Hybrid methods for elliptic and hyperbolic PDEs

Victor Michel-Dansac[*], joint work with
Hélène Barucq[†], Michel Duprez[‡], Florian Faucher[†], Emmanuel Franck[*],
Frédérique Lecourtier[‡], Vanessa Lleras[§], Laurent Navoret[*], Nicolas Victorion[†]

Friday, February 07, 2025
Séminaire de l'équipe EDPs² du LAMA, Chambéry

[*] MACARON project-team, Université de Strasbourg, CNRS, Inria, IRMA, France
[†] MAKUTU project-team, Université de Pau et des Pays de l'Adour, CNRS, Inria, France
[‡] MIMESIS project-team, Université de Strasbourg, CNRS, Inria, France
[§] IMAG, Université de Montpellier, France

Inria

IRMA
Institut de Recherche
Mathématique Avancée

Problem under consideration

To fix notation, consider the following stationary PDE:

$$\begin{cases} \mathcal{D}(W, x) = 0 & \text{for } x \in \Omega, \\ W(x) = g(x) & \text{for } x \in \partial\Omega, \end{cases}$$

where

- $\Omega \subset \mathbb{R}^d$ is the spatial domain,
- $W \in \mathbb{R}^q$ is the unknown solution,
- $\mathcal{D}$ is some differential operator,
- g is a known function.

Common framework for some classical methods

Parametric approximation in some classical methods for stationary PDEs

In many classical numerical methods, the solution is approximated by a parametric function, **linear in its parameters**, and a basis $(\varphi_j)_j$ depending on the chosen method:

$$W_\theta(x) = \sum_{j=1}^{N} \theta_j \varphi_j(x); \quad \text{the parameters } (\theta_j)_{j \in \{1,\dots,N\}} \text{ are called degrees of freedom.}$$

Common framework for some classical methods

Parametric approximation in some classical methods for stationary PDEs

In many classical numerical methods, the solution is approximated by a parametric function, **linear in its parameters**, and a basis $(\varphi_j)_j$ depending on the chosen method:

$$W_\theta(x) = \sum_{j=1}^{N} \theta_j \varphi_j(x); \quad \text{the parameters } (\theta_j)_{j \in \{1,\dots,N\}} \text{ are called degrees of freedom.}$$

- mesh-based methods, with mesh $(x_j)_{j \in \{1,\dots,N\}}$:
  - **finite difference**: $\varphi_j = \delta_{x_j}$
  - **1st-order finite volume**: φ_j are piecewise constant
  - **finite element**: φ_j are piecewise polynomial

Common framework for some classical methods

Parametric approximation in some classical methods for stationary PDEs

In many classical numerical methods, the solution is approximated by a parametric function, **linear in its parameters**, and a basis $(\varphi_j)_j$ depending on the chosen method:

$$W_\theta(x) = \sum_{j=1}^{N} \theta_j \varphi_j(x); \quad \text{the parameters } (\theta_j)_{j\in\{1,\dots,N\}} \text{ are called degrees of freedom.}$$

- mesh-based methods, with mesh $(x_j)_{j\in\{1,\dots,N\}}$:
  - **finite difference**: $\varphi_j = \delta_{x_j}$
  - **1st-order finite volume**: φ_j are piecewise constant
  - **finite element**: φ_j are piecewise polynomial
- mesh-free methods:
  - **spectral methods**: $\varphi_j = e^{ik_j x}$ in Fourier space
  - **SPH**: $\varphi_j(x) = \Xi(|x - x_j|)$ with Ξ a kernel function
  - **diffuse elements**: φ_j are piecewise polynomial

Common framework for some classical methods

Parametric approximation in some classical methods for stationary PDEs

In many classical numerical methods, the solution is approximated by a parametric function, **linear in its parameters**, and a basis $(\varphi_j)_j$ depending on the chosen method:

$$W_\theta(x) = \sum_{j=1}^{N} \theta_j \varphi_j(x); \quad \text{the parameters } (\theta_j)_{j\in\{1,\dots,N\}} \text{ are called degrees of freedom.}$$

- mesh-based methods, with mesh $(x_j)_{j\in\{1,\dots,N\}}$:
  - **finite difference**: $\varphi_j = \delta_{x_j}$
  - **1st-order finite volume**: φ_j are piecewise constant
  - **finite element**: φ_j are piecewise polynomial
- mesh-free methods:
  - **spectral methods**: $\varphi_j = e^{ik_j x}$ in Fourier space
  - **SPH**: $\varphi_j(x) = \Xi(|x - x_j|)$ with Ξ a kernel function
  - **diffuse elements**: φ_j are piecewise polynomial

Most of these approaches are **local in space**, and the number of degrees of freedom N **exponentially increases with the dimension**.

Example: the finite element method

Consider the **Poisson problem** and its **weak formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad \forall \psi \in \mathcal{H}_0^1(\Omega), \quad \int_\Omega \nabla W \cdot \nabla \psi \, dx = \int_\Omega f \psi \, dx.$$

Example: the finite element method

Consider the **Poisson problem** and its **weak formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad \forall \psi \in \mathcal{H}_0^1(\Omega), \quad \int_\Omega \nabla W \cdot \nabla \psi \, dx = \int_\Omega f \psi \, dx.$$

Now, approximate $\mathcal{H}_0^1(\Omega)$ by a **linear subspace** of polynomial functions $V = \mathrm{Span}((\varphi_j)_j)$.

The **finite element approximation** of W is, for $x \in \Omega$, $W_\theta(x) = \sum_{j=1}^{N} \theta_j \varphi_j(x)$.

Example: the finite element method

Consider the **Poisson problem** and its **weak formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad \forall \psi \in \mathcal{H}_0^1(\Omega), \quad \int_\Omega \nabla W \cdot \nabla \psi \, dx = \int_\Omega f \psi \, dx.$$

Now, approximate $\mathcal{H}_0^1(\Omega)$ by a **linear subspace** of polynomial functions $V = \mathrm{Span}((\varphi_j)_j)$.

The **finite element approximation** of W is, for $x \in \Omega$, $W_\theta(x) = \sum_{j=1}^N \theta_j \varphi_j(x)$.

Plugging these approximations in the weak formulation, we get

$$\forall k \in \{1, \dots, N\}, \quad \sum_{j=1}^N \theta_j \underbrace{\int_\Omega \nabla \varphi_j \cdot \nabla \varphi_k \, dx}_{A_{kj}} = \underbrace{\int_\Omega f \varphi_k \, dx}_{b_k},$$

i.e., with $\theta = (\theta_j)_j$, $A = (A_{kj})_{kj}$ and $b = (b_k)_k$, we have the **linear system** $A\theta = b$.

Example: the Ritz-Galerkin method

Consider the **Poisson problem** and its **energy formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

Example: the Ritz-Galerkin method

Consider the **Poisson problem** and its **energy formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

Now, approximate $\mathcal{H}_0^1(\Omega)$ by a **linear subspace** of polynomial functions $V = \operatorname{Span}((\varphi_j)_j)$.

The **finite element approximation** of W is, for $x \in \Omega$, $W_\theta(x) = \sum_{j=1}^N \theta_j \varphi_j(x)$. Therefore,

$$W_\theta = \underset{\varphi \in V}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \varphi|^2 - \int_\Omega f \varphi \right].$$

Example: the Ritz-Galerkin method

Consider the **Poisson problem** and its **energy formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

Now, approximate $\mathcal{H}_0^1(\Omega)$ by a **linear subspace** of polynomial functions $V = \operatorname{Span}((\varphi_j)_j)$.

The **finite element approximation** of W is, for $x \in \Omega$, $W_\theta(x) = \sum_{j=1}^N \theta_j \varphi_j(x)$. Therefore,

$$\theta = \underset{\vartheta \in \mathbb{R}^N}{\operatorname{argmin}} \left[\sum_{j=1}^N \sum_{k=1}^N \frac{1}{2} \vartheta_j \vartheta_k \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_k - \sum_{j=1}^N \vartheta_j \int_\Omega f \varphi_j \right].$$

We can write $\theta = \operatorname{argmin}_{\vartheta \in \mathbb{R}^N} \mathcal{J}(\vartheta)$, with $\mathcal{J}$ a quadratic function.

Solving this **quadratic minimization problem**, we obtain the same **linear system** $A\theta = b$.

Example: the Ritz-Galerkin method

Consider the **Poisson problem** and its **energy formulation**, with unknown $W \in \mathcal{H}_0^1(\Omega)$:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

Now, approximate $\mathcal{H}_0^1(\Omega)$ by a $\boxed{\textbf{linear subspace} \text{ of polynomial functions } V = \operatorname{Span}((\varphi_j)_j)}$.

The **finite element approximation** of W is, for $x \in \Omega$, $W_\theta(x) = \sum_{j=1}^N \theta_j \varphi_j(x)$. Therefore,

$$\theta = \underset{\vartheta \in \mathbb{R}^N}{\operatorname{argmin}} \left[\sum_{j=1}^N \sum_{k=1}^N \frac{1}{2} \vartheta_j \vartheta_k \int_\Omega \nabla \varphi_j \cdot \nabla \varphi_k - \sum_{j=1}^N \vartheta_j \int_\Omega f \varphi_j \right].$$

We can write $\theta = \operatorname{argmin}_{\vartheta \in \mathbb{R}^N} \mathcal{J}(\vartheta)$, with $\mathcal{J}$ a quadratic function.

Solving this **quadratic minimization problem**, we obtain the same **linear system** $A\theta = b$.

Example: the "Deep Ritz" method [W. E and B. Yu (2018)]

Consider the **Poisson problem** and its **energy formulation**:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

Example: the “Deep Ritz” method [W. E and B. Yu (2018)]

Consider the **Poisson problem** and its **energy formulation**:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

We approximate $\mathcal{H}_0^1(\Omega)$ by the subspace $V = \{x \mapsto \varphi(x, \theta),\ \theta \in \mathbb{R}^N\}$, with $\varphi : \mathbb{R}^d \times \mathbb{R}^N \to \mathbb{R}$ a **nonlinear function** of both inputs.

The **nonlinear approximation** of W becomes, for $x \in \Omega$, $W_\theta(x) = \varphi(x, \theta)$. Therefore,

$$W_\theta = \underset{\varphi \in V}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \varphi(x, \theta)|^2 \, dx - \int_\Omega f(x) \varphi(x, \theta) \, dx \right].$$

Example: the "Deep Ritz" method [W. E and B. Yu (2018)]

Consider the **Poisson problem** and its **energy formulation**:

$$\begin{cases} -\Delta W = f & \text{in } \Omega, \\ W = 0 & \text{on } \partial\Omega, \end{cases} \qquad \Longleftrightarrow \qquad W = \underset{\psi \in \mathcal{H}_0^1(\Omega)}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \psi|^2 - \int_\Omega f \psi \right].$$

We approximate $\mathcal{H}_0^1(\Omega)$ by the subspace $V = \{x \mapsto \varphi(x, \theta),\ \theta \in \mathbb{R}^N\}$, with $\varphi : \mathbb{R}^d \times \mathbb{R}^N \to \mathbb{R}$ a **nonlinear function** of both inputs.

The **nonlinear approximation** of W becomes, for $x \in \Omega$, $W_\theta(x) = \varphi(x, \theta)$. Therefore,

$$\theta = \underset{\vartheta \in \mathbb{R}^N}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla \varphi(x, \vartheta)|^2 \, dx - \int_\Omega f(x) \varphi(x, \vartheta) \, dx \right].$$

We can write $\theta = \operatorname{argmin}_{\vartheta \in \mathbb{R}^N} \mathcal{J}(\vartheta)$, with $\mathcal{J}$ a nonquadratic function.

We now have to solve a **nonlinear minimization problem**!

Summary

We have presented two ways of approximating our unknown function W. In both cases, we define degrees of freedom $\theta \in \mathbb{R}^N$, and set $W(x) \simeq W_\theta(x) = \varphi(x, \theta)$, for all $x \in \Omega$.

The main difference lies in the **choice of the function** φ: it is always **nonlinear in space**, but its **behavior with respect to** θ changes the nature of the approximation problem.

Summary

We have presented two ways of approximating our unknown function W. In both cases, we define degrees of freedom $\theta \in \mathbb{R}^N$, and set $W(x) \simeq W_\theta(x) = \varphi(x, \theta)$, for all $x \in \Omega$.

