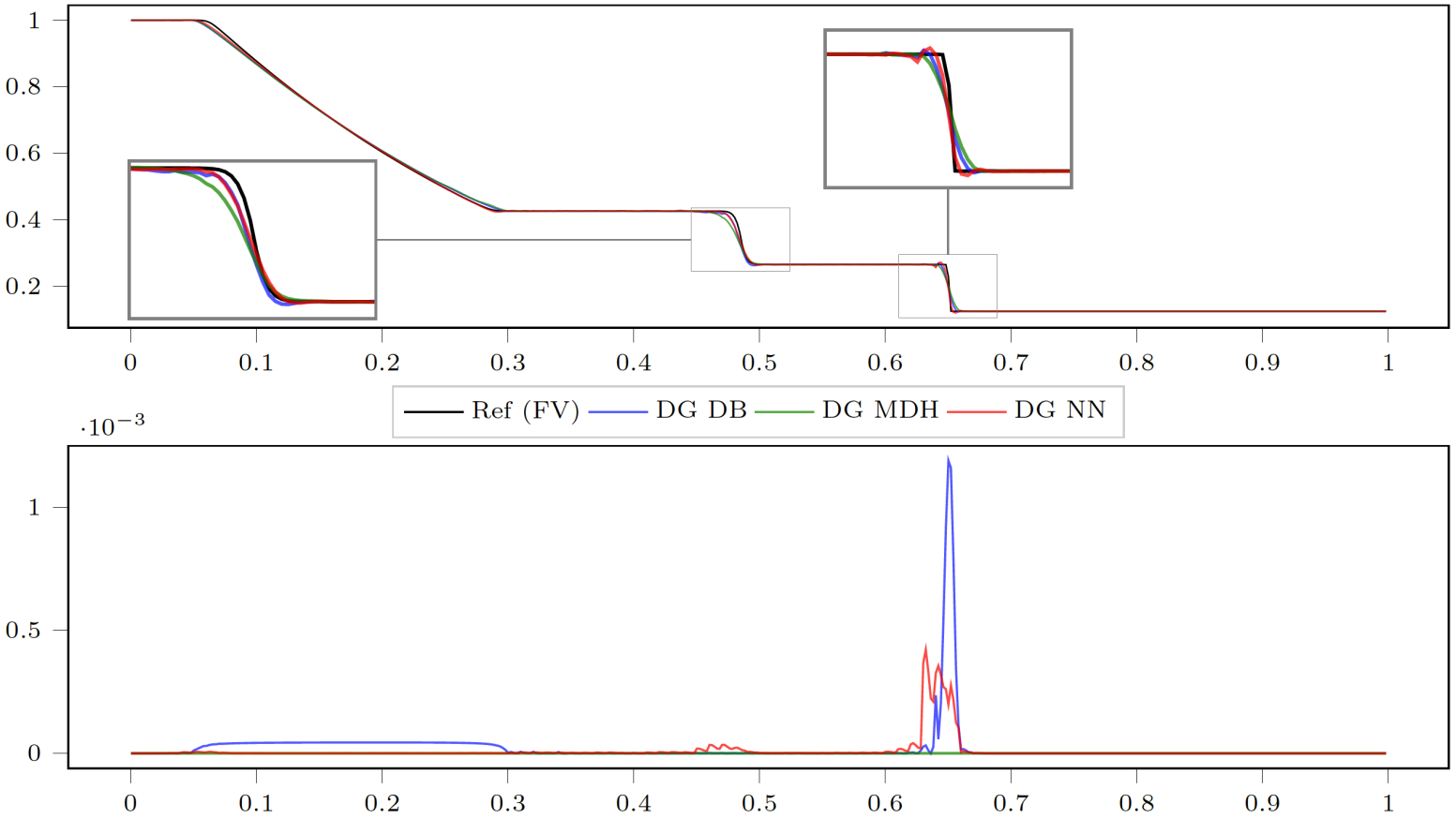
Results III

- We solve $\partial_t \rho + \partial_x \rho = 0$ with periodic boundary condition with $T_f = 2$ (long time).
- Comparison between differents viscosities:



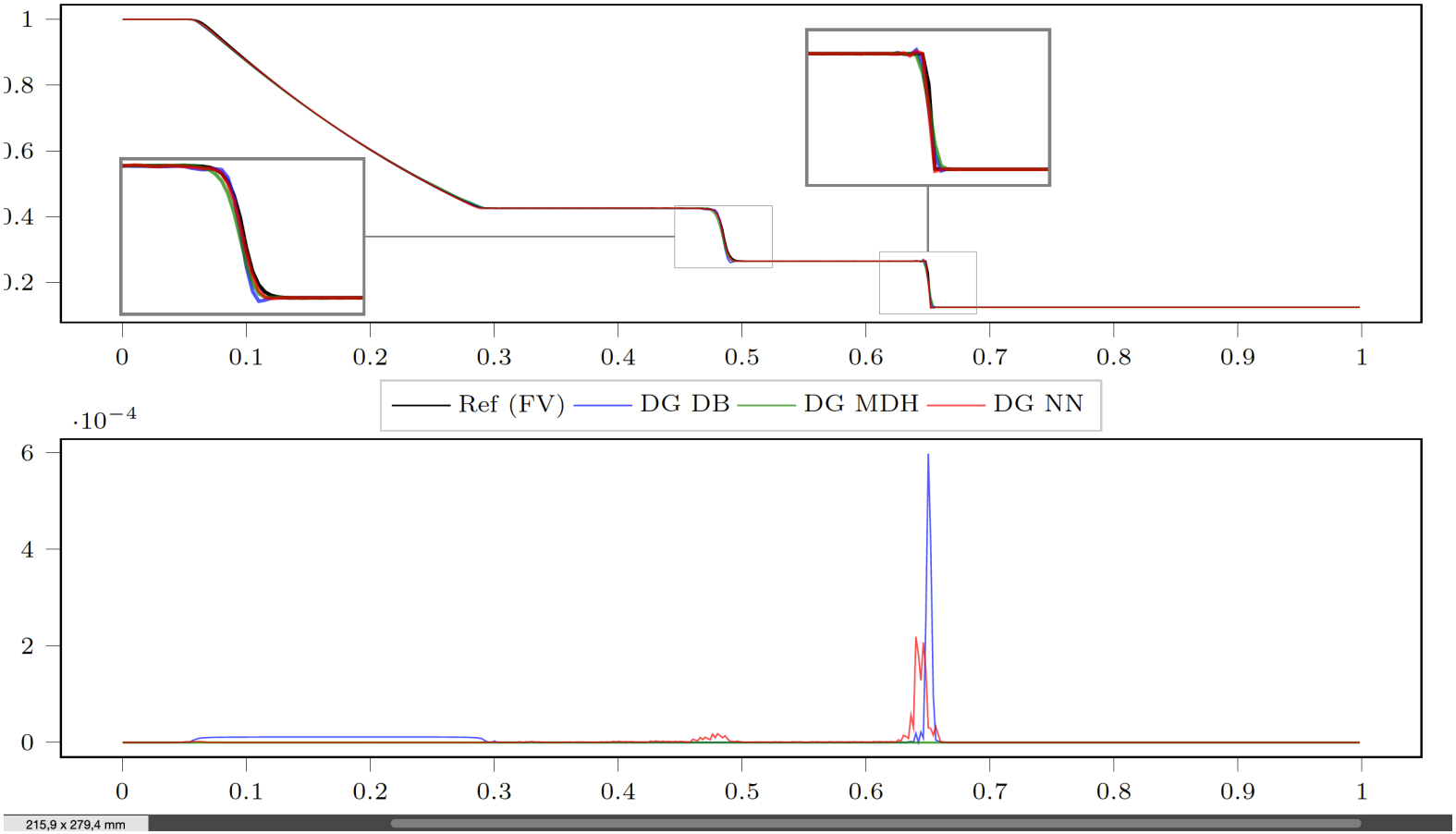
Results III

- We solve the Euler equation with Neumann BC.
- Comparison between differents viscosities:
- SOD test case 32 cells



