Enhanced DG schemes by neural networks

L. Bois[1], E. Franck[12], Victor Michel-Dansac[12],
L. Navoret[12], V. Vigon[12]

Scientific Computing seminar. University of Waterloo

[1]Inria Nancy Grand Est, France
[2]IRMA, Strasbourg university, France

Outline

Introduction

Machine learning and numerical method

- Common objectif between ML and numerical methods:
- We consider a unknown function

$$y = f(x), \quad x \in V \subset \mathbb{R}^d, \quad y \in W \subset \mathbb{R}^p$$

- **Objective**: Find $f_h \in H$ an approximation of f with H a functional space.
- **Difficulty**: we want construct a infinite dimensional space

Solution: parametric models

- We consider a known function $f_\theta(x)$ with θ the unknowns parameters
- the problem becomes:

$$\text{find } \theta, \text{ such that } \| f_\theta - f \|_H \leq \epsilon$$

- **ML approach**: we construct θ by constraining the approximation by the data
- We assume that we have data $\{(x_1, f_1), \dots, (x_N, f_N)\}$ such that:

$$f_i = f(x_i) + \epsilon_i, \quad \epsilon_i \sim \mathcal{N}(0, 1)$$

- The parameters θ are chosen such that f_θ well approximate f on the data which is equivalent to solve:

$$\text{argmin}_\theta \sum_{i=1}^{N} d(u_i, u_\theta(x_i))$$

- **Numerical method approaches**: we construct θ by constraining the approximation by a physical equation
- **Objective**: transform PDE constrains on the function into a constrain on the parameters

$$L(u(x)) = f(x) \Longrightarrow A(\theta) = b(\theta)$$

with L a operator and A, b a linear system.

- Equation form /parametric model change with the method.

Revolution of deep learning

- Classical parametric model in ML : $u_\theta(x) = \sum_{i=1}^{N} \langle \theta, \Phi(x) \rangle$ with Φ a vector of simpel functions (polynomial, kernel, affine function, etc).

Deep ML

One of the major change with neural network (NN): we use parametric models nonlinear compared to the parameters θ.

Consequences

- These models needs significantly less parameters for large dimensional problems.
- The convex optimization associated to linear model becomes nonlinear/ non convex
- We project the function on finite dimension manifold and not vectorial space. Few garanties compared to the linear case.
- Very efficient methods for automatic differentiation of large composition of functions have been designed.

Aim

Our aim is to use this news tools to provide more efficients numerical methods.

- We will show how design numerical method using NN (PINNs)
- We will propose a DG method using the good ability of NN in large dimension.
- We will propose a DG viscosity using the good property fo networks and autodiff tools.

Numerical methods and PINNs

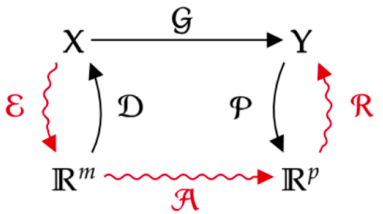
Mesh based methods

Polynomial Lagrange interpolation

We consider a domain $[a, b]$. There exists a polynomial P of degree k such that, for any $f \in C^0([a, b])$,

$$|f(x) - P(x)| \leq |b - a|^k \max_{x \in [a,b]} |f^{k+1}(x)|.$$

- On small domains ($|b - a| \ll 1$) or for large k, this polynomial gives a very good approximation of any continuous function.
- Very high degrees k can generate oscillations (like in ML).
- To obtain good approximation: we introduce a mesh and a cell-wise polynomial approximation



- Possible since contrary to ML, the domain of inputs is always well-known.

First step: choose a parametric function

We define a mesh by splitting the geometry in small sub-intervals $[x_k, x_{k+1}]$, and we propose the following candidate to approximate the PDE solution T

$$T_{|[x_k, x_{k+1}]}(t, x) = \sum_{j=1}^{k} \theta_k^j \phi_j(x).$$

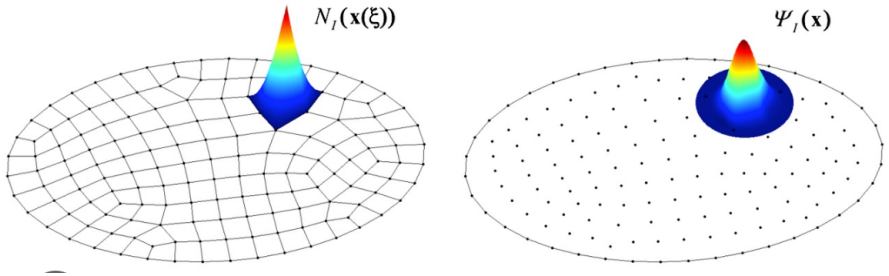
This is a piecewise polynomial representation.

Classical numerical methods

Parametric model for all numerical methods;

$$f_\theta = \sum_{i=1}^{n} \theta_i \phi_i(x)$$

- Classical mesh based methods:
  - **Finite element**: C^p continuity between the cells (depend of the finite element) so $\phi_i(x)$ piecewise polynomial.
  - **DG**: discontinuity between the cell so $\phi_i(x) = p_j(x)\chi_{x\in\Omega_i}$.
  - **DG Treffz**: same as DG but non-polynomial.
  - **Finite difference**: punctual value so $\phi_i(x) = \delta_{x_i}(x)$ with x_i a mesh node.
- Classical mesh free methods:
  - **Spectral**: we use Hilbert basis so $\phi_i(x) = sin(2\pi k_i x)$ for example (same with Hermite, Laguerre, Legendre polynomiales).
  - **Radial basis**: we use radial basis so for example $\phi_i = \phi(\mid x - x_i)$ with ϕ a Gaussian or $\frac{1}{1+\sigma^2 x^2}$.



How determinate the degree of freedom

General method

The aim is to transform the PDE on T into a equation on θ (DOF).

- We note $V_\theta = Span\{f_\theta, \text{such that }, \theta \in V \in \mathbb{R}^n\}$
- First approach: Galerkin
  - Rewrite the problem:

$$-\Delta T(x) = f(x), \Longleftrightarrow \min_{T \in H} \int_\Omega \left(\mid \nabla T(x) \mid^2 - f(x) T(x) \right) dx$$

  - Galerkin projection:

$$\min_{T_\theta \in V_\theta} \int_\Omega \left(\mid \nabla T_\theta(x) \mid^2 - f(x) T_\theta(x) \right) dx$$

- The problem is quadratic in θ. The parameters which put the gradient at zero satisfy

$$\int_\Omega (-\Delta T_\theta(x) - f)\phi_i(x) = 0, \quad \forall i \in \{1, ..., n\}$$

- Since we can compute exactly the derivative and numerically the integral we precompute everything (after in general a integration by part) to obtain

$$A\theta = b$$

- Second approach: Least square Galerkin projection

$$\min_{\theta \in V} \int_\Omega \mid -\Delta T - f \mid^2 dx$$

Garanties

Essential point

The space V_θ is a a vectorial space. So the projector is on subspace is unique (projection on convexe subspace of Hilbert theorem). It allows to assure that the problem on parameters admit also a unique solution.

Convergence

The previous property coupled the approximation theorem of polynomial or Hilbert basis allows to assure that

$$\| f_\theta - f \|_h \to 0, \text{when } , n \to \infty$$

Curse of dimensionality

For mesh based approaches

$$\| f_\theta - f \|_H \leq Ch^p$$

with h characteristic size of the cells and the number of cell $N = O(\frac{1}{h^d})$. For that we need p polynomial by cell and direction so $O(p^d)$ parameters by cell. There is also similar problem for mesh less methods.

Nonlinear models

- Nonlinear version of classical models: f is represented by the DoF α_i, μ_i, ω_i or Σ_i:

$$f(x;\alpha,\mu,\Sigma)=\sum_{i=1}\alpha_i e^{(x-\mu_i)\Sigma_i^{-1}(x-\mu_i)},\quad f(x;\alpha,\omega)=\sum_{i=1}\alpha_i sin(\omega_i x)$$

- Neural networks (NN).

Layer

A layer is a function $L_l(\mathbf{x}_l):\mathbb{R}^{d_l}\rightarrow\mathbb{R}^{d_{l+1}}$ given by

$$L_l(\mathbf{x}_l)=\sigma(A_l\mathbf{x}_l+\mathbf{b}_l),$$

$A_l\in\mathbb{R}^{d_{l+1},d_l}$, $\mathbf{b}\in\mathbb{R}^{d_{l+1}}$ and $\sigma()$ a nonlinear function applied component by component.

Neural network

A neural network is parametric function obtained by composition of layers:

$$f_\theta(\mathbf{x})=L_n\circ\ldots\circ L_1(\mathbf{x})$$

with θ the trainable parameters composed of all the matrices $A_{l,l+1}$ and biases $\mathbf{b}_l$.

- **Go to nonlinear models**: would allows to use less parameters and data.
- **Go to nonlinear models** allows to use NN which are: accurate global model, low frequency (better for generalization) and able to deal with large dimension.

Space-time approach: PINNs I

Neural methods

The PINNs and Neural Galerkin approaches use exactly the same strategy than classical numerical methods but project on manifold associated to nonlinear parametric models compared to the parameters

Idea of PINNs

- For u in some function space $\mathcal{H}$, we wish to solve the following PDE:

$$\partial_t u = \mathcal{F}(u, \nabla u, \Delta u) = F(u).$$

- Classical representation for space-time approach: $u(t,x) = \sum_{i=1}^{N} \theta_i \phi_i(x,t)$
- Deep representation: $u(t,x) = u_{nn}(x,t;\theta)$ with u_{nn} a NN with trainable parameters θ.

Which projection

- Galerkin projection is just valid for elliptic equations with energetic form.
- More general: Least square Galerkin. We minimize the least square residue of the restricted to the manifold associated by our chosen neural architecture.

Space-time approach: PINNs II

- We define the residual of the PDE:

$$R(t,x) = \partial_t u_{nn}(t,x;\theta) - \mathcal{F}(u_{nn}(t,x;\theta), \partial_x u_{nn}(t,x;\theta), \partial_{xx} u_{nn}(t,x;\theta))$$

- To learn the parameters θ in $u_{nn}(t,x;\theta)$, we minimize:

$$\theta = \arg\min_{\theta} \Big(J_r(\theta) + J_b(\theta) + J_i(\theta) \Big),$$

with

$$J_r(\theta) = \int_0^T \int_\Omega |R(t,x)|^2 dx dt$$

and

$$J_b(\theta) = \int_0^T \int_{\partial\Omega} \|u_{nn}(t,x;\theta) - g(x)\|_2^2 dx dt, \quad J_i(\theta) = \int_\Omega \|u_{nn}(0,x;\theta) - u_0(x)\|_2^2 dx.$$

- If these residuals are all equal to zero, then $u_{nn}(t,x;\theta)$ is a solution of the PDE.
- To complete the determination of the method, we need a way to compute the integrals. In practice we use Monte Carlo.
- **Important point**: the derivatives are computed exactly using automatic differentiation tools and back propagation. Valid for any decoder proposed.