The main difference lies in the **choice of the function** φ: it is always **nonlinear in space**, but its **behavior with respect to** θ changes the nature of the approximation problem.

1. φ is linear in θ , $\varphi(x, \theta) = \sum_{j=1}^{N} \theta_j \varphi_j(x)$:
   1.1 W is projected onto a finite-dimensional linear subspace
   1.2 solve a convex quadratic optimization problem to determine θ

Summary

We have presented two ways of approximating our unknown function W. In both cases, we define degrees of freedom $\theta \in \mathbb{R}^N$, and set $W(x) \simeq W_\theta(x) = \varphi(x, \theta)$, for all $x \in \Omega$.

The main difference lies in the **choice of the function** φ: it is always **nonlinear in space**, but its **behavior with respect to** θ changes the nature of the approximation problem.

1. φ is linear in θ , $\varphi(x, \theta) = \sum_{j=1}^{N} \theta_j \varphi_j(x)$:
   1.1 W is projected onto a finite-dimensional linear subspace
   1.2 solve a convex quadratic optimization problem to determine θ

2. φ is nonlinear in θ:
   2.1 W is projected onto a finite-dimensional "submanifold"
   2.2 solve a nonlinear, (usually) non-convex optimization problem to determine θ

Summary

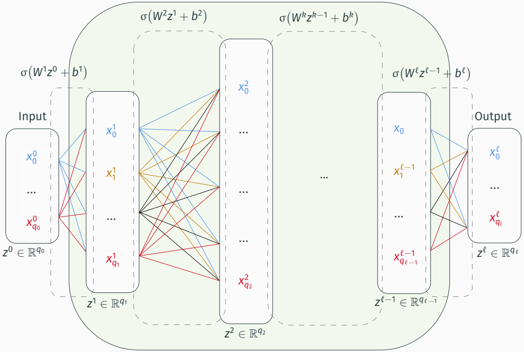
We have presented two ways of approximating our unknown function W. In both cases, we define degrees of freedom $\theta \in \mathbb{R}^N$, and set $W(x) \simeq W_\theta(x) = \varphi(x, \theta)$, for all $x \in \Omega$.

The main difference lies in the **choice of the function** φ: it is always **nonlinear in space**, but its **behavior with respect to** θ changes the nature of the approximation problem.

1. φ is linear in θ , $\varphi(x, \theta) = \sum_{j=1}^{N} \theta_j \varphi_j(x)$:
   1.1 W is projected onto a finite-dimensional linear subspace
   1.2 solve a convex quadratic optimization problem to determine θ
2. φ is nonlinear in θ:
   2.1 W is projected onto a finite-dimensional "submanifold"
   2.2 solve a nonlinear, (usually) non-convex optimization problem to determine θ

Question: How to construct suitable nonlinear functions φ?

Multilayer perceptron (MLP)



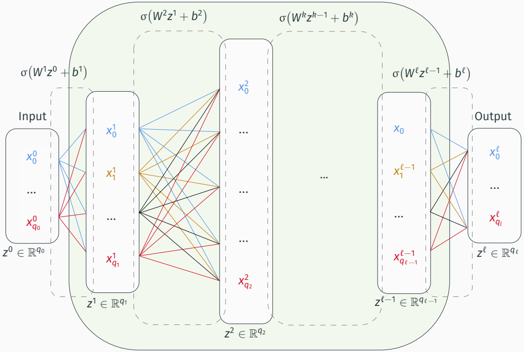
Schematic of an MLP (by A. Bélières-Frendo).

An MLP is a nonlinear parametric function $\varphi : \mathbb{R}^d \times \mathbb{R}^N \to \mathbb{R}^q$.

It results from a composition of several non-linear layers. For instance, the first layer is:

- $z^1 = \sigma(A^1 z^0 + b^1) \in \mathbb{R}^{q_1}$,
- $z^0 \in \mathbb{R}^{q_0}$ (with $q_0 = d$),
- $A^1 \in \mathcal{M}_{q_1,q_0}(\mathbb{R})$,
- $b^1 \in \mathbb{R}^{q_1}$,
- $\sigma \in \mathcal{C}^0(\mathbb{R}, \mathbb{R})$, applied component-wise.

Multilayer perceptron (MLP)



Schematic of an MLP (by A. Bélières-Frendo).

An MLP is a nonlinear parametric function $\varphi : \mathbb{R}^d \times \mathbb{R}^N \to \mathbb{R}^q$.

It results from a composition of several non-linear layers. For instance, the first layer is:

- $z^1 = \sigma(A^1 z^0 + b^1) \in \mathbb{R}^{q_1}$,
- $z^0 \in \mathbb{R}^{q_0}$ (with $q_0 = d$),
- $A^1 \in \mathcal{M}_{q_1,q_0}(\mathbb{R})$,
- $b^1 \in \mathbb{R}^{q_1}$,
- $\sigma \in \mathcal{C}^0(\mathbb{R}, \mathbb{R})$, applied component-wise.

In the end, the function $\varphi : (z^0, \theta) \mapsto z^\ell$ reads

$$\varphi(z^0, \theta) = \sigma(A^\ell \sigma(A^{\ell-1} \ldots \sigma(A^1 z^0 + b^1) \cdots + b^{\ell-1}) + b^\ell).$$

The degrees of freedom are $\theta = (A^1, b^1, \ldots, A^\ell, b^\ell) \in \mathbb{R}^N$, with $N = \sum_{i=1}^{\ell} q_i(q_{i-1} + 1)$.

Arbitrary-width case [G. Cybenko, Math. Control Signals Systems (1989)]

Let $\sigma \in \mathcal{C}^0(\mathbb{R}, \mathbb{R})$ be a non-polynomial function. Then, for all $(m, n) \in \mathbb{N}^2$, $\mathcal{K} \subseteq \mathbb{R}^n$ compact set, $f \in \mathcal{C}^0(\mathcal{K}, \mathbb{R}^m)$, and $\varepsilon > 0$, there exist $k \in \mathbb{N}$, $A \in \mathcal{M}_{k,n}(\mathbb{R})$, $b \in \mathbb{R}^k$ and $C \in \mathcal{M}_{m,k}(\mathbb{R})$ such that

$$\|f(x) - C\sigma(Ax + b)\|_{L^\infty(K)} < \varepsilon.$$

Arbitrary-depth case [P. Kidger and T. Lyons, (2020)]

Let $\sigma \in \mathcal{C}^0(\mathbb{R}, \mathbb{R})$ be a non-affine function, continuously differentiable in at least one point. Let $\mathcal{N}^\sigma_{n,m,n+m+2}$ denote the set of MLPs with n inputs, m outputs, whose hidden layers have $n + m + 2$ neurons, and with activation function σ. Then, for all $(m, n) \in \mathbb{N}^2$, $\mathcal{K} \subseteq \mathbb{R}^n$ compact set, $f \in \mathcal{C}^0(\mathcal{K}, \mathbb{R}^m)$, and $\varepsilon > 0$, there exists $W_\theta \in \mathcal{N}^\sigma_{n,m,n+m+2}$ such that

$$\|f(x) - W_\theta(x)\|_{L^\infty(K)} < \varepsilon.$$

Determination of the parameters θ – PINNs

Equipped with the expression of $W_\theta : \mathbb{R}^d \to \mathbb{R}^q$, with $W_\theta = \varphi(\cdot, \theta)$, the goal is to **determine the N parameters** θ such that W_θ is an approximation to the PDE solution W.

Equipped with the expression of $W_\theta : \mathbb{R}^d \to \mathbb{R}^q$, with $W_\theta = \varphi(\cdot, \theta)$, the goal is to **determine the N parameters** θ such that W_θ is an approximation to the PDE solution W.

This is done through nonlinear optimization[1]: define a **loss function** $\mathcal{J}$ measuring[2] the PDE residual, i.e.,

$$\mathcal{J}(\theta) = \int_\Omega \mathcal{D}(W_\theta, x)^2 \, dx + \int_{\partial\Omega} (W_\theta(x) - g(x))^2 \, dx.$$

The optimal parameters are then given by:

$$\theta_{\text{opt}} = \underset{\theta}{\operatorname{argmin}} \, \mathcal{J}(\theta).$$

[1] Usually, using the ADAM algorithm [D. Kingma and J. Ba, (2015)] for stochastic gradient descent.
[2] This corresponds to PINNs (Physics-Informed Neural Networks, M. Raissi et al., *J. Comput. Phys.* (2019)).

Determination of the parameters θ – Deep Ritz

Another way of determining parameters θ lies in the **Deep Ritz method**[3]. The solution remains approximated by a neural network $W_\theta : \mathbb{R}^d \to \mathbb{R}^q$.

This time, the PDE is written in energy form. In the case of the Poisson problem, this leads to the following minimization problem:

$$\theta = \underset{\vartheta \in \mathbb{R}^N}{\operatorname{argmin}} \left[\frac{1}{2} \int_\Omega |\nabla W_\theta(x)|^2 \, dx - \int_\Omega f(x) W_\theta(x) \, dx \right].$$

We can write $\theta = \operatorname{argmin}_{\vartheta \in \mathbb{R}^N} \mathcal{J}(\vartheta)$, which is a nonlinear optimization problem.

[3]see W. E and B. Yu, *Commun. Math. Stat.* (2018)

PINNs: recap

The PINN W_θ approximates the solution W to the BVP:

$$\begin{cases} \mathcal{D}(W,x) = 0 & \text{for } x \in \Omega, \\ W(x) = g(x) & \text{for } x \in \partial\Omega. \end{cases} \quad \rightsquigarrow \quad \begin{cases} \mathcal{D}(W_\theta,x) \simeq 0 & \text{for } x \in \Omega, \\ W_\theta(x) \simeq g(x) & \text{for } x \in \partial\Omega. \end{cases}$$

To train the PINN (i.e., to determine the optimal parameters θ_{opt}), one fashions a loss function using the PDE residual:

$$\mathcal{J}_{\text{PDE}}(\theta) = \int_\Omega \|\mathcal{D}(W_\theta,x)\|_2^2\, dx + \int_{\partial\Omega} \|W_\theta(x) - g(x)\|_2^2\, dx, \quad \text{and then} \quad \theta_{\text{opt}} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{J}_{\text{PDE}}(\theta).$$

$\rightsquigarrow$ How to compute the integrals?

Multidimensional integration

$$\mathcal{J}_{\text{PDE}}(\theta) = \underbrace{\int_\Omega \|\mathcal{D}(W_\theta, x)\|_2^2 \, dx}_{\mathcal{J}_\Omega(\theta)} + \underbrace{\int_{\partial\Omega} \|W_\theta(x) - g(x)\|_2^2 \, dx}_{\mathcal{J}_{\text{boundary}}(\theta)}$$

We have to compute two integrals:

- $\mathcal{J}_\Omega(\theta)$ over $\Omega \subset \mathbb{R}^d$,
- $\mathcal{J}_{\text{boundary}}(\theta)$ over $\partial\Omega \subset \mathbb{R}^{d-1}$.

The classical approach involves quadrature methods. However, they require a grid, which is a problem in high dimension or on complex domains...

$\rightsquigarrow$ Use the **Monte-Carlo approach**, a mesh-less method whose convergence is **slow but independent of the dimension**.

PINNs: advantages and drawbacks

Once trained, PINNs with Monte-Carlo integration are able to

- **quickly** provide an approximation to the PDE solution,
- in a **mesh-less** fashion and on **complex domains**,
- **independently of the dimension**.

PINNs: advantages and drawbacks

Once trained, PINNs with Monte-Carlo integration are able to

- **quickly** provide an approximation to the PDE solution,
- in a **mesh-less** fashion and on **complex domains**,
- **independently of the dimension**.

However, PINNs

- have trouble **generalizing** to $x \notin \Omega$;
- are usually **not competitive with classical numerical methods for computational fluid dynamics**: to reach a given error (if possible), training takes longer than using a classical numerical method, and no convincing convergence results exist at the moment.

PINNs: advantages and drawbacks

Once trained, PINNs with Monte-Carlo integration are able to

- **quickly** provide an approximation to the PDE solution,
- in a **mesh-less** fashion and on **complex domains**,
- **independently of the dimension**.