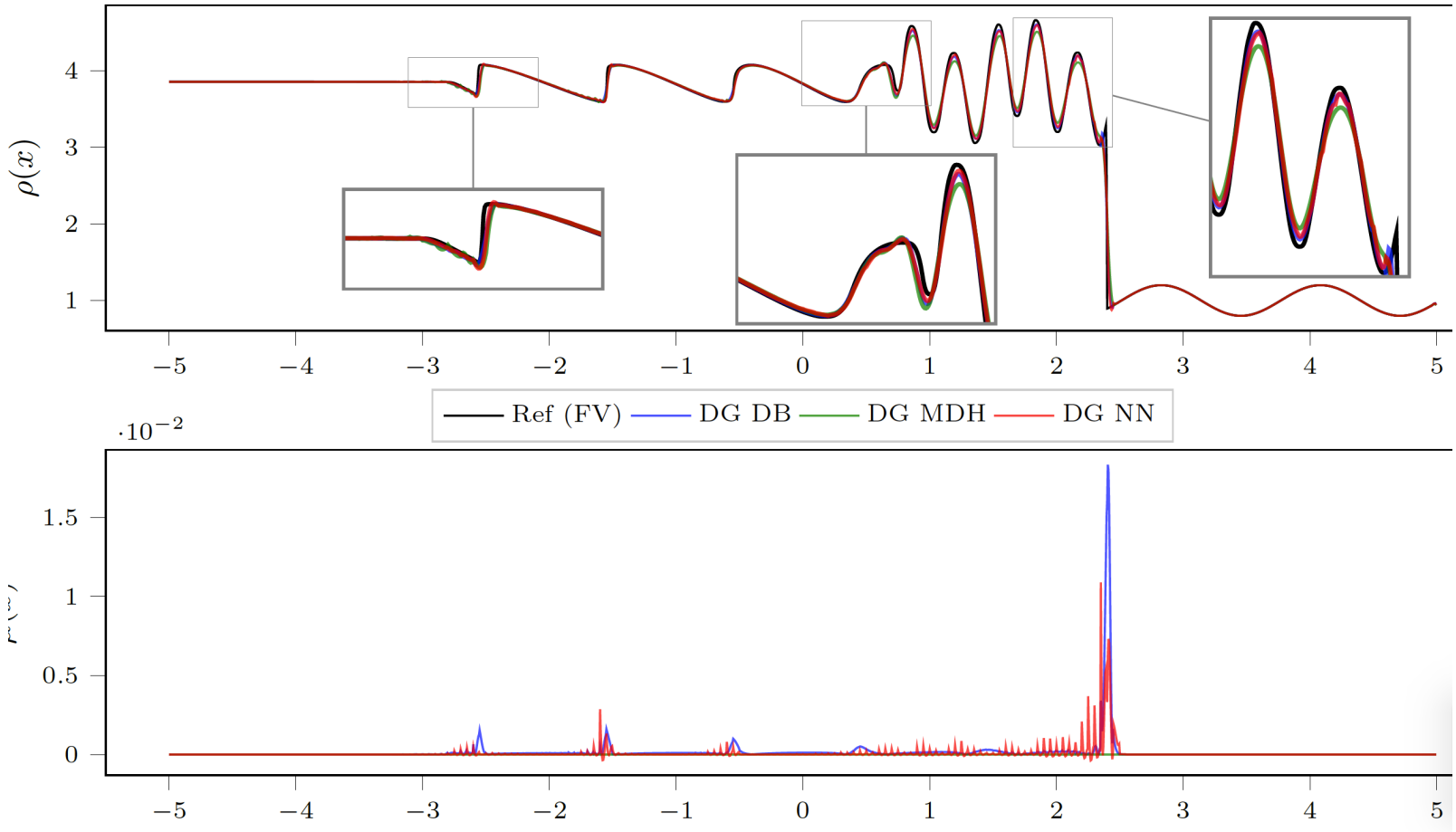
Results III

- We solve the Euler equation with Neumann BC.
- Comparison between differents viscosities:
- SOD test case 64 cells



Results III

- We solve the Euler equation with Neumann BC.
- Comparison between differents viscosities:
- Shu Osher test case



Conclusion

Conclusion

Modeling

- The neural network can approximate complicate nonlocal function like closure. Use to construct intermediaries/reduced models. 2D Extension for gas less stable.
- **Current work**: Indermediary models between Euler and Vlasov with stability proof.

PINNs/NO

- The PINNs or Neural operator allows obtaining good prior to a solution.
- We show two ways to plug it in numerical method to win CPU time or accuracy. Could be generalized to other numerical (DG, Semi-Lagrangian etc).
- How deal with more complex multi-scale PDE ?

Differentiable physics

- DP allows tuning a scheme or a part of your scheme. Here viscosity but it can be: closure, fluxes, splitting coefficients, etc.
- How take into account to the long time behavior ? Mix with reinforcement learning ?

INRIA team

The INRIA Team Tonus on plasma modeling would come from MACARON on ML and PDE. Possible open positions.

Bonus: Unstructured meshes

Graph neural networks

CNN and signal processus

Convolutional neural networks are very useful to analyze pictures and detect patterns (segmentation, . . .). For PDEs, they can be useful for discontinuity or front tracking.

- How to use them on unstructured meshes?
- CNN have been made for pictures, and for regular grids.