Space-time approach: PINNs II

- We define the residual of the PDE:

$$R(t,x) = \partial_t u_{nn}(t,x;\theta) - \mathcal{F}(u_{nn}(t,x;\theta), \partial_x u_{nn}(t,x;\theta), \partial_{xx} u_{nn}(t,x;\theta))$$

- To learn the parameters θ in $u_{nn}(t,x;\theta)$, we minimize:

$$\theta = \arg\min_{\theta} \Big(J_r(\theta) + J_b(\theta) + J_i(\theta) \Big),$$

with

$$J_r(\theta) = \sum_{n=1}^{N} \sum_{i=1}^{N} |R(t_n, x_i)|^2$$

with (t_n, x_i) sampled uniformly or through importance sampling, and

$$J_b(\theta) = \sum_{n=1}^{N_b} \sum_{i=1}^{N_b} |u_{nn}(t_n, x_i; \boldsymbol{\theta}) - g(x_i)|^2, \quad J_i(\theta) = \sum_{i=1}^{N_i} |u_{nn}(0, x_i; \boldsymbol{\theta}) - u_0(x_i)|^2.$$

- If these residuals are all equal to zero, then $u_{nn}(t,x;\theta)$ is a solution of the PDE.
- To complete the determination of the method, we need a way to compute the integrals. In practice we use Monte Carlo.
- **Important point**: the derivatives are computed exactly using automatic differentiation tools and back propagation. Valid for any decoder proposed.

PINNs for parametric PDEs

- **Advantages of PINNs**: mesh-less approach, not too sensitive to the dimension.
- **Drawbacks of PINNs**: they are often not competitive with classical methods.
- Interesting possibility: use the strengths of PINNs to solve PDEs parameterized by some $\boldsymbol{\mu}$.
- The neural network becomes $u_{nn}(t, x, \boldsymbol{\mu}; \theta)$.

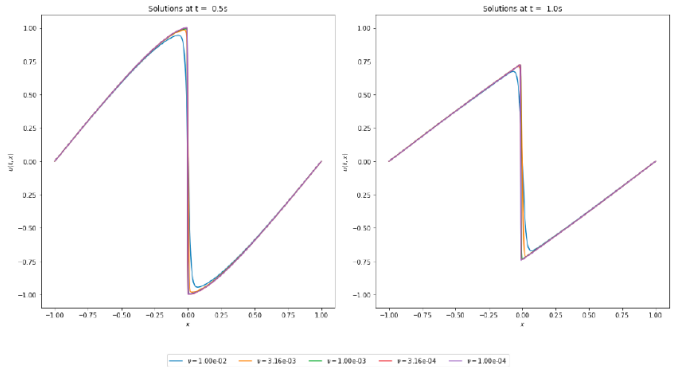
New Optimization problem for PINNs

$$\min_{\theta} J_r(\theta) + \ldots, , \quad \text{with}$$

$$J_r(\theta) = \int_{V_\mu} \int_0^T \int_\Omega \left\| \partial_t u_{nn} - \mathcal{L}\left(u_{nn}(t, x, \boldsymbol{\mu}), \partial_x u_{nn}(t, x, \boldsymbol{\mu}), \partial_{xx} u_{nn}(t, x, \boldsymbol{\mu})\right) \right\|_2^2 dxdt$$

with V_μ a subspace of the parameters $\boldsymbol{\mu}$.

- Application to the Burgers equations with many viscosities $[10^{-2}, 10^{-4}]$:



- Training for $\mu = 10^{-4}$: 2h. Training for the full viscosity subset: 2h.

First enhanced DG scheme

Nonlinar conservation laws and WB schemes

- We consider the following type of models (like everybody here):

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

- We are interested by the simulation of flows such as:

$$\partial_x \mathbf{F}(\mathbf{U}) = \mathbb{S}(\mathbf{U}) + \varepsilon \mathbf{P}(t, x)$$

Numerical difficulties

We consider a scheme of order q. For a equilibrium $\partial_x \mathbf{F}(\mathbf{U}) = \mathbf{S}(\mathbf{U})$ we have

$$\partial_x \mathbf{F}(\mathbf{U}_h) = \mathbb{S}(\mathbf{U}_h) + C\Delta x^q \mathbf{Q}_h(t, x).$$

if $\varepsilon < C\Delta x^q$ our scheme will not correctly capture perturbed flows.

WB and A-WB schemes

For a equilibrium $\partial_x \mathbf{F}(\mathbf{U}) = \mathbf{S}(\mathbf{U})$, a Well-Balanced scheme is such that $\partial_x \mathbf{F}(\mathbf{U}_h) = \mathbf{S}(\mathbf{U}_h)$, and an Approximately Well-Balanced scheme is such that

$$\partial_x \mathbf{F}(\mathbf{U}_h) = \mathbf{S}(\mathbf{U}_h) + C_2\Delta x^{q_2} \mathbf{Q}_h(t, x)$$

with $q_2 > q$ or $C_2 \ll C$.

- WB et A-WB make it possible to capture these perturbed flows.

DG schemes

- We recall quickly the Discontinuous Galerkin method.

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

$$\int_{\Omega_j} \partial_t \mathbf{U} \phi dx + \int_{\Omega_j} \partial_x \mathbf{F}(\mathbf{U}) \phi dx = \int_{\Omega_j} \mathbf{S}(\mathbf{U}) \phi dx$$

- In each cell we consider a discrete vectorial polynomial space: $V_h = Span(\phi_1(x), ...\phi_q(x))$ and we use

$$\mathbf{U}_{|\Omega_j}(t,x) = \sum_{i=1}^{q} \alpha_i(t)\phi_i(x) \in V_h$$

and

$$\phi = \phi_1, ...\phi = \phi_q$$

- We obtain a matrix-vector system of size $q \times q$

$$\boxed{\mathcal{M}\partial_t \boldsymbol{\alpha}(t) + \mathcal{K}(\boldsymbol{\alpha}(t)) = \mathcal{S}(\boldsymbol{\alpha}(t))}$$

with $\mathcal{S}, \mathcal{K} \in \mathbb{R}^q$ and $\mathcal{M} \in \mathbb{R}^{q \times q}$

Main idea

Idea

- We consider a family of equilibria $\mathbf{U}_{eq}(x;\boldsymbol{\mu})$ indexed by some parameters $\boldsymbol{\mu}$.
- We assuming that we are able to produce an approximation of this equilibrium family, called the prior: $\mathbf{U}_\theta(x;\boldsymbol{\mu})$
- We propose to Introduce the prior on the equilibrium into the DG basis, to obtain more efficient approximation.

Basis with multiplicative prior

$$V_h^1 = \text{Span}\left(\mathbf{U}_\theta(x;\boldsymbol{\mu}), \mathbf{U}_\theta(x;\boldsymbol{\mu})(x-x_j), ..., \mathbf{U}_\theta(x;\boldsymbol{\mu})\frac{(x-x_j)^k}{k!}\right)$$

Basis with additive prior

- Solution 1:

$$V_h^2 = \text{Span}\left(\mathbf{U}_\theta(x;\boldsymbol{\mu}), 1, ..., \frac{(x-x_j)^{k-1}}{(k-1)!}\right)$$

- Does DG converge with non-polynomial bases?

Convergence within the Yuan-Shu framework I

anShu06 L. Yuan and C.-W. Shu: *Discontinuous Galerkin method based on non-polynomial approximation spaces*, JCP 2006.

Main result I of [YuanShu06]

We consider a basis $(v_1, ..v_K)$ of the space V_h. If there are constant a_{ik} and b_i **independant of the size of the cell** Δx_j, and if we have

$$|v_i(x) - \sum_{k=1}^{K} a_{ik}(x - x_i)^k| \leq b_i(\Delta x_j)^{K+1} \quad (1)$$

then for any function $u \in H^{K+1}(\Omega_j)$, there exists $v_h \in V_h$ and

$$|v_h - u_h| \leq C|u|_{H^{K+1}(\Omega_j)}(\Delta x_j)^{K+\frac{1}{2}}$$

Main result II of [YuanShu06]

With the first result, we can prove the convergence (with additional steps) of the DG scheme using the V_h basis.

Convergence with the Yuan-Shu framework II

Result in the scalar case

We assume that $\mathbf{u}_\theta(x;\boldsymbol{\mu}) \in C^p(\Omega)$ with $p \geq K+1$. Then, the previously proposed bases satisfy the assumption of [YuanShu06], and the DG scheme converges.

- Example of proof for V_h^1.
- Since the neural network is $C^{K+1}(\mathbb{R})$, we can write a Taylor series, to obtain:

$$u_\theta(x) = u(x_j) + (x - x_j)u_\theta^{'} + ... + \frac{u^{(K)}(x_j)}{K!}(x - x_j)^K + \frac{u^{(K+1)}(c)}{(K+1)!}$$

with $c \in [x_j, x]$. We then get:

$$\begin{pmatrix} u_\theta(x) \\ u_\theta(x)(x-x_j) \\ ... \\ u_\theta(x)(x-x_j)^K \end{pmatrix} = \underbrace{\begin{pmatrix} u_\theta(x_j) & u_\theta^{'}(x_j) & ... & \frac{u_\theta^{(K)}(x_j)}{K!} \\ 0 & u_\theta(x_j) & ... & \frac{u_\theta^{(K-1)}(x_j)}{(K-1)!} \\ ... & & & \\ 0 & 0 & ... & u_\theta(x_j) \end{pmatrix}}_{A} \begin{pmatrix} 1 \\ (x-x_j) \\ ... \\ (x-x_j)^K \end{pmatrix} + \underbrace{\begin{pmatrix} \frac{u_\theta^{K+1}(c)}{(K+1)!} \\ \frac{u_\theta^{K}(c)}{(K)!} \\ ... \\ 1 \end{pmatrix}}_{b}$$

- It easy to see that the matrix A, its inverse A^{-1}, and the vector b are independent from Δx_j. Therefore, the assumption is verified.

Specific estimate

- **Problem**: the previous approach does not give the expected gain associated with these bases.

First lemma

We consider a basis $(v_1, ..v_K)$ of the space V_h^1 and assume that $u_\theta(x;\boldsymbol{\mu}) \in C^p(\Omega)$ $u_\theta(x;\boldsymbol{\mu})^2 > \alpha_0, \quad \forall x \in \Omega$. For any function $u \in H^{K+1}(\Omega_j)$, the L^2 projector on V_h^1, $P_h(u) \in V_h^1$, satisfies

$$|u - P_h(u)| \leq C \left| \frac{u(x)}{u_\theta(x,\boldsymbol{\mu})} \right|_{H^{K+1}(\Omega_j)} (\Delta x_j)^{K+\frac{1}{2}} |u_\theta(x,\boldsymbol{\mu})|$$

Second lemma

Under the same assumptions than the previous lemma, for the basis V_h^1, we obtain that for any function $u \in H^{K+1}(\Omega)$

$$|u - P_h(u)|_{L^2(\Omega)} \leq C \left| \frac{u(x)}{u_\theta(x,\boldsymbol{\mu})} \right|_{H^{K+\frac{1}{2}}(\Omega)} (\Delta x)^{K+1} \parallel u_\theta(x,\boldsymbol{\mu}) \parallel_\infty$$

- For the additive case, we expect an error in $\left| u(x) - u_\theta(x,\boldsymbol{\mu}) \right|_{H^{K+\frac{1}{2}}(\Omega)}$

Linear advection equation

In all the numerical experiments, we use the V_h^3 basis. Results are similar with the other bases.

We first consider the first-order advection equation

$$\begin{cases} \partial_t u + \partial_x u = s(u;\mu), \\ u(t=0,x) = u_0(x), \end{cases}$$

with the following parameterized source term and initial condition

- $s(u;\alpha,\beta) = \alpha u + \beta u^2$;
- $u_0(x) = \varepsilon + u_{\text{eq}}(x;\alpha,\beta,\upsilon)$, with the steady solution u_{eq} depending on α, β and an additional parameter υ.

Hence, we have three parameters: $0.5 \leq \alpha \leq 1$, $0.5 \leq \beta \leq 1$, $0.1 \leq \upsilon \leq 0.2$

We propose three experiments: approximate

- a steady solution,
- a perturbed steady solution,
- an unsteady solution.