However, PINNs

- have trouble **generalizing** to $x \notin \Omega$;
- are usually **not competitive with classical numerical methods for computational fluid dynamics**: to reach a given error (if possible), training takes longer than using a classical numerical method, and no convincing convergence results exist at the moment.

The most interesting use of PINNs, in our case, is to deal with **parametric PDEs**, where dimension-insensitivity is paramount.

Parametric PINNs: approximation using the PDE residual

The parametric PINN $W_\theta(x; \mu)$ with parameters $\mu \in \mathbb{M} \subset \mathbb{R}^m$ approximates the solution W to the parametric BVP:

$$\begin{cases} \mathcal{D}(W, x; \mu) = 0 & \text{for } x \in \Omega, \mu \in \mathbb{M}, \\ W(x) = g(x; \mu) & \text{for } x \in \partial\Omega, \mu \in \mathbb{M}. \end{cases} \quad \rightsquigarrow \quad \begin{cases} \mathcal{D}(W_\theta, x; \mu) \simeq 0 & \text{for } x \in \Omega, \mu \in \mathbb{M}, \\ W_\theta(x; \mu) \simeq g(x; \mu) & \text{for } x \in \partial\Omega, \mu \in \mathbb{M}. \end{cases}$$

Parametric PINNs: approximation using the PDE residual

The parametric PINN $W_\theta(x;\mu)$ with parameters $\mu \in \mathbb{M} \subset \mathbb{R}^m$ approximates the solution W to the parametric BVP:

$$\begin{cases} \mathcal{D}(W, x; \mu) = 0 & \text{for } x \in \Omega, \mu \in \mathbb{M}, \\ W(x) = g(x; \mu) & \text{for } x \in \partial\Omega, \mu \in \mathbb{M}. \end{cases} \quad \rightsquigarrow \quad \begin{cases} \mathcal{D}(W_\theta, x; \mu) \simeq 0 & \text{for } x \in \Omega, \mu \in \mathbb{M}, \\ W_\theta(x; \mu) \simeq g(x; \mu) & \text{for } x \in \partial\Omega, \mu \in \mathbb{M}. \end{cases}$$

The loss function then becomes

$$\mathcal{J}_{\mathrm{PDE}}(\theta) = \underbrace{\int_{\mathbb{M}} \int_{\Omega} \|\mathcal{D}(W_\theta, x; \mu)\|_2^2 \, dx d\mu}_{\mathcal{J}_\Omega(\theta)} + \underbrace{\int_{\mathbb{M}} \int_{\partial\Omega} \|W_\theta(x; \mu) - g(x; \mu)\|_2^2 \, dx d\mu}_{\mathcal{J}_{\mathrm{boundary}}(\theta)}.$$

Both integrals are, once again, approximated by the Monte-Carlo method.

Solving the nonlinear optimization problem

PINNs amount to solving a nonlinear optimization problem.

For such problems, state-of-the-art approaches rely on stochastic gradient descent[4], and so require differentiating the loss function with respect to θ.

Because of the Monte-Carlo estimation, the loss function contains terms in $\mathcal{D}(W_\theta, x_j; \boldsymbol{\mu}_i)$. Say $\mathcal{D}$ contains a Laplace operator: we need to compute, among other things,

$$\nabla_\theta \Delta W_\theta(x_j; \boldsymbol{\mu}_i).$$

These differentials are exactly computed, thanks to automatic differentiation tools.

Fortunately, these tools are already implemented in several libraries (we used pytorch).

[4] Namely, on the ADAM algorithm: see D. Kingma and J. Ba, (2015).

Implementation details

PINNs are implemented in scimba[5], developed in-house in the MACARON team.

The networks have 5 hidden layers of 20 neurons each, and $\sigma = \tanh$. In total, W_θ has 1761 parameters (one can compare this to a FEM with 1761 degrees of freedom). We train for 2 500 epochs (number of descent steps) and $N_c = 5\,000$ Monte-Carlo samples.

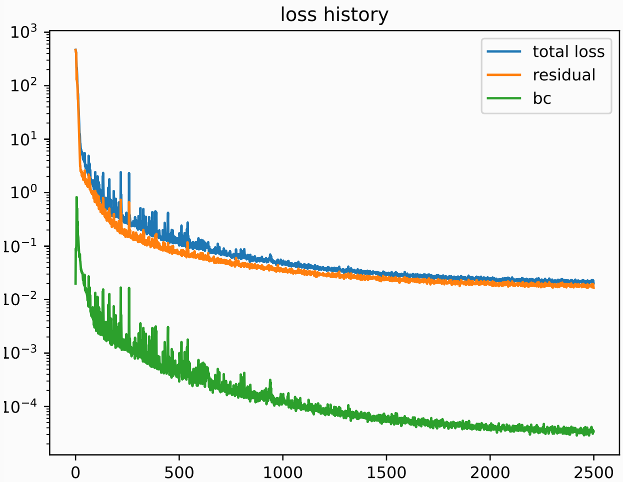
All computations are run on a single GPU, an AMD Instinct MI210.

We present PINN solutions, for several Ω and f, of a **four-dimensional parametric BVP**, whose solution depends on $x \in \Omega \subset \mathbb{R}^2$ and $\mu = (\alpha, \beta) \in \mathbb{M} \subset \mathbb{R}^2$:

$$\begin{cases} \Delta W(x;\mu) + \beta W(x;\mu) = f(x;\mu) & \text{for } (x,\mu) \in \Omega \times \mathbb{M}, \\ W(x;\mu) = 0 & \text{for } (x,\mu) \in \partial\Omega \times \mathbb{M}. \end{cases}$$

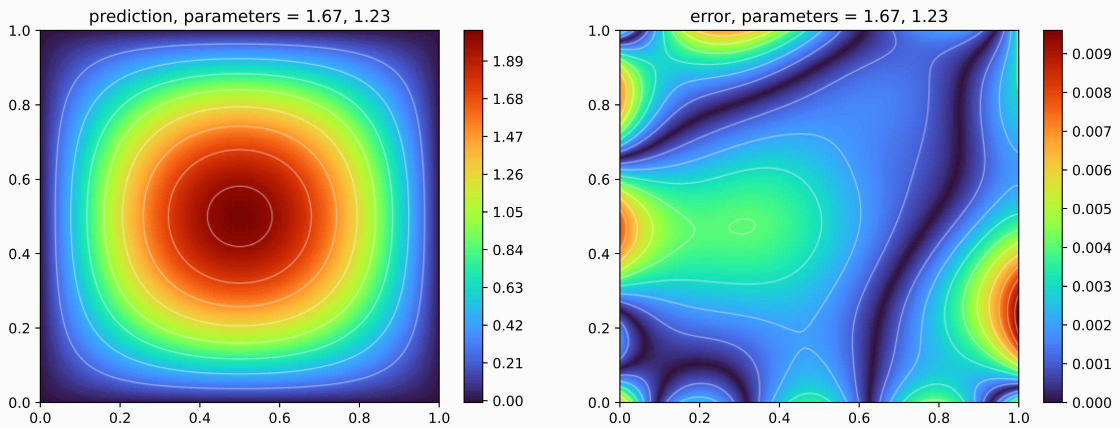
[5] freely accessible at `https://gitlab.inria.fr/scimba/scimba`

Square domain, with a boundary loss function

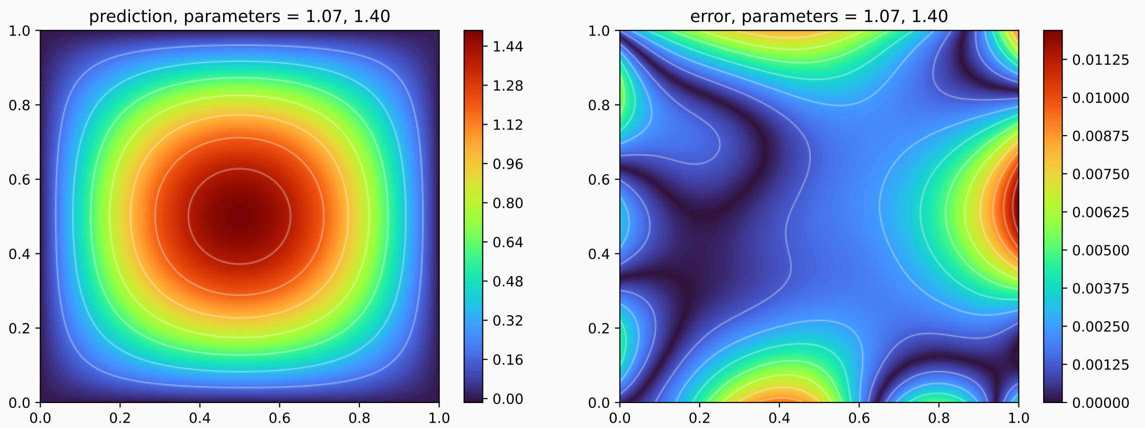


training time: $\sim$ 65 seconds

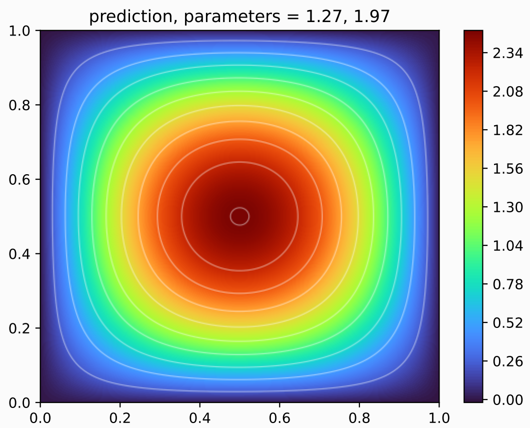
Square domain, with a boundary loss function