GNN

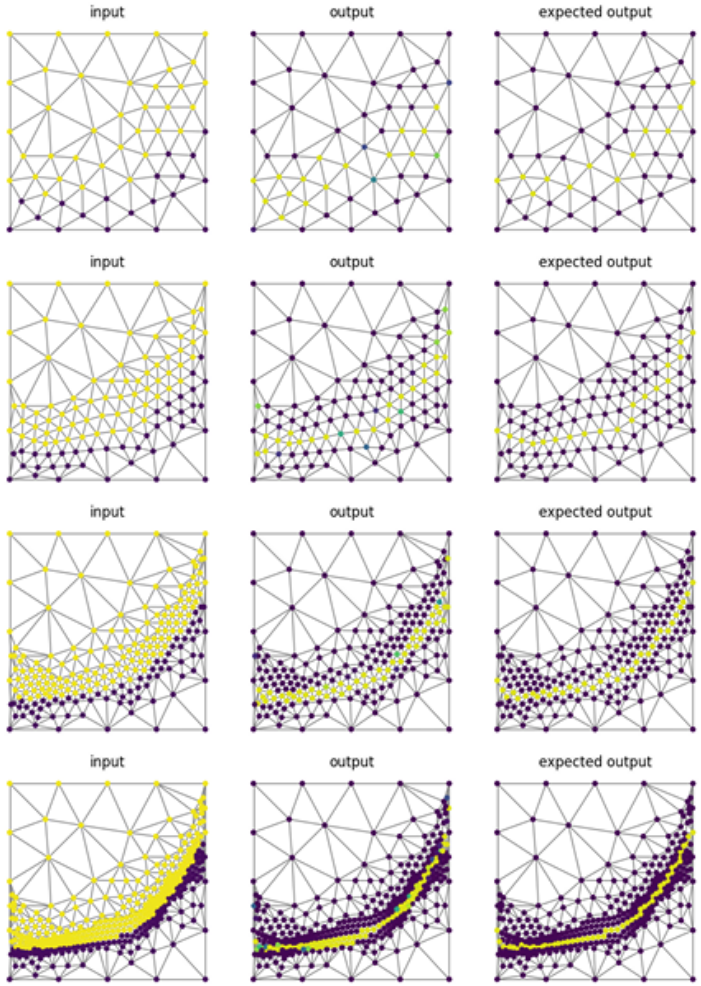
In the last five years, many Graph convolutional neural network have been proposed and can be used on general meshes.

Important

Choose the network type such that the performance is not impacted when changing the mesh but not the topology.