Training the PINN takes about 10 minutes on an old GPU, with **no data**, only the PINN loss.

Linear advection equation: steady solution

In this case, $\varepsilon = 0$ in the initial condition, so we approximate the steady solution itself.

We compute the error between the exact and approximate solutions, for polynomial bases with $n_G \in \{1, 2, 3, 4\}$ elements, and with or without PINN prior.

We take a quadrature of degree $n_Q = \max(3, n_G + 1)$.

pts	error_ϕ	order	$\text{error}_{\bar{\phi}}$	order	gain
10	7.42e-02	—	3.89e-04	—	190.66
20	2.64e-02	1.49	1.45e-04	1.42	181.76
40	9.29e-03	1.51	5.23e-05	1.47	177.55
80	3.27e-03	1.50	1.89e-05	1.46	172.63
160	1.18e-03	1.47	6.95e-06	1.45	170.09

(a) errors with a one-element basis, $n_G = 1$

Linear advection equation: steady solution

In this case, $\varepsilon = 0$ in the initial condition, so we approximate the steady solution itself.

We compute the error between the exact and approximate solutions, for polynomial bases with $n_G \in \{1, 2, 3, 4\}$ elements, and with or without PINN prior.

We take a quadrature of degree $n_Q = \max(3, n_G + 1)$.

pts	error$_\phi$	order	error$_{\bar{\phi}}$	order	gain
10	1.80e-03	—	1.09e-05	—	164.69
20	3.20e-04	2.50	1.93e-06	2.51	165.75
40	5.51e-05	2.54	3.33e-07	2.53	165.27
80	9.41e-06	2.55	5.64e-08	2.56	166.77
160	1.80e-06	2.38	1.08e-08	2.38	166.83

(b) errors with a two-element basis, $n_G = 2$

Linear advection equation: steady solution

In this case, $\varepsilon = 0$ in the initial condition, so we approximate the steady solution itself.

We compute the error between the exact and approximate solutions, for polynomial bases with $n_G \in \{1, 2, 3, 4\}$ elements, and with or without PINN prior.

We take a quadrature of degree $n_Q = \max(3, n_G + 1)$.

pts	error_ϕ	order	$\text{error}_{\bar{\phi}}$	order	gain
10	2.23e-05	—	9.34e-07	—	23.94
20	2.02e-06	3.46	8.80e-08	3.41	23.01
40	1.75e-07	3.53	7.41e-09	3.57	23.60
80	1.45e-08	3.59	6.29e-10	3.56	23.14
160	1.46e-09	3.32	6.35e-11	3.31	22.99

(c) errors with a three-element basis, $n_G = 3$

Linear advection equation: steady solution

In this case, $\varepsilon = 0$ in the initial condition, so we approximate the steady solution itself.

We compute the error between the exact and approximate solutions, for polynomial bases with $n_G \in \{1, 2, 3, 4\}$ elements, and with or without PINN prior.

We take a quadrature of degree $n_Q = \max(3, n_G + 1)$.

pts	error_ϕ	order	$\text{error}_{\bar{\phi}}$	order	gain
10	2.81e-07	—	6.49e-08	—	4.33
20	1.26e-08	4.48	3.02e-09	4.42	4.17
40	5.72e-10	4.46	1.32e-10	4.52	4.34
80	2.31e-11	4.63	5.40e-12	4.61	4.29
160	1.21e-12	4.25	2.77e-13	4.29	4.40

(d) errors with a four-element basis, $n_G = 4$

Linear advection equation: steady solution

In this case, $\varepsilon = 0$ in the initial condition, so we approximate the steady solution itself.

We compute the error between the exact and approximate solutions, for polynomial bases with $n_G \in \{1, 2, 3, 4\}$ elements, and with or without PINN prior.

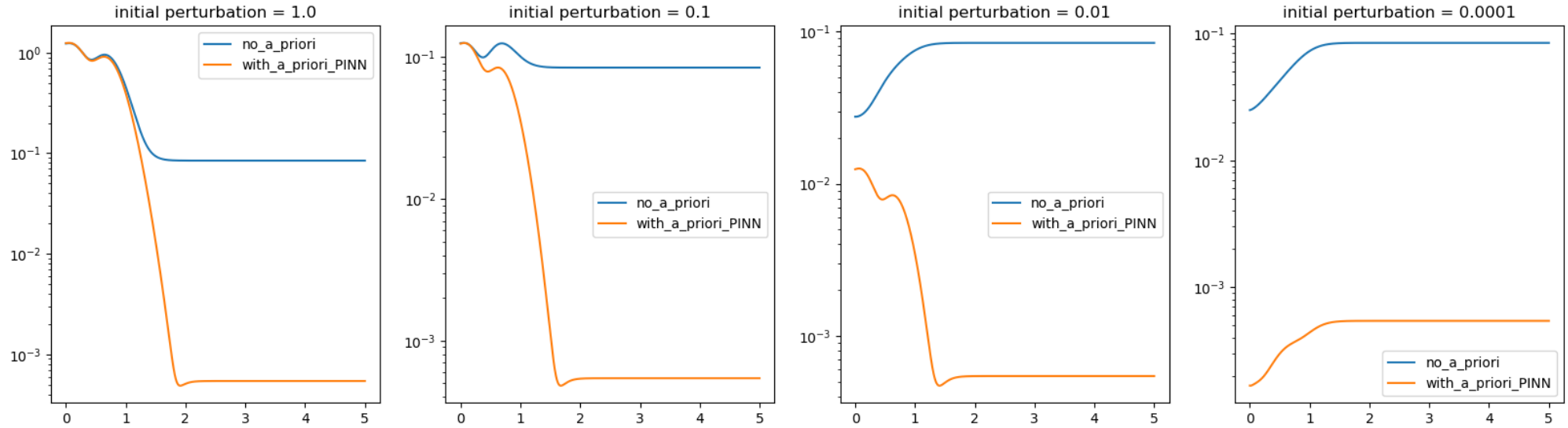
We take a quadrature of degree $n_Q = \max(3, n_G + 1)$.

The theoretical results show that the gain depend of $u - u_\theta$ in semi norm H^m. Since we train the prior with PINNs the second derivative is well reconstructed but less the high-order derivative. It explains the results.

Linear advection equation: perturbed steady solution

We now study the effect of nonzero values of ε in the initial condition: we take $\varepsilon \in \{10^{-4}, 10^{-2}, 10^{-1}, 1\}$ and 20 discretization cells.

We represent the error, over time, between the approximate and exact solutions.

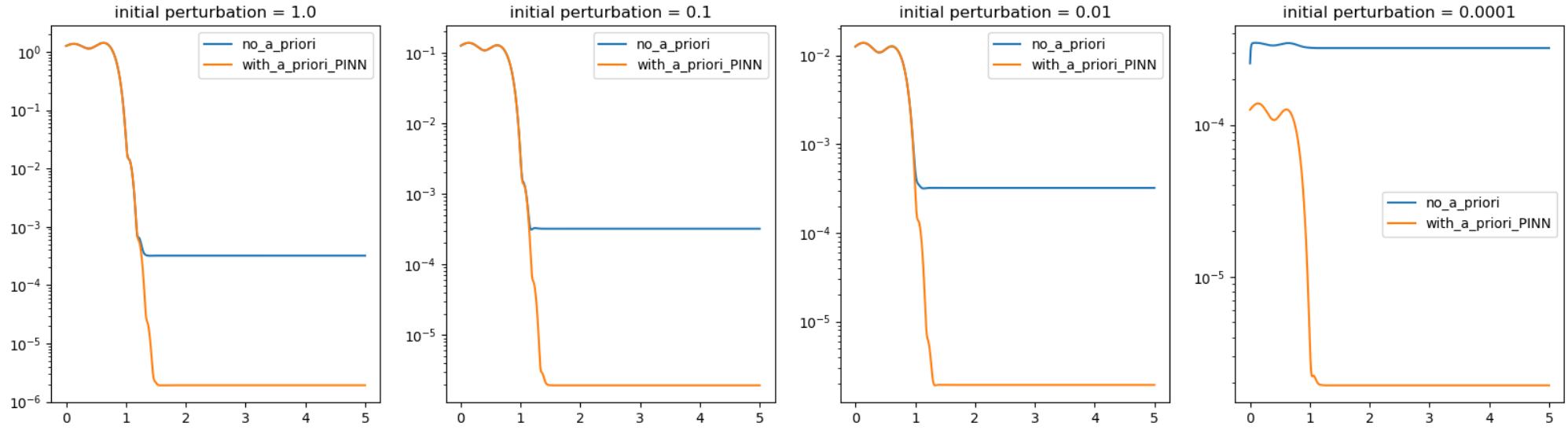


(a) errors with a one-element basis, $n_G = 1$

Linear advection equation: perturbed steady solution

We now study the effect of nonzero values of ε in the initial condition: we take $\varepsilon \in \{10^{-4}, 10^{-2}, 10^{-1}, 1\}$ and 20 discretization cells.

We represent the error, over time, between the approximate and exact solutions.

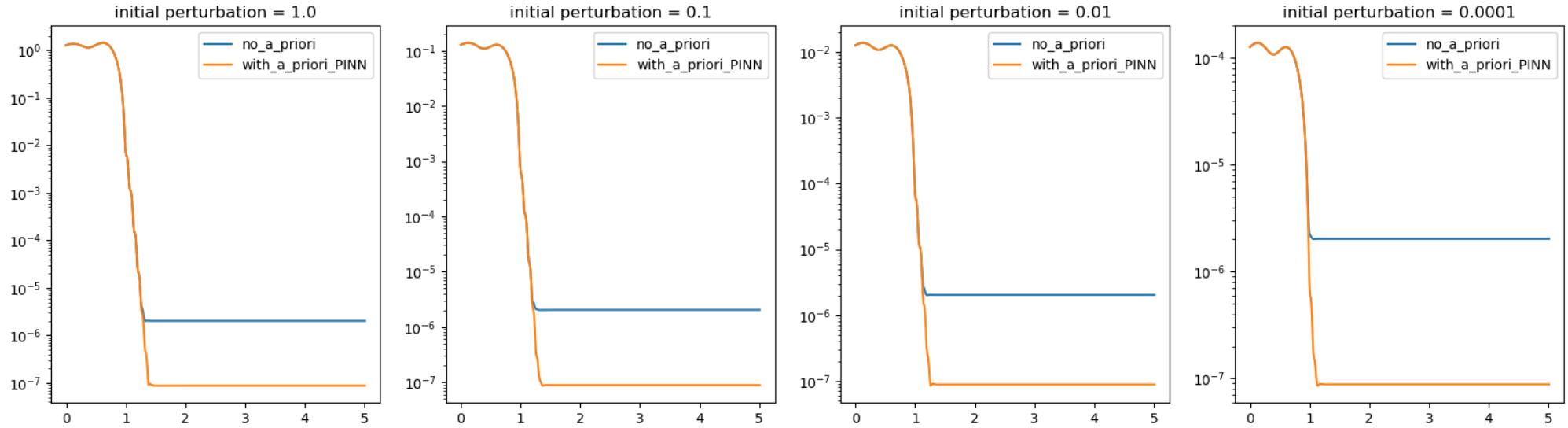


(b) errors with a two-element basis, $n_G = 2$

Linear advection equation: perturbed steady solution

We now study the effect of nonzero values of ε in the initial condition: we take $\varepsilon \in \{10^{-4}, 10^{-2}, 10^{-1}, 1\}$ and 20 discretization cells.