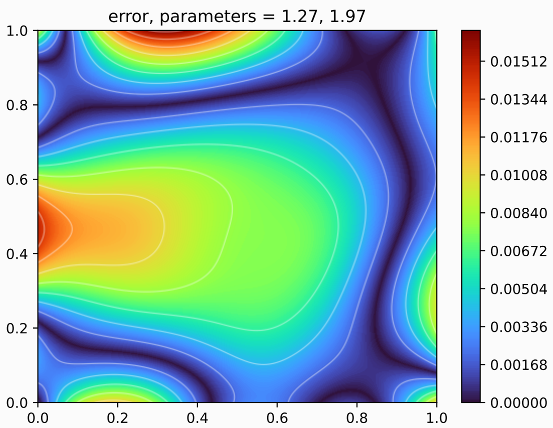


Square domain, with a boundary loss function

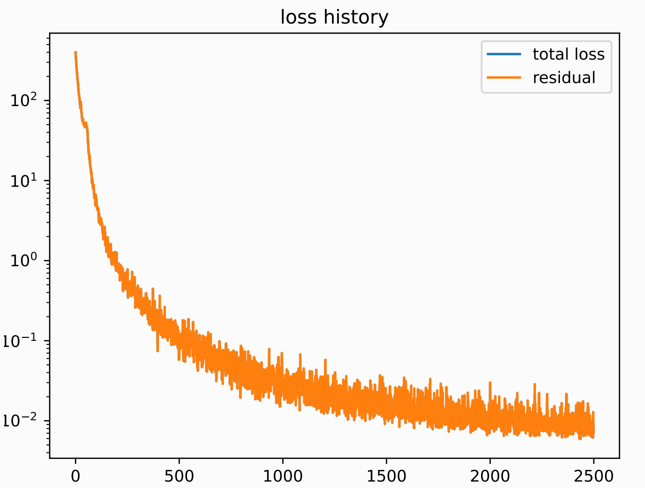


Square domain, with a boundary loss function



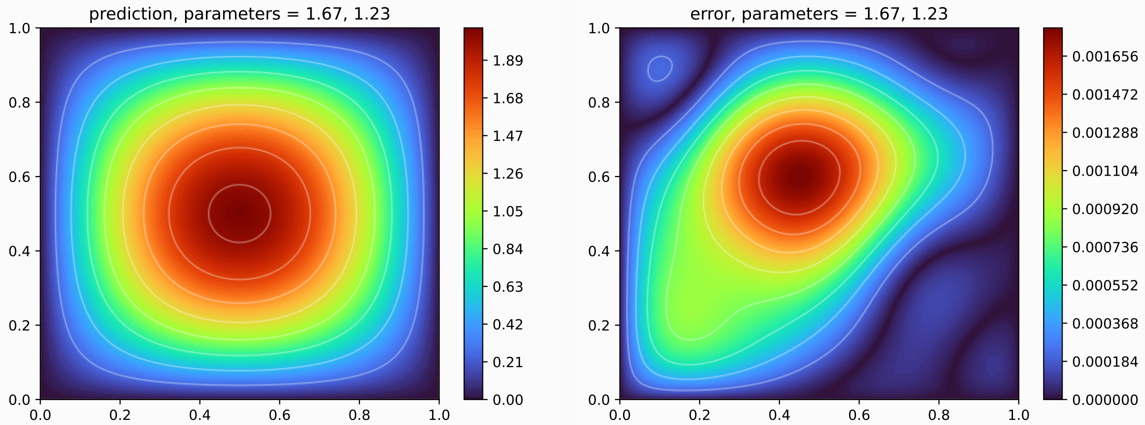


Square domain, hard-constrained boundary conditions

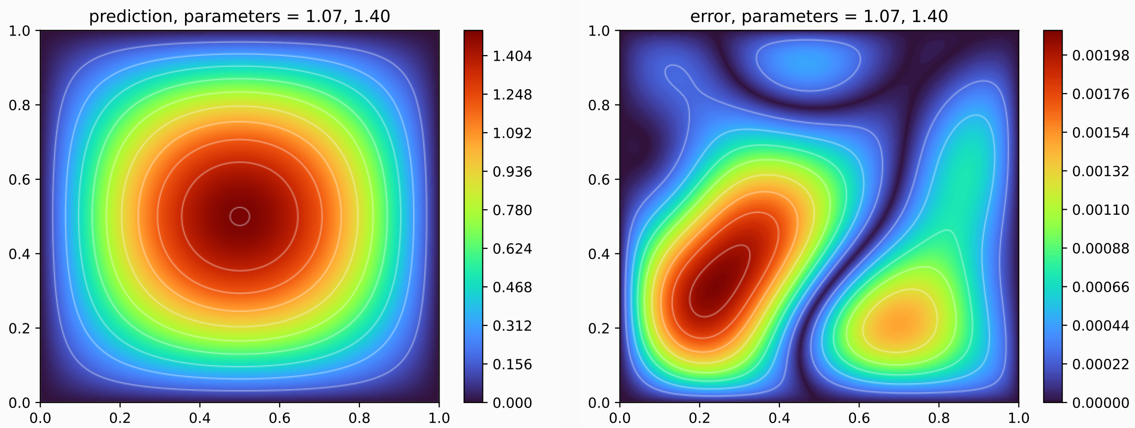


training time: ~ 50 seconds

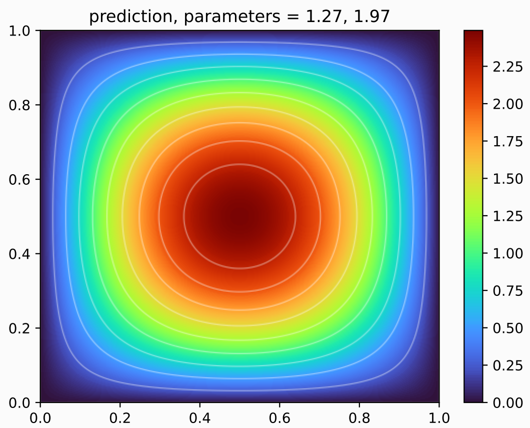
Square domain, hard-constrained boundary conditions

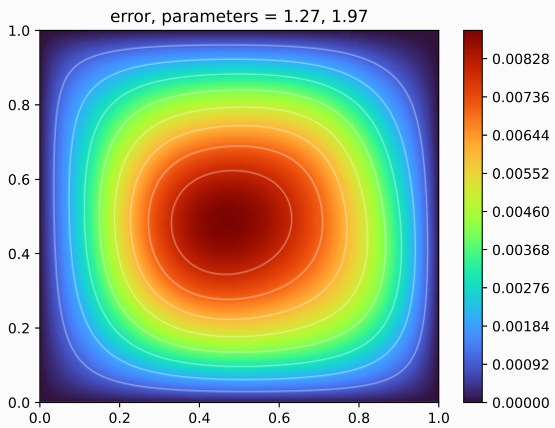


Square domain, hard-constrained boundary conditions

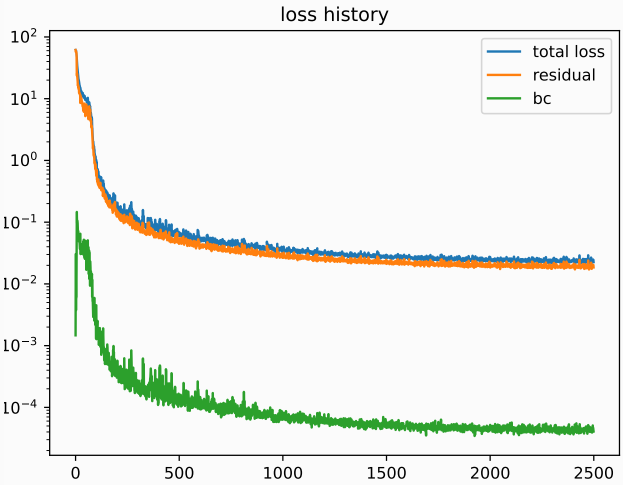


Square domain, hard-constrained boundary conditions



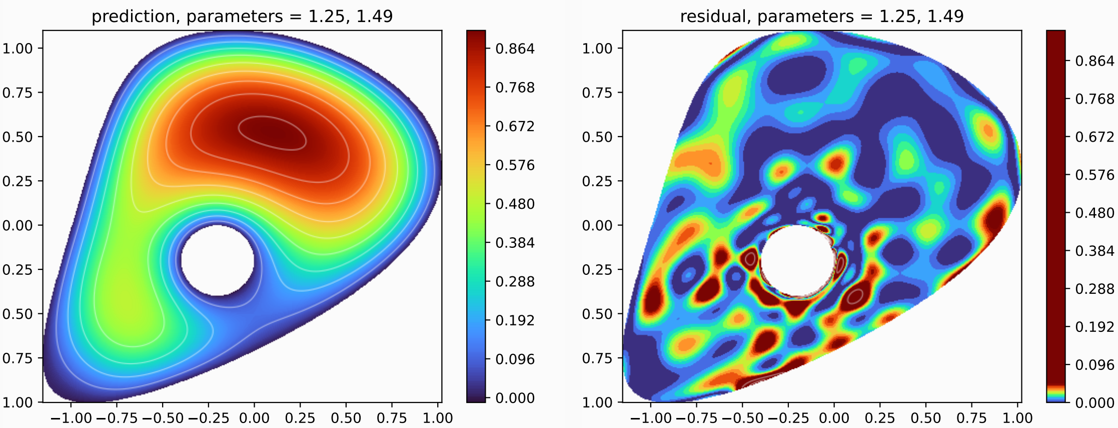


Potato-like domain with a hole, with a boundary loss function

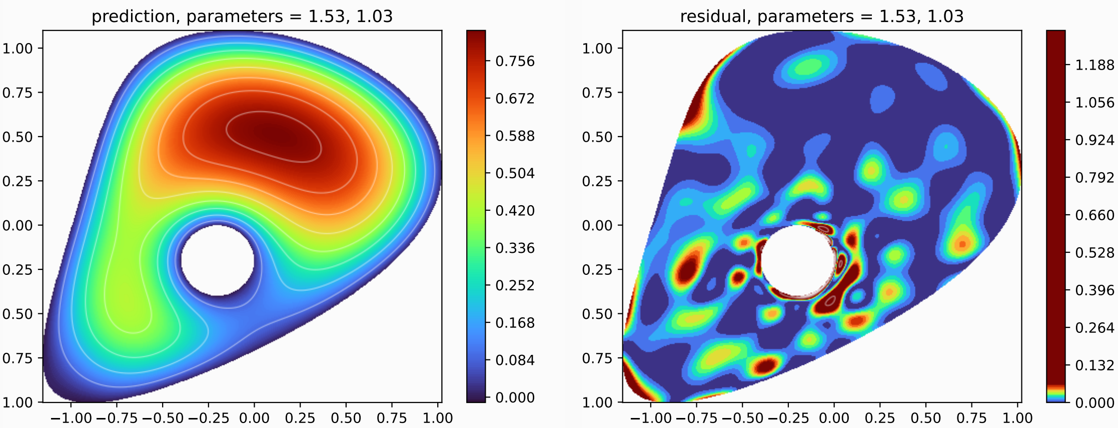


training time: $\sim$ 85 seconds

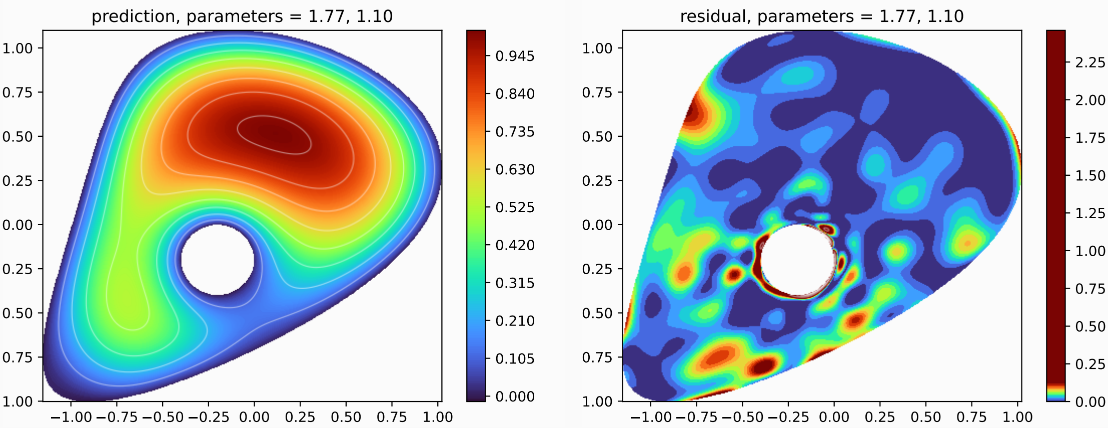
Potato-like domain with a hole, with a boundary loss function



Potato-like domain with a hole, with a boundary loss function



Potato-like domain with a hole, with a boundary loss function



What now?

The main objectives of this work are to improve:

- **the accuracy of parametric PINNs**, and
- **the error constant of classical methods**

by hybridizing PINNs with classical numerical methods.

What now?

The main objectives of this work are to improve:

- **the accuracy of parametric PINNs**, and
- **the error constant of classical methods**

by hybridizing PINNs with classical numerical methods.

More specifically, we enrich the polynomial bases of:

1. Continuous Galerkin (CG) methods for elliptic PDEs;
2. Discontinuous Galerkin (DG) methods for hyperbolic PDEs with source terms.

Context of hybrid methods

Advantages of the Continuous Galerkin (CG) method

- The CG method is provably **convergent**: more DOFs lead to a more accurate solution.
- **Optimized software** is widely used in industry and academia.

Context of hybrid methods

Advantages of the Continuous Galerkin (CG) method

- The CG method is provably **convergent**: more DOFs lead to a more accurate solution.
- **Optimized software** is widely used in industry and academia.

Advantages of PINNs

- PINNs are **mesh-less**, which is good for e.g. complex geometries.
- High-dimensional **parametric problems** are easily tackled.
- Once the network is trained, the solution **inference is quick**.

Context of hybrid methods

Advantages of the Continuous Galerkin (CG) method

- The CG method is provably **convergent**: more DOFs lead to a more accurate solution.
- **Optimized software** is widely used in industry and academia.

Advantages of PINNs

- PINNs are **mesh-less**, which is good for e.g. complex geometries.
- High-dimensional **parametric problems** are easily tackled.
- Once the network is trained, the solution **inference is quick**.

Hybrid methods seek to combine the best of both worlds: in our case, using a PINN to improve the resolution of the CG solution while retaining its order of accuracy.

Correcting the CG method with PINNs

We consider a parametric elliptic PDE $\mathcal{D}(u, x; \boldsymbol{\mu}) = 0$.

Correcting the CG method with PINNs

We consider a parametric elliptic PDE $\mathcal{D}(u, x; \boldsymbol{\mu}) = 0$.