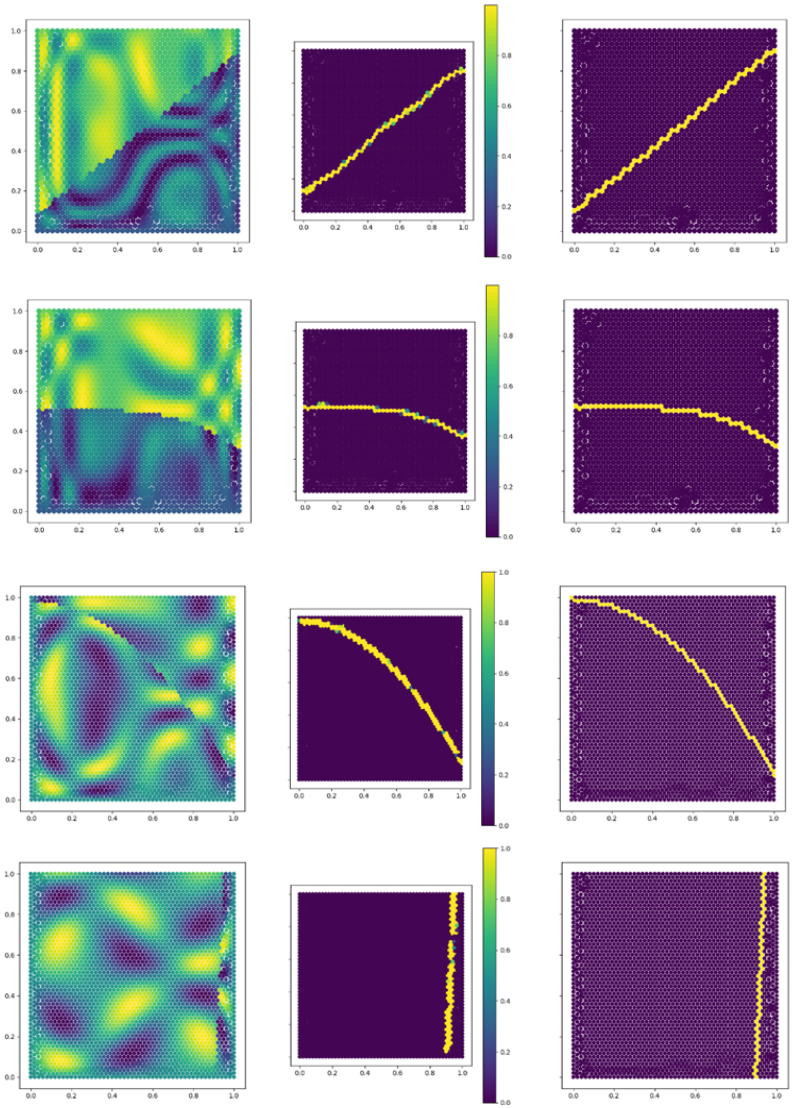
Discontinuity Tracking I

- **A first application**: discontinuity/shock detection and mesh refinement.
- **Case 1**: constant by part function. Localization of discontinuity by GCN and iterative refinement.



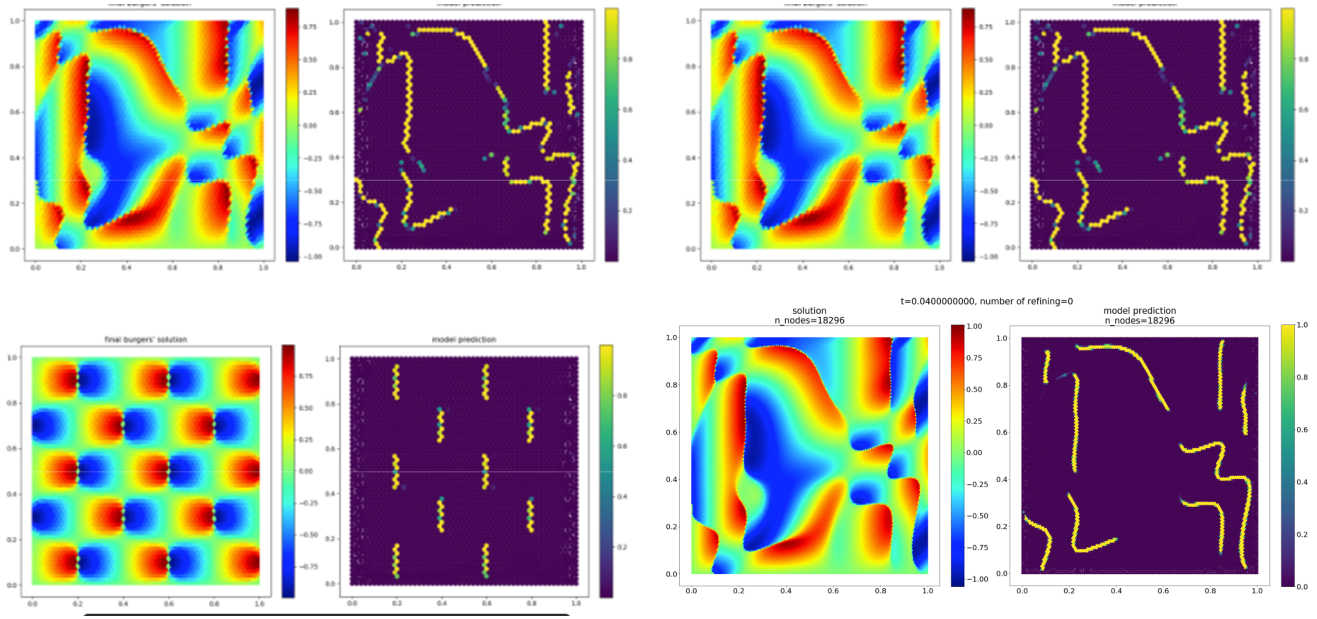
Discontinuity Tracking II

- **Case 2**: Discontinuity detection for non piecewise constant function. Unet architecture with Chebnet and geometric pooling layers.
- Training on a single mesh (for computational reasons)

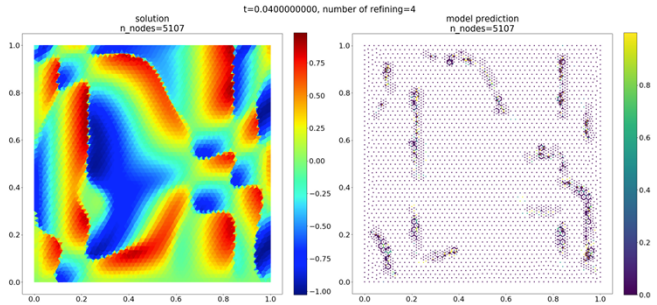


Discontinuity Tracking III

- **Case 3**: Discontinuity detection in Burgers simulation using previous training



- A first refinement approach



- After refining the mesh, the discontinuity remains detected. This effect dampens after four refinements, possibly due to training on a single mesh.