We represent the error, over time, between the approximate and exact solutions.

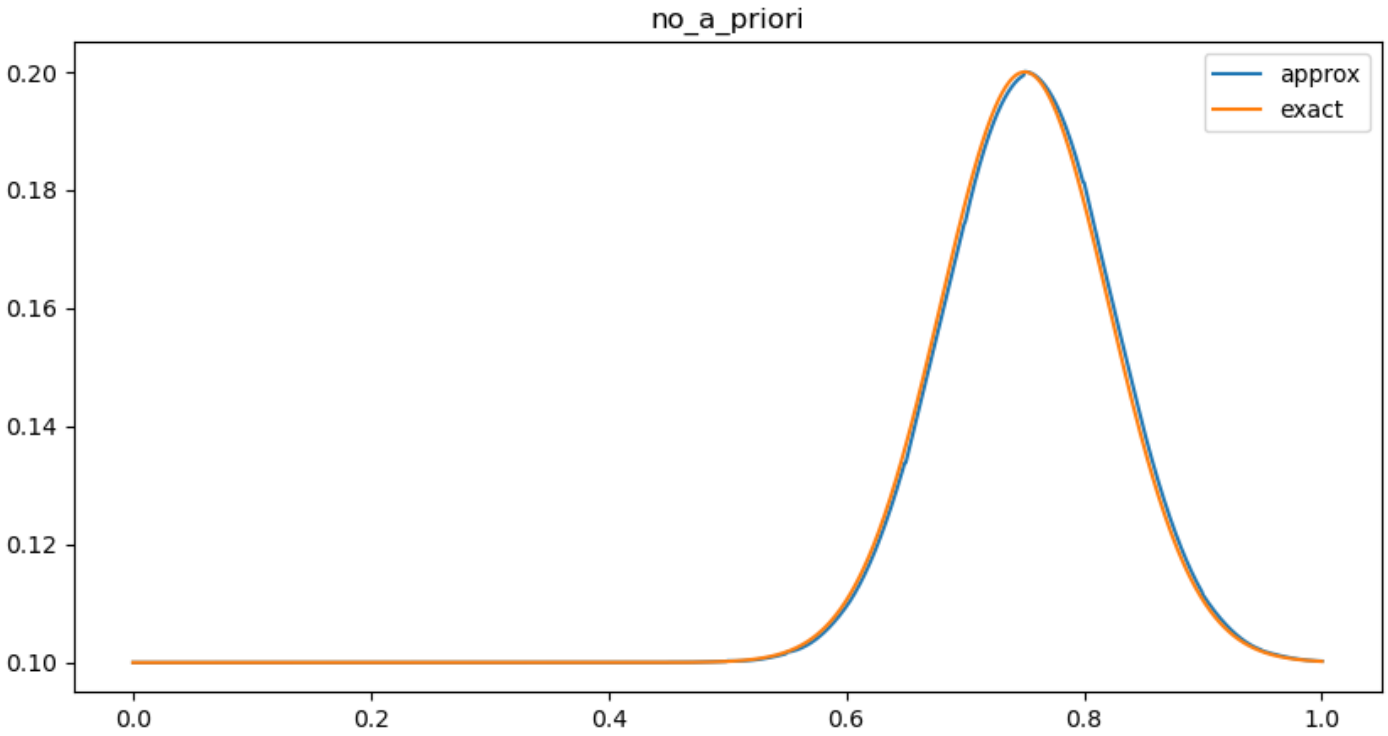


(c) errors with a three-element basis, $n_G = 3$

Linear advection equation: unsteady solution

Lastly, we perform the approximation of an unsteady solution with the two bases (with and without prior), to show that using the enhanced basis does not decrease approximation performance on unsteady solutions. The source term is zero in this case.

In this case, we take $n_G = 3$ and 20 discretization cells.

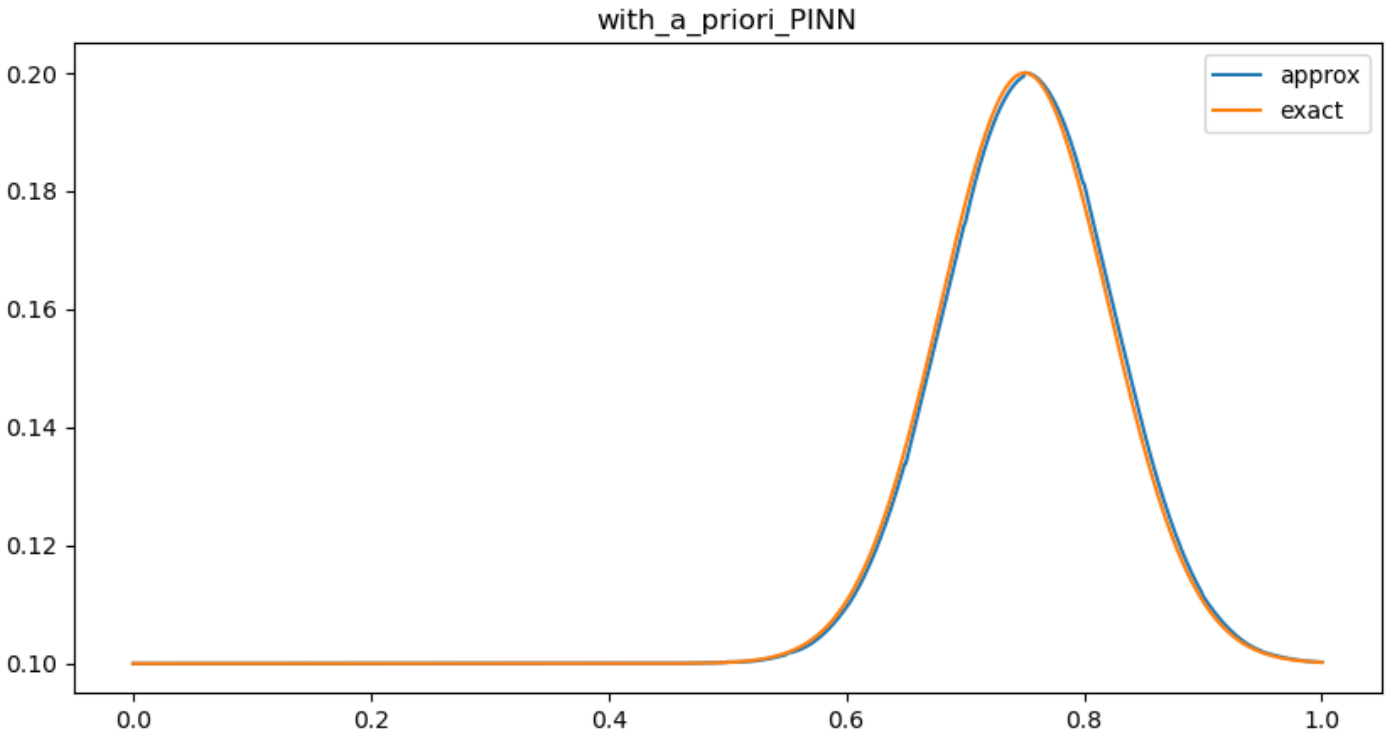


(a) without prior; error is 8.874×10^{-3}

Linear advection equation: unsteady solution

Lastly, we perform the approximation of an unsteady solution with the two bases (with and without prior), to show that using the enhanced basis does not decrease approximation performance on unsteady solutions. The source term is zero in this case.

In this case, we take $n_G = 3$ and 20 discretization cells.



(b) with prior; error is 8.874×10^{-3}, the same as without prior

Euler-Poisson system in spherical geometry

We consider the Euler-Poisson system in spherical geometry

$$\begin{cases} \partial_t \rho + \partial_r q = -\dfrac{2}{r} q, \\ \partial_t q + \partial_r \left(\dfrac{q^2}{\rho} + p \right) = -\dfrac{2}{r} \dfrac{q^2}{\rho} - \rho \partial_r \phi, \\ \partial_t E + \partial_r \left(\dfrac{q}{\rho} (E + p) \right) = -\dfrac{2}{r} \dfrac{q}{\rho} (E + p) - q \partial_r \phi, \\ \dfrac{1}{r^2} \partial_{rr} (r^2 \phi) = 4\pi G \rho, \end{cases}$$

The steady solutions at rest are given by

$$q = 0; \qquad \partial_r p + \rho \partial_r \phi = 0; \qquad \partial_{rr} (r^2 \phi) = 4\pi r^2 G \rho.$$

We consider two cases:

- a polytropic pressure law $p(\rho; \kappa, \gamma) = \kappa \rho^\gamma$ such that the steady solutions satisfy

$$\frac{d}{dr} \left(r^2 \kappa \gamma \rho^{\gamma - 2} \frac{d\rho}{dr} \right) = 4\pi r^2 G \rho,$$

Euler-Poisson in spherical geometry: steady solution

Training takes about 10 minutes on an old GPU, with **no data**, only the PINN loss. This time, we have two parameters, κ and γ.

We take a quadrature of degree $n_Q = n_G + 1$.

Results for the **polytropic pressure law**

pts	error_ϕ^h	order	error_ϕ^q	order	error_ϕ^E	order	$\text{error}_{\bar{\phi}}^h$	order	gain	$\text{error}_{\bar{\phi}}^q$	order	gain	$\text{error}_{\bar{\phi}}^E$	order	gain
10	1.90e-01	—	1.84e-02	—	4.88e-01	—	5.84e-04	—	326.34	6.32e-03	—	2.92	1.46e-03	—	333.51
20	6.78e-02	1.49	7.60e-03	1.28	1.71e-01	1.51	2.73e-04	1.10	248.20	1.67e-03	1.92	4.55	6.84e-04	1.10	250.74
40	2.41e-02	1.49	2.93e-03	1.37	6.07e-02	1.50	1.01e-04	1.43	237.53	3.75e-04	2.15	7.80	2.54e-04	1.43	238.71
80	8.55e-03	1.50	1.16e-03	1.34	2.15e-02	1.50	3.64e-05	1.48	234.68	8.15e-05	2.20	14.23	9.12e-05	1.48	236.10
160	3.03e-03	1.50	4.64e-04	1.32	7.58e-03	1.51	1.17e-05	1.63	257.14	1.60e-05	2.35	28.97	2.94e-05	1.63	257.38

(a) errors with a one-element basis, $n_G = 1$

Euler-Poisson in spherical geometry: steady solution

Training takes about 10 minutes on an old GPU, with **no data**, only the PINN loss. This time, we have two parameters, κ and γ.

We take a quadrature of degree $n_Q = n_G + 1$.

Results for the **polytropic pressure law**

pts	error_ϕ^h	order	error_ϕ^q	order	error_ϕ^E	order	$\text{error}_{\bar{\phi}}^h$	order	gain	$\text{error}_{\bar{\phi}}^q$	order	gain	$\text{error}_{\bar{\phi}}^E$	order	gain
10	3.72e-03	—	5.34e-03	—	6.49e-03	—	3.74e-05	—	99.38	4.70e-05	—	113.63	9.19e-05	—	70.67
20	6.59e-04	2.50	1.21e-03	2.14	1.21e-03	2.42	7.00e-06	2.42	94.19	1.28e-05	1.87	94.14	1.68e-05	2.45	72.07
40	1.17e-04	2.49	2.27e-04	2.41	2.21e-04	2.45	1.27e-06	2.45	91.93	2.56e-06	2.33	88.59	3.07e-06	2.45	71.84
80	2.06e-05	2.51	4.05e-05	2.49	3.86e-05	2.52	2.24e-07	2.51	92.05	4.70e-07	2.45	86.03	5.45e-07	2.50	70.86
160	3.64e-06	2.51	7.15e-06	2.50	6.56e-06	2.56	3.90e-08	2.52	93.17	8.27e-08	2.51	86.41	9.50e-08	2.52	69.08

(b) errors with a two-element basis, $n_G = 2$

Euler-Poisson in spherical geometry: steady solution

Training takes about 10 minutes on an old GPU, with **no data**, only the PINN loss. This time, we have two parameters, κ and γ.