We propose a two-step hybrid method:

1. **Offline phase**: train a neural network (e.g. a parametric PINN) to approximate a large family of solutions to the PDE;
2. **Online phase**: use the trained network to **correct the FEM approximation space**, and run the CG simulation on a coarse grid.

Classical CG method (finite element method)

The classical CG method relies on the following steps.

1. Rewrite the PDE $\mathcal{D}(u, x; \boldsymbol{\mu}) = 0$ as a variational problem:
$$\text{Find } u \in V \text{ such that } a(u, v) = \ell(v) \quad \forall v \in V,$$
where $V = \mathcal{H}_0^m(\Omega)$ is a Hilbert space, a a bilinear form, and ℓ a linear form.

2. Discretize the domain Ω and introduce V_h a finite-dimensional subspace of V, to get
$$\text{Find } u_h \in V_h \text{ such that } a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_h.$$

3. Solve the above linear system to get the approximation u_h of u.

Classical CG method (finite element method)

The classical CG method relies on the following steps.

1. Rewrite the PDE $\mathcal{D}(u, x; \boldsymbol{\mu}) = 0$ as a variational problem:
$$\text{Find } u \in V \text{ such that } a(u, v) = \ell(v) \quad \forall v \in V,$$
where $V = \mathcal{H}_0^m(\Omega)$ is a Hilbert space, a a bilinear form, and ℓ a linear form.

2. Discretize the domain Ω and introduce V_h a finite-dimensional subspace of V, to get
$$\text{Find } u_h \in V_h \text{ such that } a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_h.$$

3. Solve the above linear system to get the approximation u_h of u.

The approximation space V_h is made of piecewise polynomial functions on the mesh $\mathcal{T}_h$:
$$V_h = \left\{ v_h \in \mathcal{H}_0^m(\Omega) \cap C^0(\overline{\Omega}) \quad \text{such that} \quad \forall K \in \mathcal{T}_h,\ v_h\big|_K \in \mathbb{P}_q(K) \right\}.$$

Enriching the approximation space

Assume that we have a prior[6] $u_\theta \in \mathcal{H}_0^m(\Omega)$ on the solution u.

$\rightsquigarrow$ How to use u_θ to **improve the CG solution**?

[6] Here, given by a PINN, but that is not necessarily the case.

Enriching the approximation space

Assume that we have a prior[6] $u_\theta \in \mathcal{H}_0^m(\Omega)$ on the solution u.

$\rightsquigarrow$ How to use u_θ to **improve the CG solution**?

We suggest to modify the CG approximation space, replacing V_h by V_h^+, defined by:

$$V_h^+ = \{v_h = u_\theta + p_h^+, \quad p_h^+ \in V_h\}.$$

Since $u_\theta \in \mathcal{H}_0^m(\Omega)$, V_h^+ remains a subspace of $\mathcal{H}_0^m(\Omega)$, like V_h.

[6] Here, given by a PINN, but that is not necessarily the case.

Enriching the approximation space

Assume that we have a prior[6] $u_\theta \in \mathcal{H}_0^m(\Omega)$ on the solution u.

$\rightsquigarrow$ How to use u_θ to **improve the CG solution**?

We suggest to modify the CG approximation space, replacing V_h by V_h^+, defined by:

$$V_h^+ = \left\{ v_h = u_\theta + p_h^+, \quad p_h^+ \in V_h \right\}.$$

Since $u_\theta \in \mathcal{H}_0^m(\Omega)$, V_h^+ remains a subspace of $\mathcal{H}_0^m(\Omega)$, like V_h.

The discrete variational problem becomes[7]:

$$\begin{pmatrix} \text{Find } u_h \in V_h \text{ such that} \\ \forall v_h \in V_h,\ a(u_h, v_h) = \ell(v_h) \end{pmatrix} \rightsquigarrow \begin{pmatrix} \text{Find } u_h^+ \in V_h^+ \text{ such that} \\ \forall v_h \in V_h,\ a(u_h, v_h) = \ell(v_h) \end{pmatrix}$$

[6] Here, given by a PINN, but that is not necessarily the case.
[7] This sets the method in the Petrov-Galerkin framework, where trial and test spaces are different.

Enriching the approximation space

Assume that we have a prior[6] $u_\theta \in \mathcal{H}_0^m(\Omega)$ on the solution u.

$\rightsquigarrow$ How to use u_θ to **improve the CG solution**?

We suggest to modify the CG approximation space, replacing V_h by V_h^+, defined by:

$$V_h^+ = \left\{ v_h = u_\theta + p_h^+, \quad p_h^+ \in V_h \right\}.$$

Since $u_\theta \in \mathcal{H}_0^m(\Omega)$, V_h^+ remains a subspace of $\mathcal{H}_0^m(\Omega)$, like V_h.

The discrete variational problem becomes[7]:

$$\left(\begin{array}{c} \text{Find } u_h^+ \in V_h^+ \text{ such that} \\ \forall v_h \in V_h,\ a(u_h^+, v_h) = \ell(v_h) \end{array} \right) \iff \left(\begin{array}{c} \text{Find } p_h^+ \in V_h \text{ such that} \\ \forall v_h \in V_h,\ a(p_h^+, v_h) = \ell(v_h) - a(u_\theta, v_h). \end{array} \right)$$

[6] Here, given by a PINN, but that is not necessarily the case.
[7] This sets the method in the Petrov-Galerkin framework, where trial and test spaces are different.

Error analysis (proof in appendix)

Equipped with the modified approximation space, we now perform an error analysis.

Theorem: Let $u \in \mathcal{H}_0^m$ be the exact solution of the BVP, $u_\theta \in \mathcal{H}_0^m(\Omega)$ a prior on u, and $u_h^+ \in V_h^+$ the enriched CG solution (considering $\mathbb{P}_q$ polynomials, with $m \leqslant q$). Then:

$$\|u - u_h^+\|_{H^m} \lesssim C_{\mathrm{gain}}^+ \underbrace{h^{q+1-m}|u|_{H^{q+1}}}_{\text{classical CG error}}.$$

Error analysis (proof in appendix)

Equipped with the modified approximation space, we now perform an error analysis.

Theorem: Let $u \in \mathcal{H}_0^m$ be the exact solution of the BVP, $u_\theta \in \mathcal{H}_0^m(\Omega)$ a prior on u, and $u_h^+ \in V_h^+$ the enriched CG solution (considering $\mathbb{P}_q$ polynomials, with $m \leqslant q$). Then:

$$\|u - u_h^+\|_{H^m} \lesssim C_{\mathrm{gain}}^+ \underbrace{h^{q+1-m}|u|_{H^{q+1}}}_{\text{classical CG error}}.$$

In this result, the constant

$$C_{\mathrm{gain}}^+ = \frac{|u - u_\theta|_{H^{q+1}}}{|u|_{H^{q+1}}}$$

represents the potential gain compared to the error of the classical CG method.

Key remark: The prior u_θ must be a good approximation of the $(q+1)^{\text{th}}$ derivative of u.
This is why we use PINNs, rather than purely data-driven priors!

Summary

This hybrid method can be seen as

- enriching the CG approximation space with a PINN prior, to get V_h^+;
- or ensuring the convergence of a PINN approximation by adding a CG approximation on a coarse grid.

Remark: The hybrid method consists in offline and online parts:

Offline: Train the PINN on the parametric PDE (potentially time-consuming).

Online: There are two online substeps:

1. evaluate the NN at Gauss points to compute the approximation space,
2. use a regular, coarse CG solver with the new approximation space.

NN inference is quick, so the online cost of using the NN is negligible!

2D+2D Poisson problem

First, we tackle the following 4D PDE (2D in space, with two parameters):

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

with $\Omega = (-\frac{\pi}{2}, \frac{\pi}{2})^2$, parameters $x_1^0, x_2^0 \sim \mathcal{U}(-0.5, 0.5)$ and exact solution

$$u(x_1, x_2; x_1^0, x_2^0) = \sin(2x_1)\sin(2x_2)\exp\left(-\frac{1}{2}\left((x_1 - x_1^0)^2 + (x_2 - x_2^0)^2\right)\right).$$

2D+2D Poisson problem

First, we tackle the following 4D PDE (2D in space, with two parameters):

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

with $\Omega = (-\frac{\pi}{2}, \frac{\pi}{2})^2$, parameters $x_1^0, x_2^0 \sim \mathcal{U}(-0.5, 0.5)$ and exact solution

$$u(x_1, x_2; x_1^0, x_2^0) = \sin(2x_1)\sin(2x_2)\exp\left(-\frac{1}{2}\left((x_1 - x_1^0)^2 + (x_2 - x_2^0)^2\right)\right).$$

With a given q, we compare, averaging over 50 values of the parameters (x_1^0, x_2^0), the relative L^2 errors of the enhanced $\mathbb{P}_q$ CG method (with approximation space V_h^+) to

- the classical $\mathbb{P}_q$ CG method (with approximation space V_h);
- the results of the PINN.

2D+2D Poisson problem – gains

We add a component to the loss function: the derivatives with respect to the parameters

$$\|\partial_{x_1^0}(\Delta u_\theta + f)\| + \|\partial_{x_2^0}(\Delta u_\theta + f)\|.$$

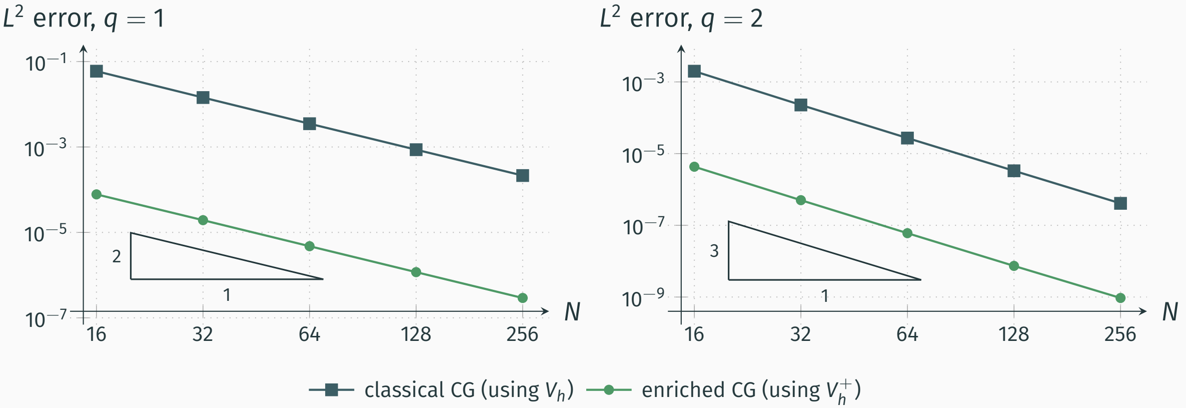
2D+2D Poisson problem – gains

We add a component to the loss function: the derivatives with respect to the parameters

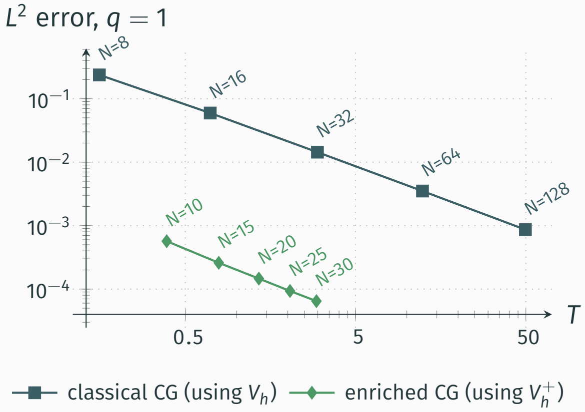
$$\|\partial_{x_1^0}(\Delta u_\theta + f)\| + \|\partial_{x_2^0}(\Delta u_\theta + f)\|.$$

		Gains (L^2 error): ours w.r.t. PINNs			Gains (L^2 error): ours w.r.t. CG		
q	**N**	**min**	**max**	**mean**	**min**	**max**	**mean**
1	20	18.28	66.19	43.42	243.79	874.3	633.45
1	40	73.45	272.36	176.52	241.8	843.29	621.68
2	20	362.57	2,052.78	1,025.28	177.74	476.76	376.16
2	40	3,081.22	17,532.62	8,725.57	177.16	472.55	371.93
3	20	4,879.13	32,757.68	14,646.89	116.52	298.33	208.35
3	40	88,736.63	587,716.86	264,383.45	117.46	296.34	208.29

2D+2D Poisson problem – convergence



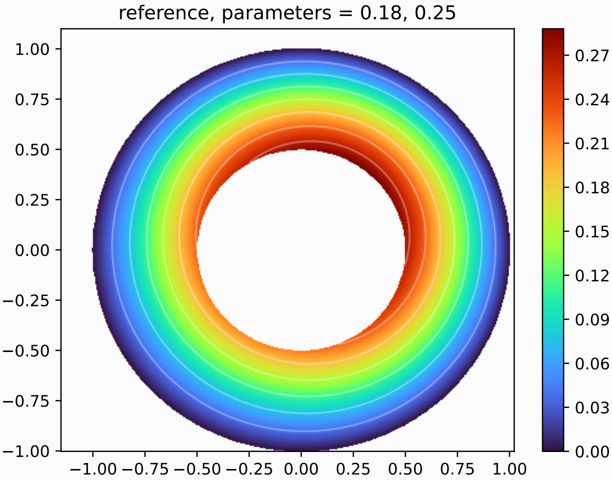
We now compare computation times: we record the mesh size and the computation time T (*excluding training time*! see appendix) required to reach an error E.



E		10^{-3}	10^{-4}
	V_h	120	373
N	V_h^+	8	25
	gain	15	14.9
	V_h	43	424
T	V_h^+	0.24	1.93
	gain	179	220

2D+2D Poisson problem on a donut

We now consider the Poisson problem on a donut, with Dirichlet boundary conditions.

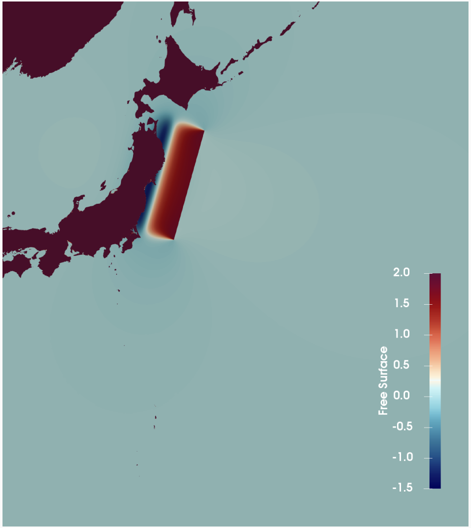


2D+2D Poisson problem on a donut – gains

		Gains (L^2 error): ours w.r.t. PINNs			Gains (L^2 error): ours w.r.t. CG		
q	N	min	max	mean	min	max	mean
1	20	10.18	35.8	19.49	71.17	254.32	153.44
1	40	33.35	125.03	65.64	63.93	199.95	131.06
2	20	189.1	1,331.27	485.95	32.47	80.69	58.98
2	40	1,241.42	9,686.46	3,261.71	30.57	74.15	54.09
3	20	5,630.17	39,651.58	14,987.25	15.73	32.1	23.07
3	40	74,794.74	573,631.63	202,631.9	13.67	29.52	20.57

Tsunami simulation: naive numerical method

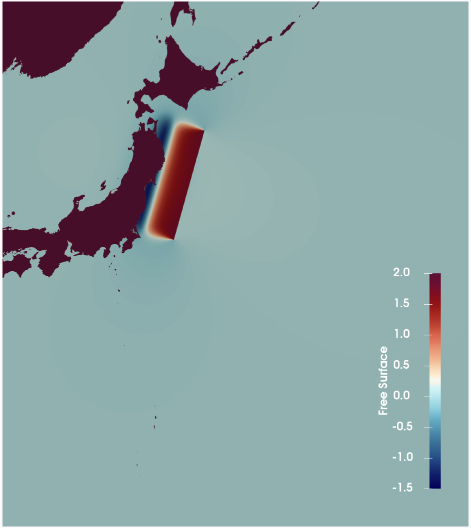
Tsunami initialization



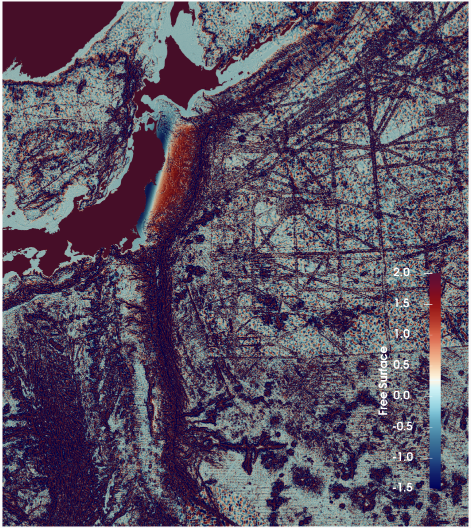
Simulation with a naive numerical method

Tsunami simulation: naive numerical method

Tsunami initialization



Simulation with a naive numerical method



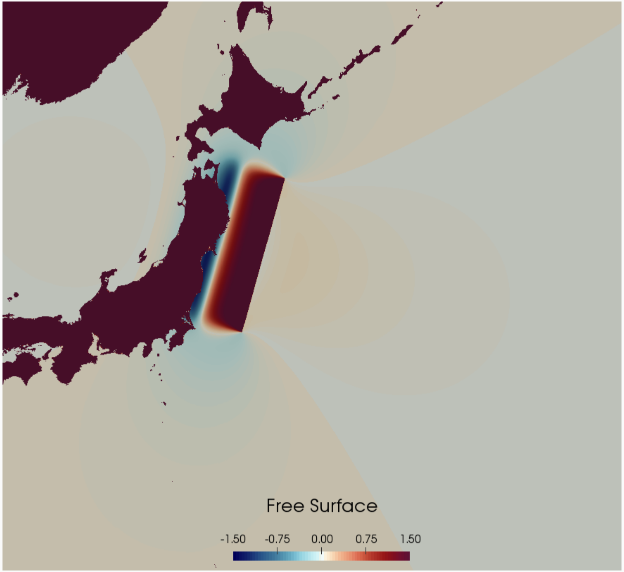
Tsunami simulation: failure

$\rightsquigarrow$ **The simulation is not usable!**

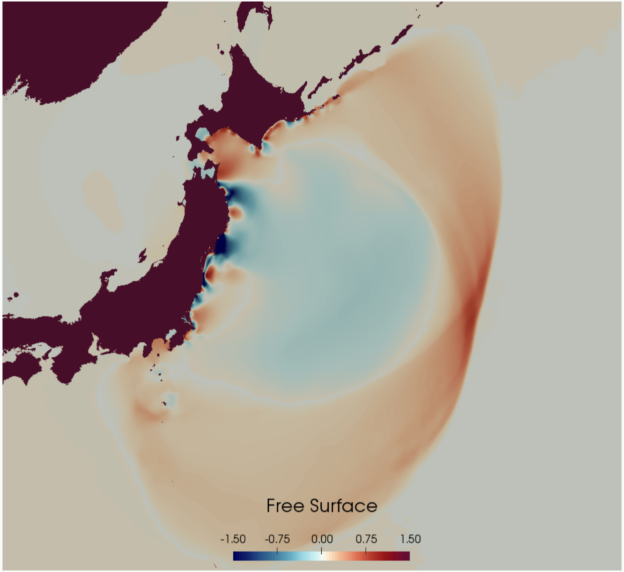
Indeed, the ocean at rest, far from the tsunami, started spontaneously producing waves.

This comes from the non-preservation of stationary solutions, hence the need to develop numerical methods that **preserve stationary solutions**: so-called **well-balanced** methods.

Tsunami simulation: well-balanced method



Tsunami simulation: well-balanced method



Objectives

The goal of this work is to provide a numerical method which:

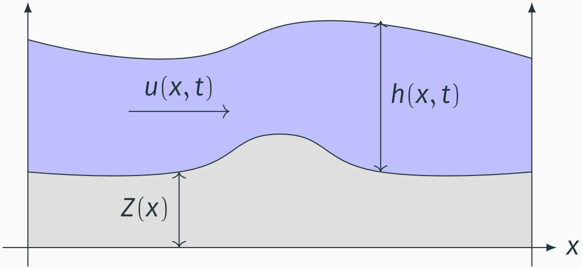
- is able to deal with generic systems of balance laws,
- can provide a very good approximation of families of steady solutions,
- is as accurate as classical methods on unsteady solutions,
- with provable convergence estimates.

To that end, we select the **Discontinuous Galerkin (DG)** framework.

The shallow water equations

The shallow water equations are governed by the following PDE:

$$\begin{cases} \partial_t h + \partial_x q = 0, \\ \partial_t q + \partial_x \left(\dfrac{q^2}{h} + \dfrac{1}{2} g h^2 \right) = -g h \partial_x Z(x). \end{cases}$$

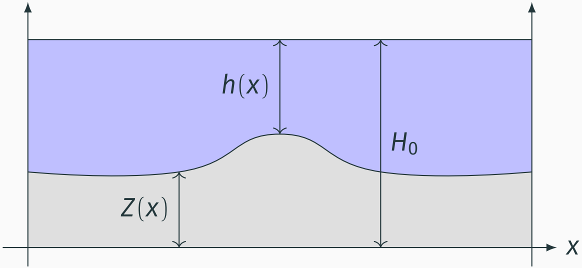


- $h(x,t)$: water depth
- $u(x,t)$: water velocity
- $q = hu$: water discharge
- $Z(x)$: known topography
- g: gravity constant

The shallow water equations: steady solutions

The steady solutions of the shallow water equations are governed by the following ODEs:

$$\begin{cases} \partial_x q = 0, \\ \partial_x \left(\dfrac{q^2}{h} + \dfrac{1}{2} g h^2 \right) = -g h \partial_x Z(x), \end{cases} \quad \rightsquigarrow \quad \begin{cases} q = \mathrm{cst} =: q_0, \\ \dfrac{q_0^2}{2h^2} + g(h + Z) = \mathrm{cst}. \end{cases}$$



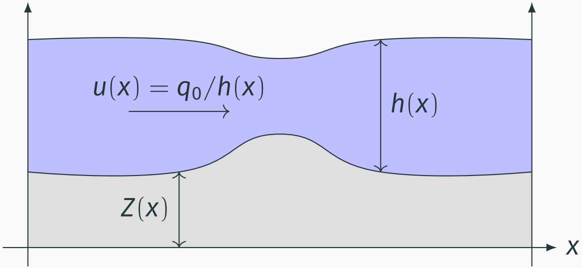
If the velocity vanishes, i.e. $q_0 = 0$, we obtain the lake at rest steady solution:

$$h + Z = \mathrm{cst} =: H_0.$$

The shallow water equations: steady solutions

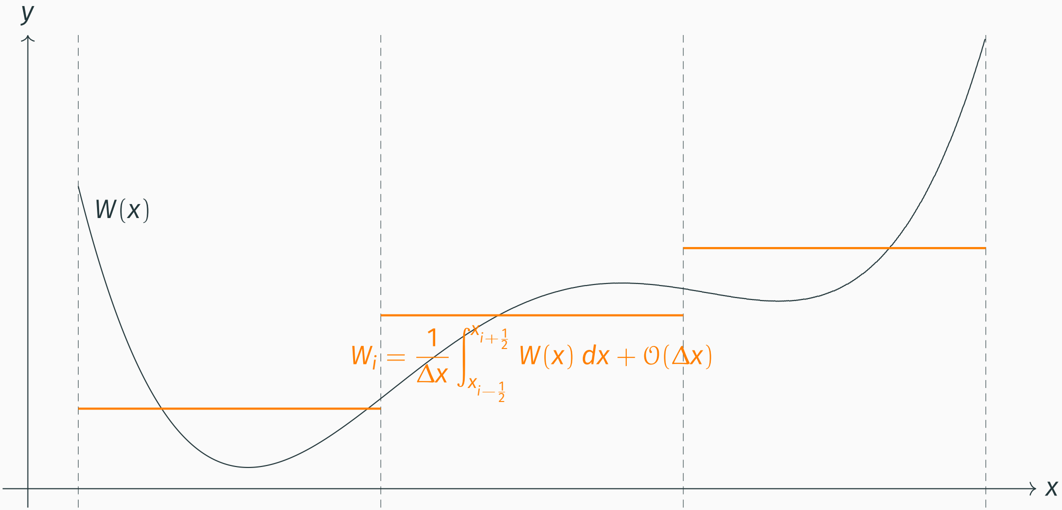
The steady solutions of the shallow water equations are governed by the following ODEs:

$$\begin{cases} \partial_x q = 0, \\ \partial_x \left(\dfrac{q^2}{h} + \dfrac{1}{2} g h^2 \right) = -gh\partial_x Z(x), \end{cases} \quad \rightsquigarrow \quad \begin{cases} q = \text{cst} =: q_0, \\ \dfrac{q_0^2}{2h^2} + g(h+Z) = \text{cst}. \end{cases}$$