We take a quadrature of degree $n_Q = n_G + 1$.

Results for the **polytropic pressure law**

pts	error_{ϕ}^{h}	order	error_{ϕ}^{q}	order	error_{ϕ}^{E}	order	$\text{error}_{\bar{\phi}}^{h}$	order	gain	$\text{error}_{\bar{\phi}}^{q}$	order	gain	$\text{error}_{\bar{\phi}}^{E}$	order	gain
10	7.92e-06	—	5.39e-06	—	3.25e-04	—	3.68e-06	—	2.15	3.16e-06	—	1.71	8.16e-06	—	39.81
20	6.96e-07	3.51	9.10e-07	2.57	3.39e-05	3.26	3.60e-07	3.36	1.93	6.02e-07	2.39	1.51	7.41e-07	3.46	45.79
40	6.03e-08	3.53	9.46e-08	3.27	3.21e-06	3.40	3.26e-08	3.47	1.85	5.64e-08	3.42	1.68	7.74e-08	3.26	41.47
80	5.31e-09	3.51	7.97e-09	3.57	2.84e-07	3.50	2.98e-09	3.45	1.78	5.07e-09	3.47	1.57	7.09e-09	3.45	40.15
160	4.81e-10	3.46	7.26e-10	3.46	2.51e-08	3.50	2.74e-10	3.45	1.76	4.61e-10	3.46	1.57	6.46e-10	3.46	39.00

(c) errors with a three-element basis, $n_G = 3$

Euler-Poisson in spherical geometry: steady solution

Training takes about 10 minutes on an old GPU, with **no data**, only the PINN loss. This time, we have two parameters, κ and γ.

We take a quadrature of degree $n_Q = n_G + 1$.

Results for the **polytropic pressure law**

Statistics: gain with respect to the parameter space (from top to bottom: $n_G = 1, n_G = 2, n_G = 3$)

	min. gain	avg. gain	max. gain
ρ	22.21	412.57	6080.00
q	40.90	411.13	5384.43
E	22.25	411.40	6014.11

	min. gain	avg. gain	max. gain
ρ	6.57	154.29	1249.70
q	7.47	180.19	1317.09
E	6.14	110.27	627.65

	min. gain	avg. gain	max. gain
ρ	0.17	12.80	102.00
q	0.20	14.12	109.50
E	3.69	48.66	433.81

2D shallow water system

We consider the 2D shallow water equations

$$\begin{cases} \partial_t h + \nabla \cdot q = 0 \\ \partial_t q + \nabla \cdot \left(\dfrac{q \otimes q}{h} + \dfrac{1}{2} g h^2 \right) = -gh \nabla Z(x, y; \alpha, r_0) \end{cases}$$

We define the following compactly supported bump function:

$$\Omega(x, y; \alpha, r_0) = \begin{cases} \alpha \exp\left(\dfrac{-1}{\left(1 - \dfrac{r^2}{r_0^2}\right)^3} \right) & \text{if } r < r_0, \\ 0 & \text{otherwise,} \end{cases}$$

and we take $Z(x, y; \alpha, r_0) = \Omega(x, y; \alpha, r_0)$.

The steady solution is a **vortex**, whose amplitude and radius depend on α, r_0 and an additional parameter Γ: this time, we have three parameters, in addition to x and y.

2D shallow water system: steady solution

Training takes about 20 minutes on an old GPU, with the PINN loss **supplemented with data**.

We need a high-quadrature, of degree $n_Q = 14$, because of the large derivatives of the compactly supported smooth bump function.

pts	error_{ϕ}^{h}	order	$\text{error}_{\phi}^{q_x}$	order	$\text{error}_{\phi}^{q_y}$	order	$\text{error}_{\bar{\phi}}^{h}$	order	gain	$\text{error}_{\bar{\phi}}^{q_x}$	order	gain	$\text{error}_{\bar{\phi}}^{q_y}$	order	gain
20	1.91e-01	—	1.13e+00	—	1.13e+00	—	2.31e-03	—	82.79	1.02e-03	—	1116.93	1.01e-03	—	1119.33
40	4.72e-02	2.02	2.76e-01	2.04	2.76e-01	2.04	5.85e-04	1.98	80.64	2.30e-04	2.15	1199.70	2.22e-04	2.19	1242.66
80	1.16e-02	2.02	6.71e-02	2.04	6.71e-02	2.04	1.46e-04	2.00	79.77	5.72e-05	2.01	1173.39	5.52e-05	2.01	1216.72
160	2.90e-03	2.00	1.68e-02	1.99	1.68e-02	1.99	3.66e-05	2.00	79.45	1.43e-05	2.00	1178.29	1.38e-05	2.00	1222.59

(a) errors with a one-element basis, $n_G = 1$

2D shallow water system: steady solution

Training takes about 20 minutes on an old GPU, with the PINN loss **supplemented with data**.

We need a high-quadrature, of degree $n_Q = 14$, because of the large derivatives of the compactly supported smooth bump function.

pts	error$^h_\phi$	order	error$^{q_x}_\phi$	order	error$^{q_y}_\phi$	order	error$^h_{\bar{\phi}}$	order	gain	error$^{q_x}_{\bar{\phi}}$	order	gain	error$^{q_y}_{\bar{\phi}}$	order	gain
20	2.32e-02	—	2.10e-01	—	2.10e-01	—	2.59e-04	—	89.71	5.49e-04	—	382.67	5.73e-04	—	367.32
40	3.60e-03	2.69	2.86e-02	2.88	2.86e-02	2.88	3.15e-05	3.04	114.33	4.24e-05	3.70	675.67	4.30e-05	3.73	665.36
80	5.28e-04	2.77	3.56e-03	3.01	3.57e-03	3.01	3.95e-06	2.99	133.61	6.07e-06	2.80	587.71	6.16e-06	2.80	578.89
160	7.02e-05	2.91	4.63e-04	2.94	4.63e-04	2.94	4.96e-07	2.99	141.49	7.90e-07	2.94	586.16	8.02e-07	2.94	577.49

(b) errors with a two-element basis, $n_G = 2$

2D shallow water system: steady solution

Training takes about 20 minutes on an old GPU, with the PINN loss **supplemented with data**.

We need a high-quadrature, of degree $n_Q = 14$, because of the large derivatives of the compactly supported smooth bump function.

pts	error$^{h}_{\phi}$	order	error$^{q_x}_{\phi}$	order	error$^{q_y}_{\phi}$	order	error$^{h}_{\bar{\phi}}$	order	gain	error$^{q_x}_{\bar{\phi}}$	order	gain	error$^{q_y}_{\bar{\phi}}$	order	gain
20	5.17e-03	—	6.05e-02	—	6.05e-02	—	3.05e-04	—	16.97	1.63e-03	—	37.11	1.60e-03	—	37.72
40	4.32e-04	3.58	4.35e-03	3.80	4.34e-03	3.80	2.07e-06	7.20	208.24	4.35e-06	8.55	999.02	4.47e-06	8.49	969.66
80	2.87e-05	3.91	2.73e-04	3.99	2.73e-04	3.99	1.30e-07	3.99	220.41	2.84e-07	3.94	961.63	2.89e-07	3.95	942.16
160	1.72e-06	4.06	1.81e-05	3.91	1.81e-05	3.91	8.17e-09	3.99	210.88	1.59e-08	4.15	1136.57	1.62e-08	4.16	1117.87