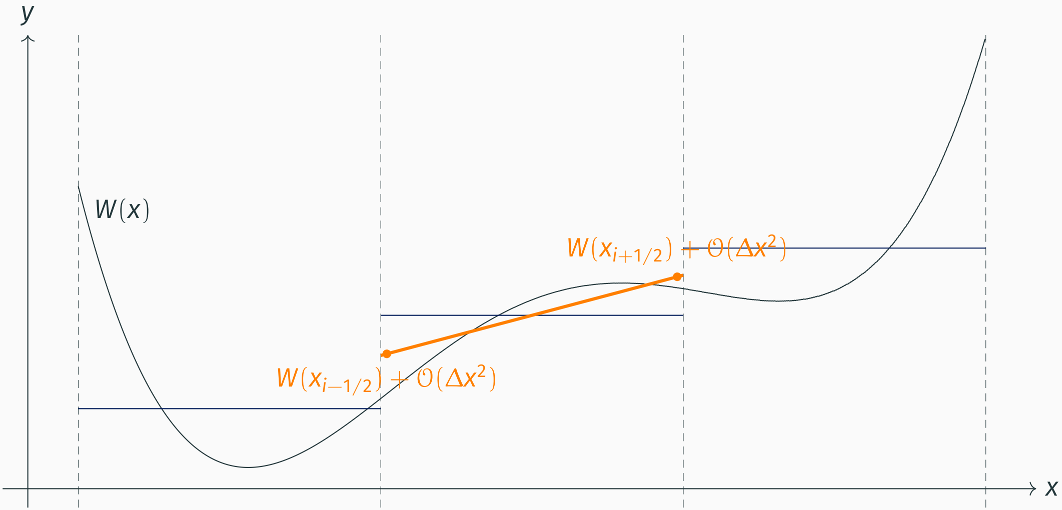


For a nonzero discharge $q_0 \neq 0$, we obtain a moving steady solution: $h(x)$ satisfies a polynomial equation of degree 3 for all x.

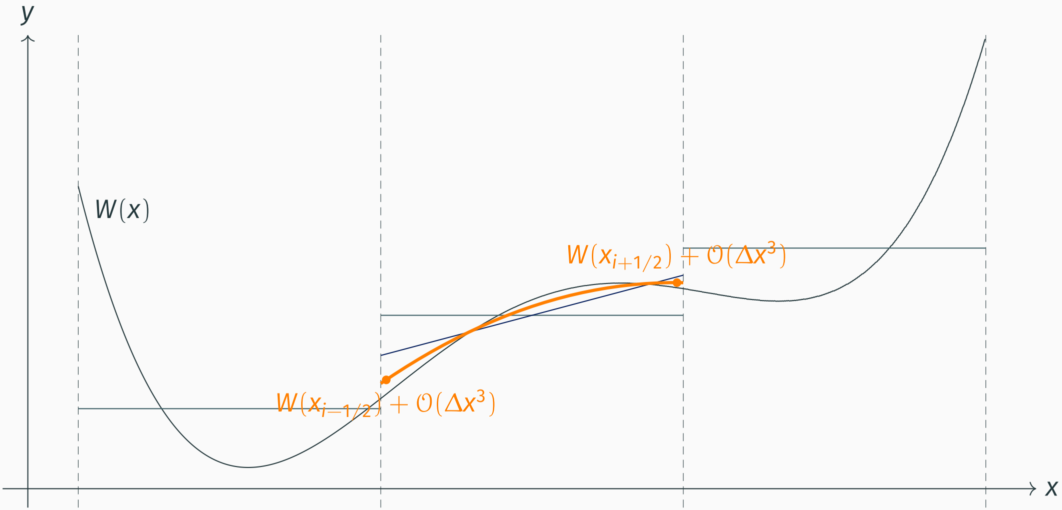
Finite volume method, visualized



Discontinuous Galerkin, visualized



Discontinuous Galerkin, visualized



Discontinuous Galerkin: an example

On the previous slide, the data W is represented by

- a polynomial of degree 2 in each cell (Galerkin approximation),
- which is Discontinuous at interfaces between cells.

Discontinuous Galerkin: an example

On the previous slide, the data W is represented by

- a polynomial of degree 2 in each cell (Galerkin approximation),
- which is Discontinuous at interfaces between cells.

Therefore, in each cell Ω_i, W is approximated by

$$W\big|_{\Omega_i} \simeq W_i^{\mathrm{DG}} := \alpha_0 + \alpha_1 x + \alpha_2 x^2 = \sum_{j=0}^{2} \alpha_j x^j,$$

where the polynomial coefficients α_0, α_1 and α_2 are determined to ensure fitness between the continuous data and its polynomial approximation.

Any polynomial of degree two can be exactly represented this way.

Discontinuous Galerkin: polynomial basis

More generally, we define a polynomial basis $\varphi_0, \dots, \varphi_N$ on each cell Ω_i and approximate the solution in this basis.

A usual example is the following so-called **modal basis**:

$$\forall j \in \{0, \dots, N\}, \quad \varphi_j(x) = x^j.$$

Discontinuous Galerkin: polynomial basis

More generally, we define a polynomial basis $\varphi_0, \ldots, \varphi_N$ on each cell Ω_i and approximate the solution in this basis.

A usual example is the following so-called **modal basis**:

$$\forall j \in \{0, \ldots, N\}, \quad \varphi_j(x) = x^j.$$

Main takeaway: The DG scheme is exact on every function that can be exactly represented in the basis!

Main idea

Recall that the DG scheme will be exact on every function that can be exactly represented in the DG basis, as soon as it is also a solution to the PDE.

Main idea

Recall that the DG scheme will be exact on every function that can be exactly represented in the DG basis, as soon as it is also a solution to the PDE.

Main idea

Enhance the DG basis by using the steady solution!

$\rightsquigarrow$ If the **steady solution or an approximation thereof is contained in the basis**, then:

- using the exact steady solution in the basis will make the scheme exactly well-balanced;
- using an approximation of the steady solution will make the scheme approximately well-balanced.

Enhanced DG bases

Assume that you know a **prior** W_θ on the steady solution.

It can be the exact steady solution ($W_\theta = W_{eq}$), or it can be an approximation ($W_\theta \simeq W_{eq}$).

The goal is now to **enhance the modal basis** V using W_θ:

$$V = \{1, x, x^2, \dots, x^N\}.$$

Enhanced DG bases

Assume that you know a **prior** W_θ on the steady solution.

It can be the exact steady solution ($W_\theta = W_{\text{eq}}$), or it can be an approximation ($W_\theta \simeq W_{\text{eq}}$).

The goal is now to **enhance the modal basis** V using W_θ:

$$V = \{1, x, x^2, \dots, x^N\}.$$

First possibility: multiply the whole basis by W_θ

$$V_*^\theta = \{W_\theta, x\, W_\theta, x^2\, W_\theta, \dots, x^N\, W_\theta\}.$$

Enhanced DG bases

Assume that you know a **prior** W_θ on the steady solution.

It can be the exact steady solution ($W_\theta = W_{\text{eq}}$), or it can be an approximation ($W_\theta \simeq W_{\text{eq}}$).

The goal is now to **enhance the modal basis** V using W_θ:

$$V = \{1, x, x^2, \dots, x^N\}.$$

First possibility: multiply the whole basis by W_θ

$$V_*^\theta = \{W_\theta, x\, W_\theta, x^2\, W_\theta, \dots, x^N\, W_\theta\}.$$

Second possibility: replace the first element with W_θ

$$V_+^\theta = \{W_\theta, x, x^2, \dots, x^N\}.$$

Error estimates

We denote by:

- W_{ex} the exact solution,
- W_{DG} the approximate solution without prior,
- W_{DG}^{θ} the approximate solution with prior W_{θ} and basis V_{*}^{θ}.

For a DG scheme of order $q+1$, we obtain[8] the following error estimates:

$$\|W_{\text{ex}} - W_{\text{DG}}\| \lesssim \left|W_{\text{ex}}\right|_{H^{q+1}} \Delta x^{q+1},$$

$$\|W_{\text{ex}} - W_{\text{DG}}^{\theta}\| \lesssim \left|\frac{W_{\text{ex}}}{W_{\theta}}\right|_{H^{q+1}} \Delta x^{q+1} \|W_{\theta}\|_{L^{\infty}}.$$

Conclusion of the error estimates: the prior W_{θ} needs to provide a **good approximation of the derivatives** of the steady solution.

[8] Rigorous error estimates are written in terms of the error in the projection onto both bases.

Obtaining a prior

For very simple systems, one can use the exact steady solution as a prior.

However, in many cases, even for some simple and well-known systems, one cannot compute the exact steady solution. Therefore, **an approximation is required**.

How to obtain such an approximation?

Obtaining a prior

For very simple systems, one can use the exact steady solution as a prior.

However, in many cases, even for some simple and well-known systems, one cannot compute the exact steady solution. Therefore, **an approximation is required**.

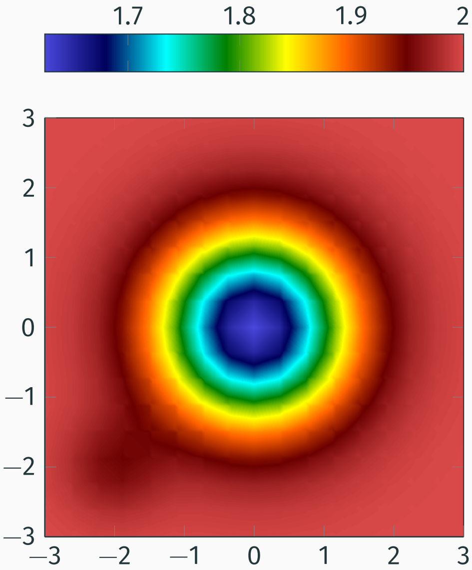
How to obtain such an approximation?

1. **First possibility**: use a traditional numerical approximation, obtained by classical ODE solvers (e.g. Runge-Kutta schemes).
2. **Second possibility**: use a Physics-Informed Neural Network (PINN).

Since we need a good approximation of the derivatives, we use a PINN.

Next step: Validate the method with several numerical experiments.

Perturbation of a shallow water steady solution



PINN trained on a parametric steady solution, driven by the topography

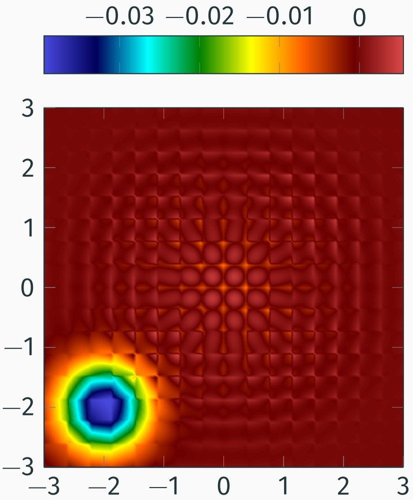
$$Z(x;\mu) = \Gamma \exp\big(\alpha(r_0^2 - \|x\|^2)\big),$$

with physical parameters

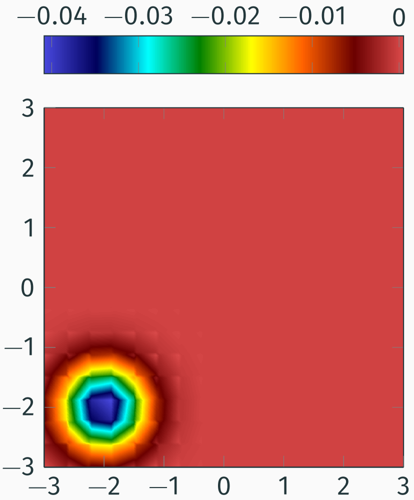
$$\mu \in \mathbb{P} \iff \begin{cases} \alpha \in [0.25, 0.75], \\ \Gamma \in [0.1, 0.4], \\ r_0 \in [0.5, 1.25]. \end{cases}$$

Left plot: initial condition, made of a perturbed steady solution.