(c) errors with a three-element basis, $n_G = 3$

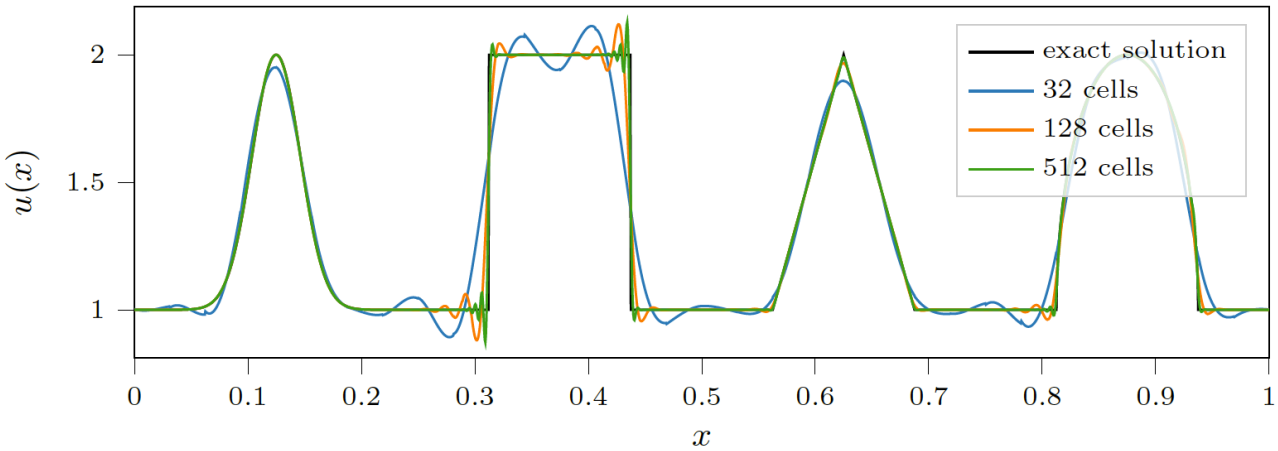
Second enhanced DG schemes

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

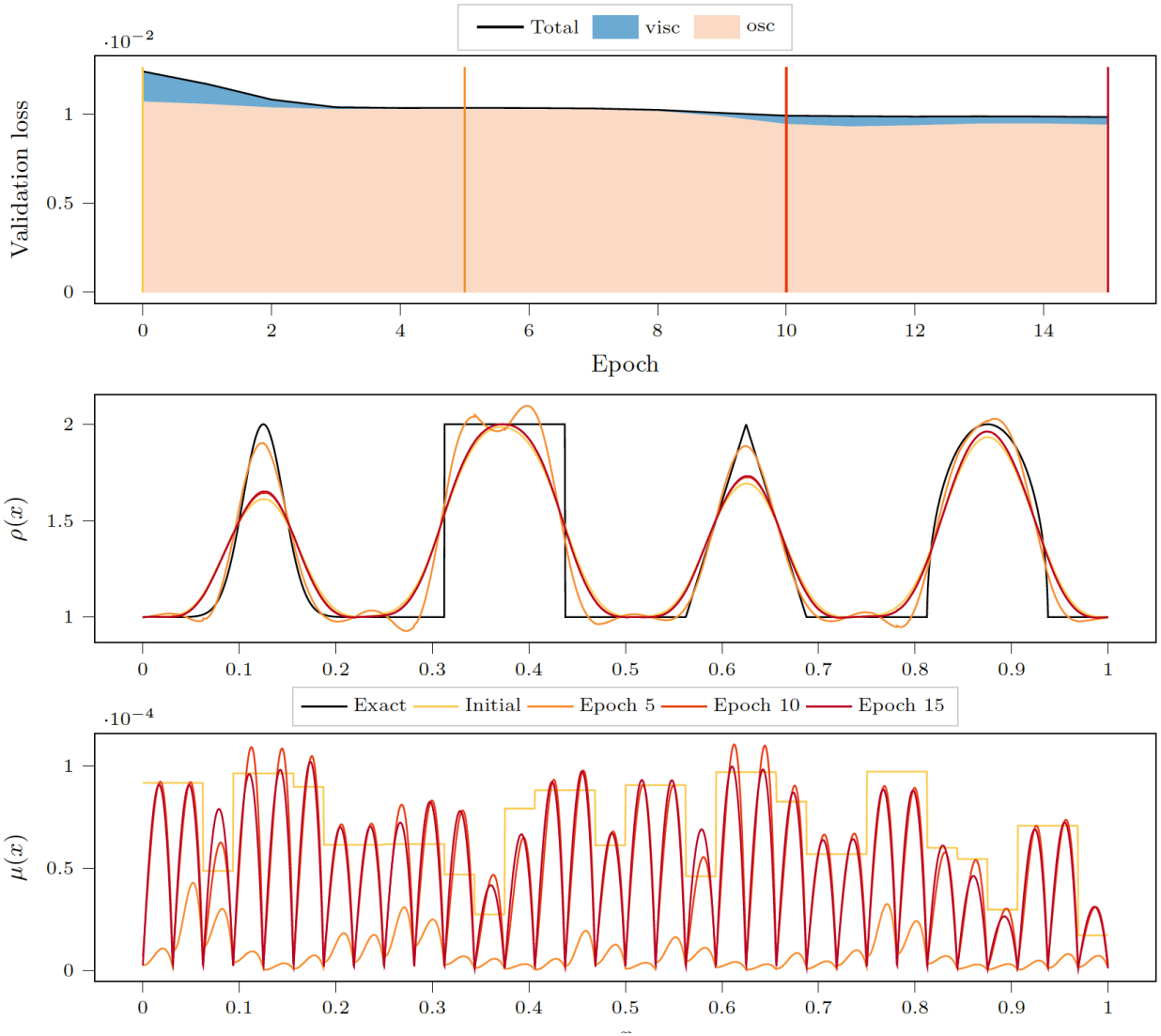
$$C_{\text{osc}}(\mathbf{U}_h^n) = h_{fv} \sum_{i=1}^{n} \left\| D_{xx}^{fv}(\Pi_{fv}(\mathbf{U}_h^n))_i - D_{xx}^{fv}\mathbf{U}_{j,ref} \right\|_1 .$$

- L^2 norm of D_θ:

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

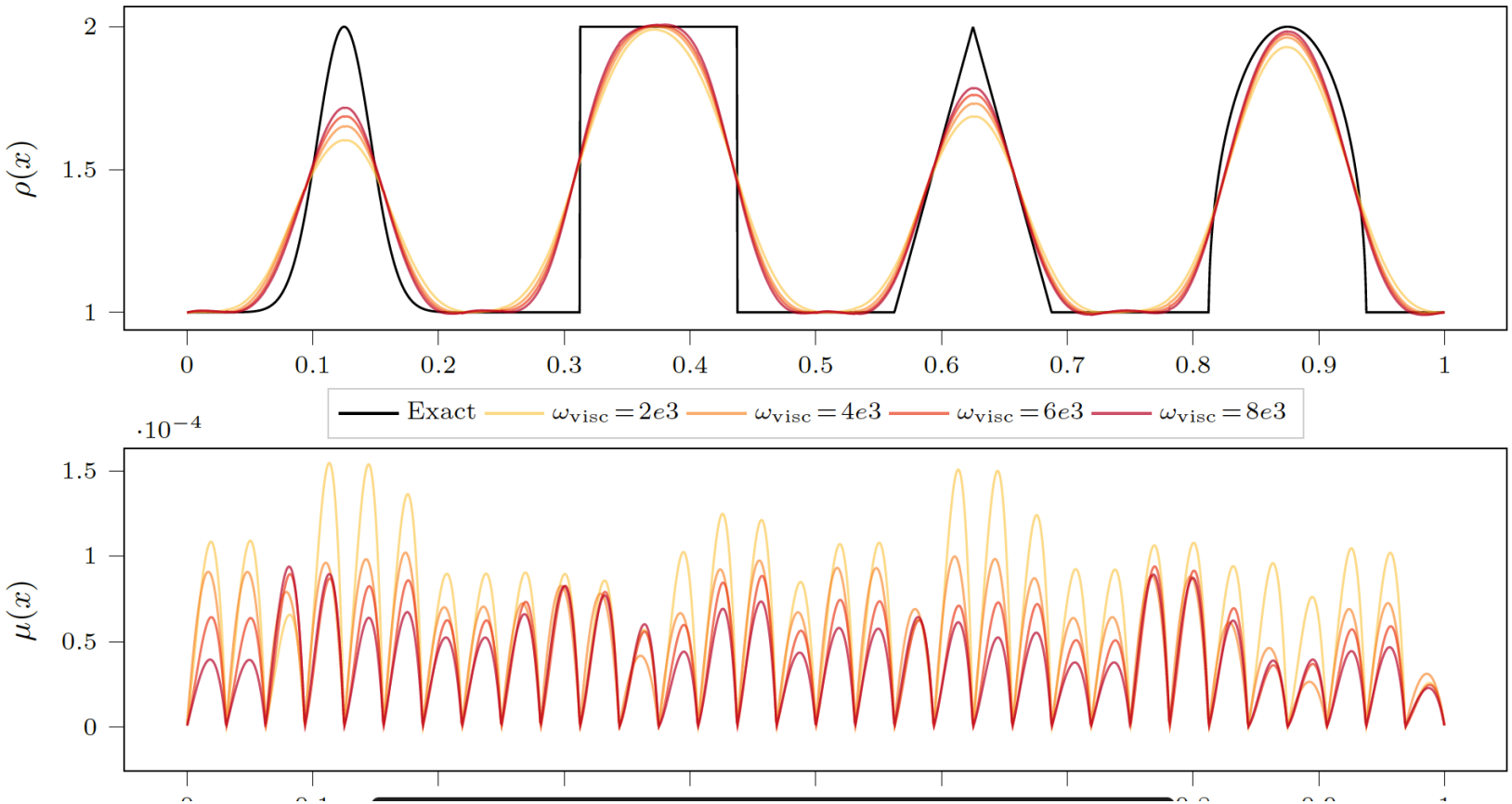
Results I

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



Results II

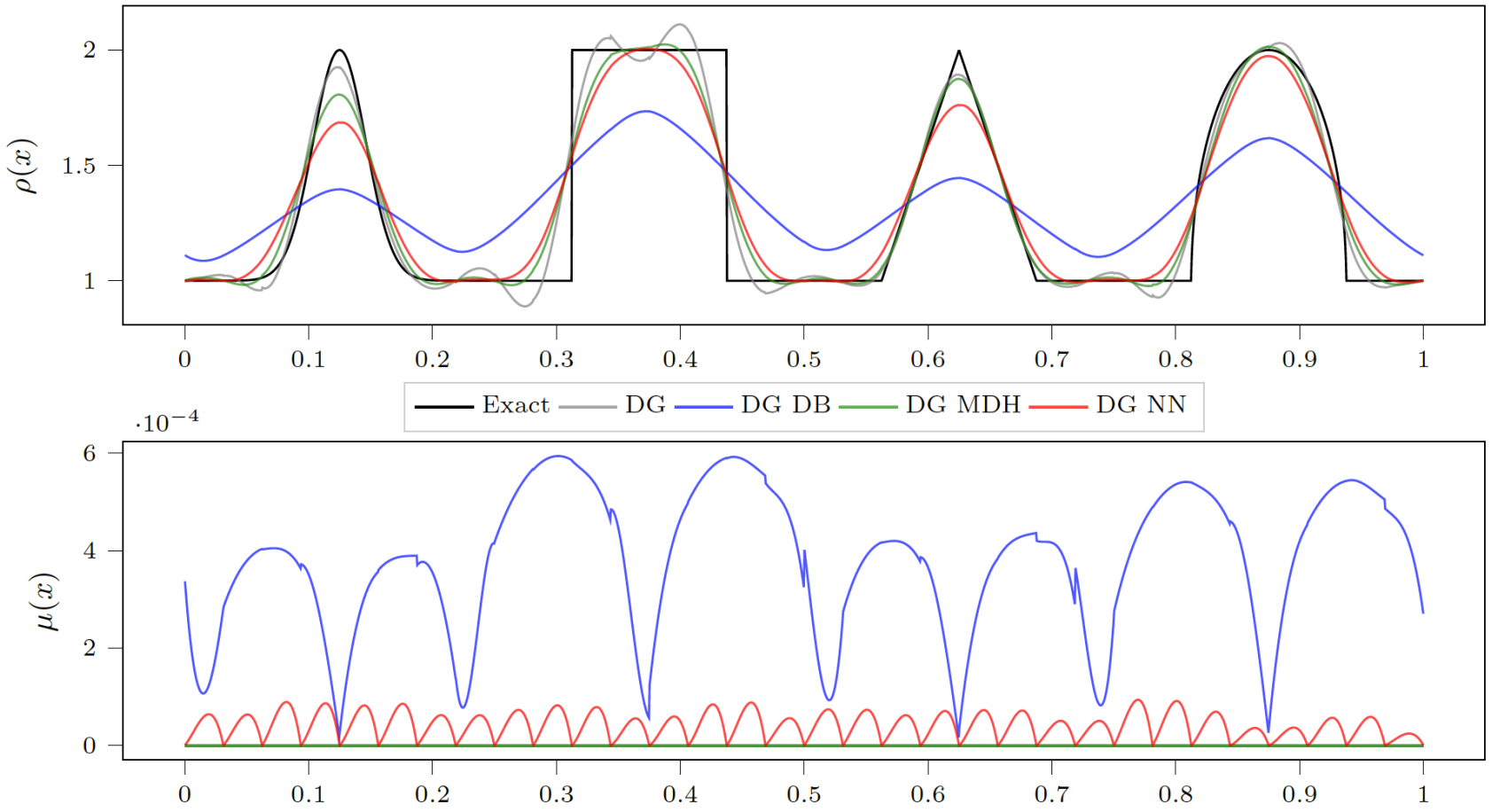
- We compare the training for different ratio loss.
- We fixe the weight of the oscillation loss.



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

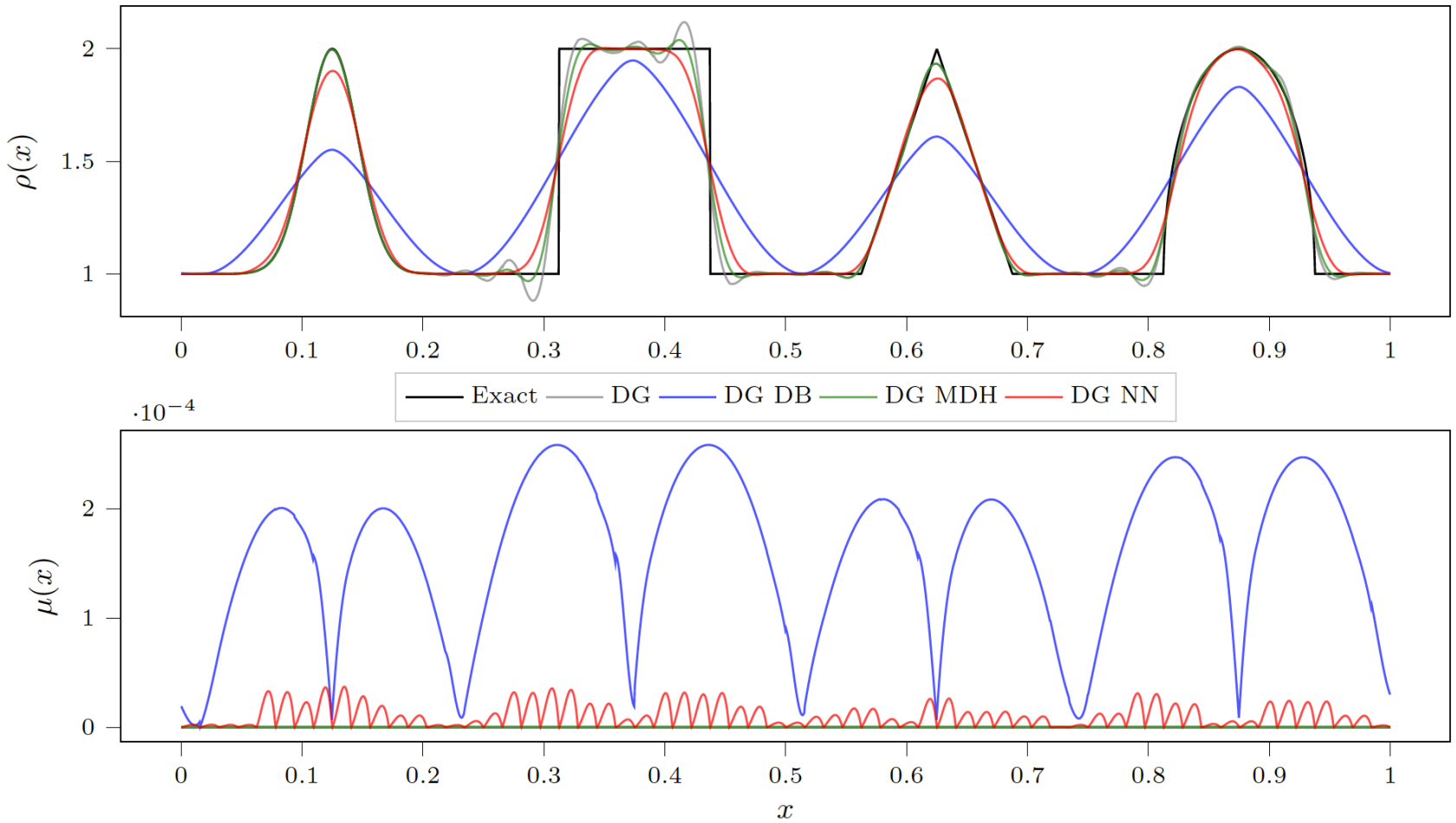
Results III

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



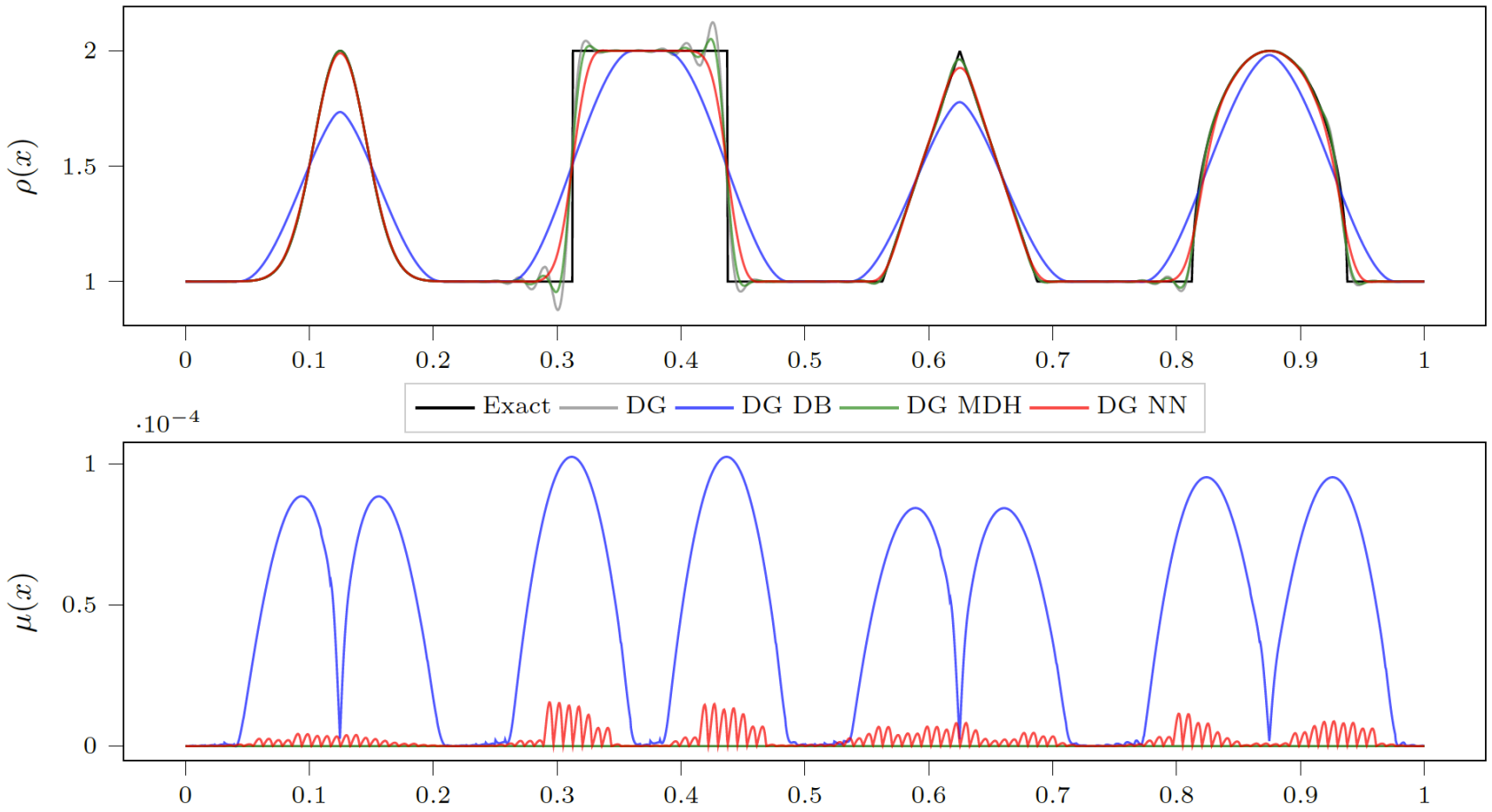
Results III

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



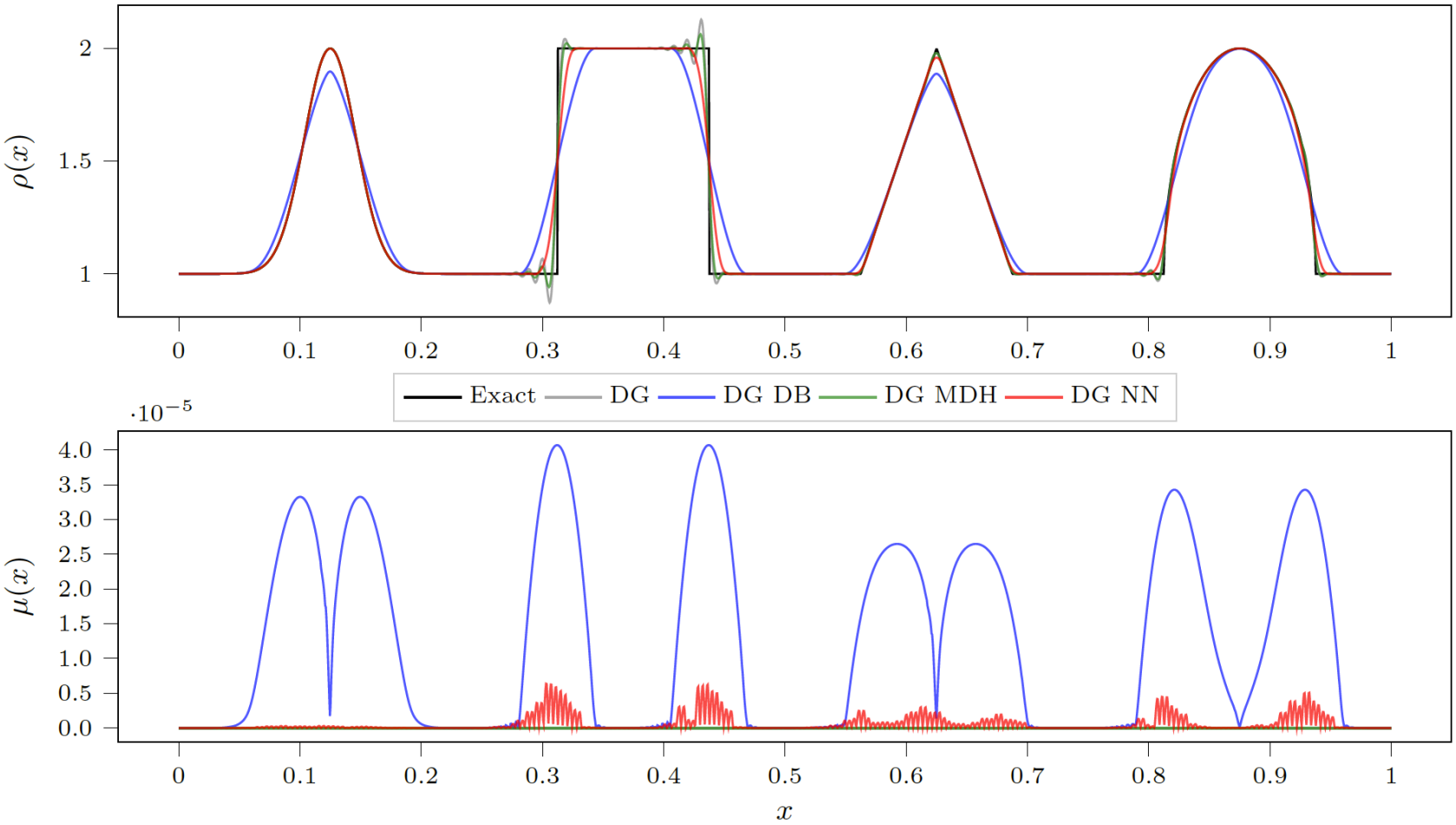
Results III

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



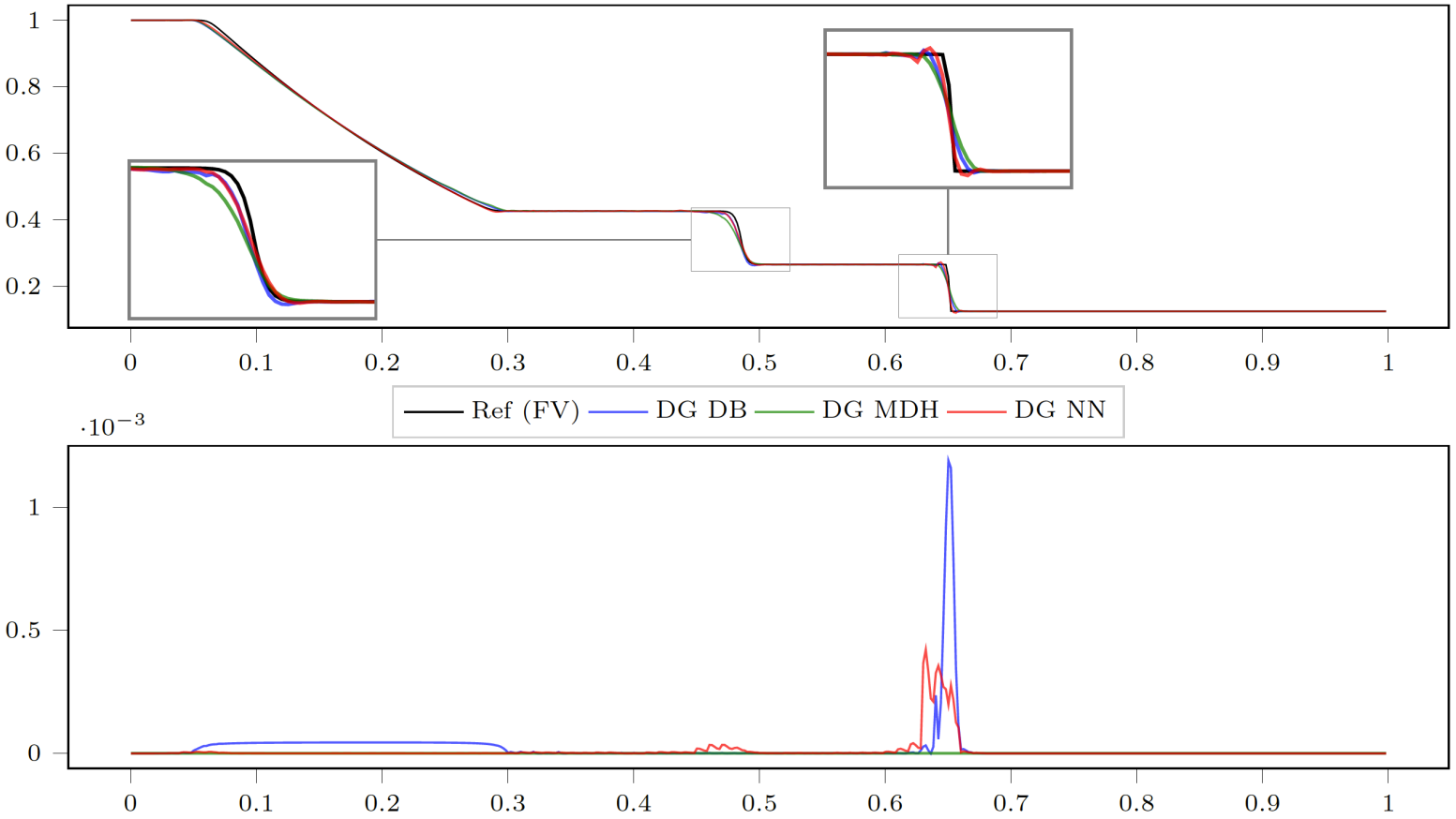
Results III

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



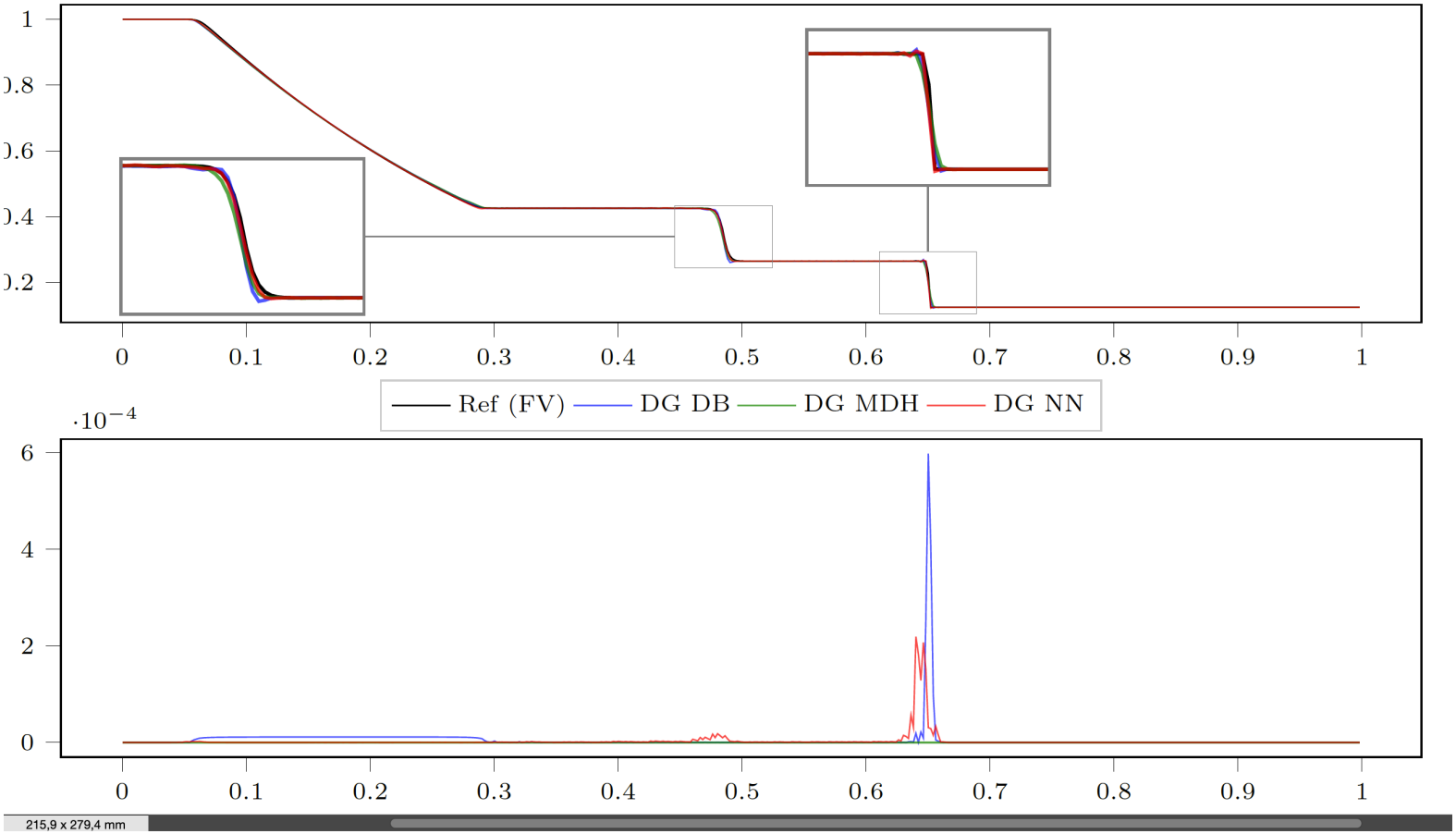
Results III

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



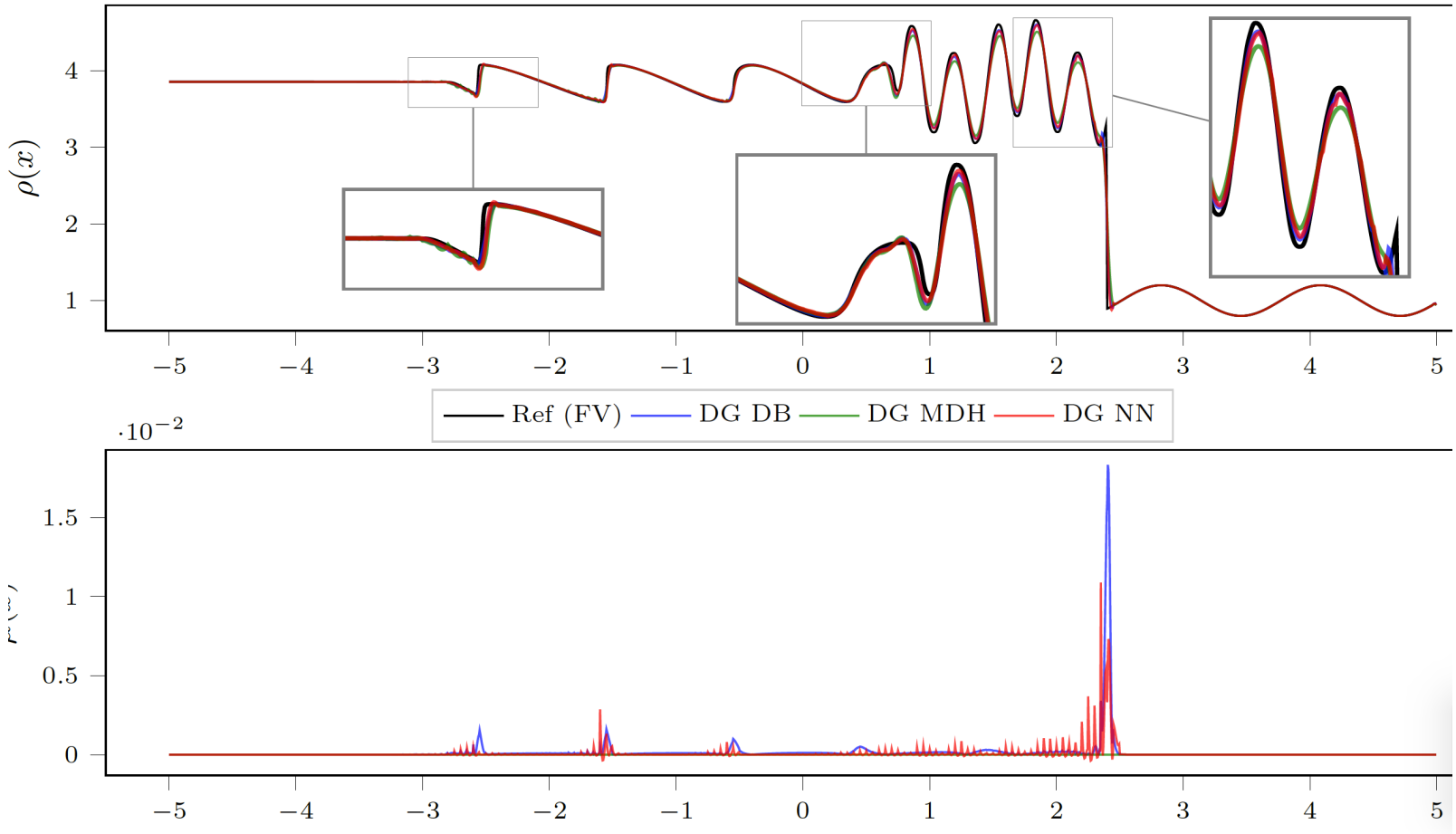
Results III

- 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 and futur works

Conclusion

Deep learning

The deep learning approaches give news tools for large dimensional problem (NN) and optimization (autodiff tools)

WB

Using a "offline prediction online corrector" method where we compute a "large dimensional" equilibrium family offline and use it online to solve a perturbative flow we obtain a very efficient scheme for complex equilibrium for hyperbolic systems.

Viscosity

Using the autodiff tools we compute a new viscosity model without reference viscosity models taking into account to the effect of the viscosity model on the simulation in time.

Full hybrid code

- We want in the futur design a "full hybrid code".
- **Modeling**:

$$\partial_t \mathbf{U} + \nabla \cdot F_\theta(\mathbf{U}) = \nabla \cdot (D_\theta(\mathbf{U}) \nabla(\mathbf{U}))$$

with $F_\theta = F_{local}(\mathbf{U}) + F_f(\mathbf{U}) \star \mathbf{U}$, $D_\theta = D_{local}(\mathbf{U}) + K_g(\mathbf{U}) \star$.

- All the terms can be analytic, partial learned or fully learned with neural networks, symbolic models (Sindy etc) imposing or not specific structures.
- **Resolution**: we can partial or fully learn the fluxes, the basis, change of variables etc.
- We can pre-train the network (as in the WB project) or training solving the scheme (as in the viscosity project)
- We want apply this to hyperbolic systems and general moment models for kinetic equations.