Perturbation of a shallow water steady solution

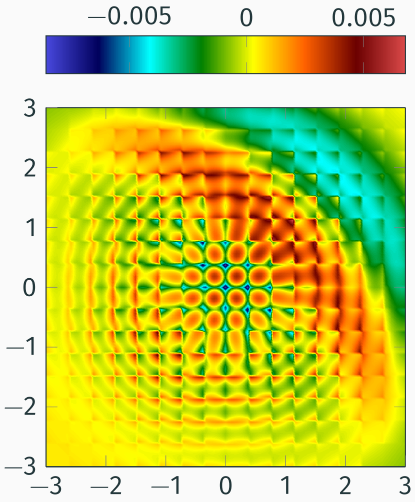


(a) classical basis

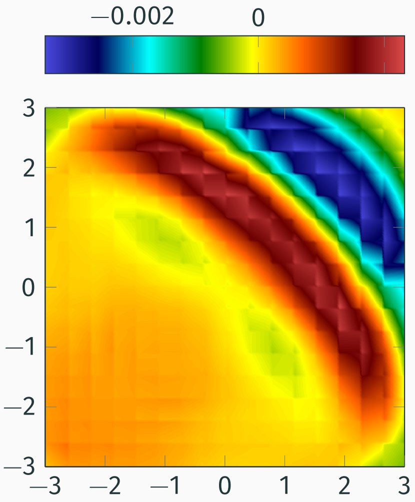


(b) enhanced basis

Perturbation of a shallow water steady solution



(a) classical basis



(b) enhanced basis

Conclusion

We introduced

- a framework for approximating solutions to PDEs with linear or nonlinear functions,
- physics-informed neural networks (PINNs),
- a hybrid method between FEM and PINNs, applied to elliptic problems,
- a hybrid method blending physics-informed learning and DG bases.

Perspectives include

- tackling time-dependent solutions,
- going to complex three-dimensional geometries and richer PDEs.

Related paper: E. Franck, V. Michel-Dansac and L. Navoret.
"Approximately WB DG methods using bases enriched with PINNs.", J. Comput. Phys., 2024
git repository: https://github.com/Victor-MichelDansac/DG-PINNs

Thank you for your attention!

Exact imposition of the boundary conditions

For the moment, the **boundary conditions are viewed as constraints**, and the solution will not exactly satisfy them.

This can be remedied by introducing a **suitable ansatz**[9]. To that end, we define

$$\widetilde{W}_\theta = \mathcal{B}(W_\theta, x, t; \mu), \quad \text{such that} \quad \widetilde{W}_\theta(x, t; \mu) = g(x, t; \mu) \quad \text{for } x \in \partial\Omega.$$

Clearly, the new approximate solution $\widetilde{W}_\theta$ exactly satifies the boundary conditions.

Moreover, the boundary loss function can be eliminated, thus **reducing competition** between the loss functions.

$\rightsquigarrow$ How to get such an ansatz? We check on an example.

[9] I. E. Lagaris et al., *IEEE Trans. Neural Netw.* (1998)

Exact imposition of the boundary conditions: example

Let us go back to the parameterized Laplace equation, where $\mu = (\alpha, \beta)$:

$$\begin{cases} \Delta W(x;\mu) + \beta W(x;\mu) = f(x;\mu) & \text{for } (x,\mu) \in \Omega \times \mathbb{P}, \\ W(x;\mu) = 0 & \text{for } (x,\mu) \in \partial\Omega \times \mathbb{P}. \end{cases}$$

Homogeneous Dirichlet BC are imposed on $\Omega = (0,1)^2$, and so we define the ansatz

$$\widetilde{W_\theta} = \mathcal{B}(W_\theta, x; \mu) = x_1(1-x_1)\, x_2(1-x_2)\, W_\theta.$$

This obviously satisfies the boundary conditions, since $\forall x \in \partial\Omega, \widetilde{W_\theta}(x;\mu) = 0$.

Therefore, the loss function only has to ensure that $\widetilde{W_\theta}$ approximates the solution to the PDE in the interior of Ω, through minimizing the loss function

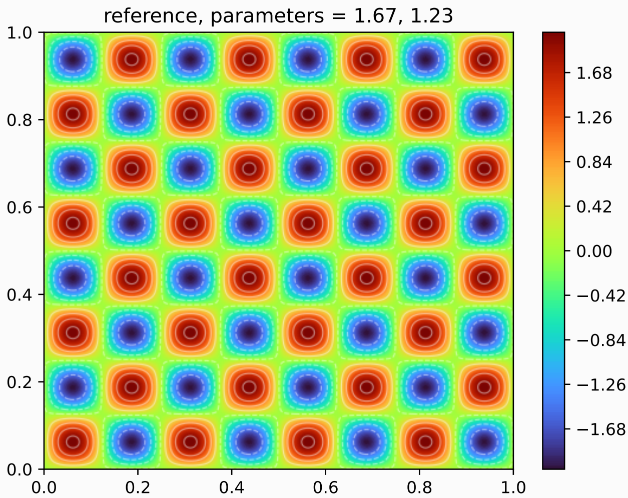
$$\mathcal{J}_{\mathrm{PDE}}(\theta) = \int_{\mathbb{P}} \int_{\Omega} \left\| \Delta \widetilde{W_\theta}(x;\mu) + \beta \widetilde{W_\theta}(x;\mu) - f(x;\mu) \right\|_2^2 dx\, d\mu.$$

High-frequency problem: spectral bias of MLPs

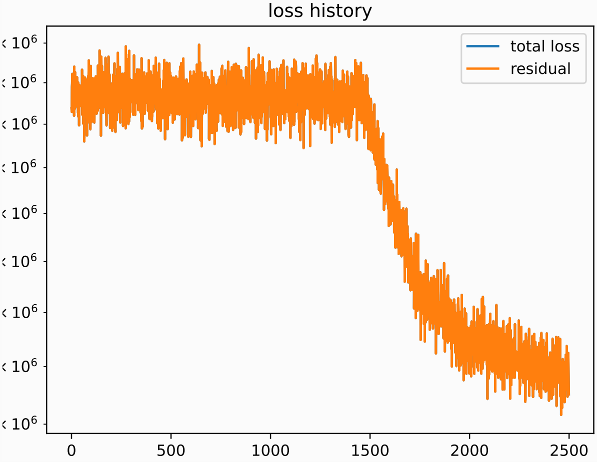
Spectral bias: MLPs first learn the low frequencies, before learning the high ones (with difficulty).

To illustrate this, we consider the high-frequency solution

$$W_{\text{exact}}(x; \boldsymbol{\mu}) = \alpha\beta \sin(8\pi x_1) \sin(8\pi x_2).$$

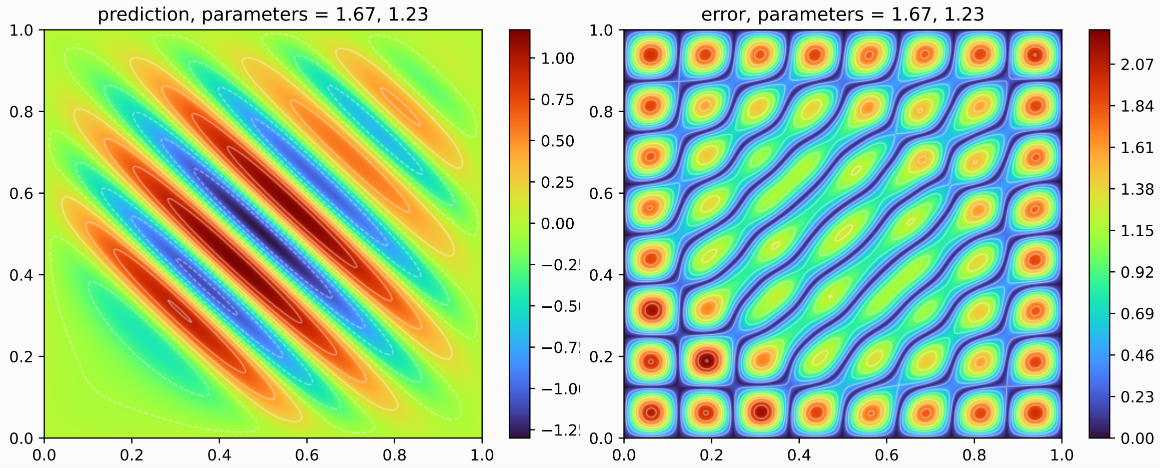


High-frequency problem: spectral bias of MLPs

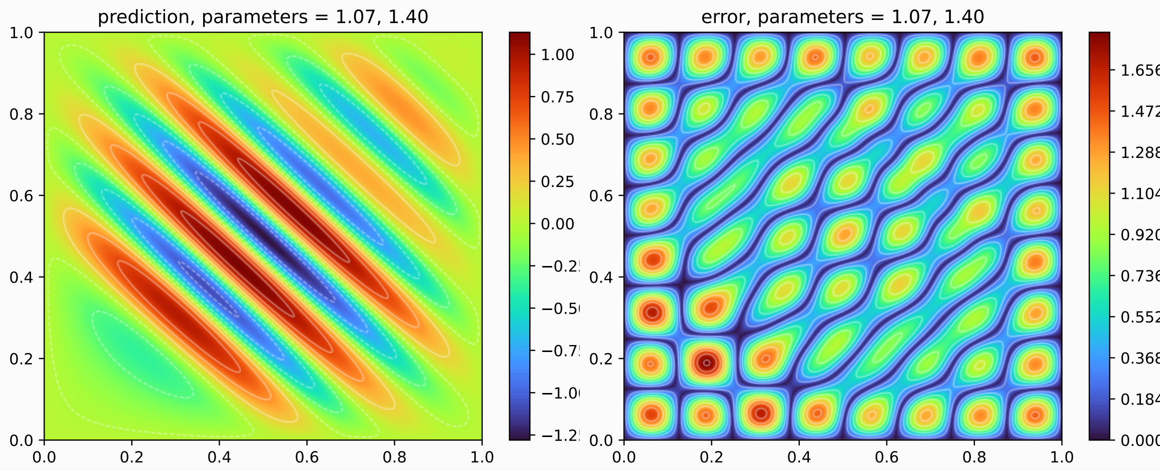


elapsed time: ~ 50 seconds

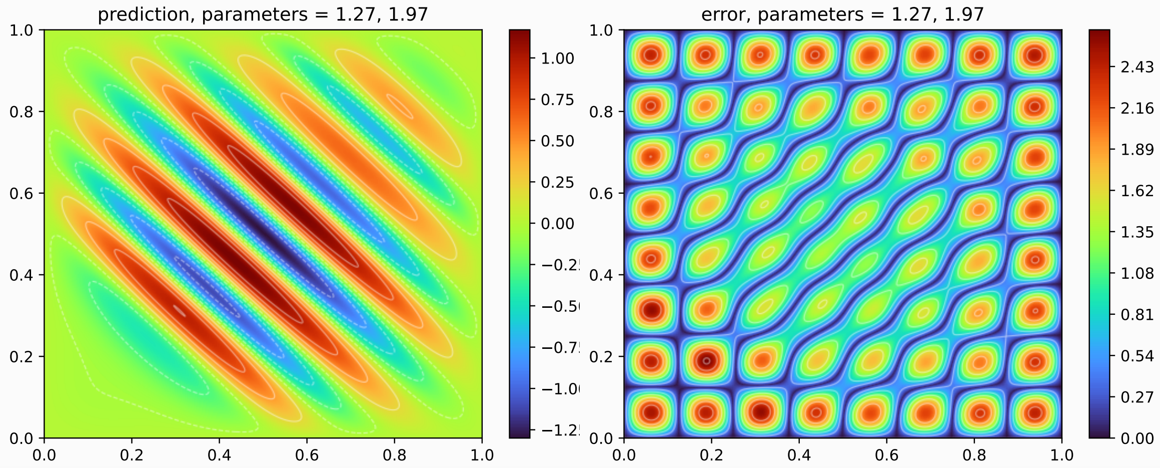
High-frequency problem: spectral bias of MLPs



High-frequency problem: spectral bias of MLPs



High-frequency problem: spectral bias of MLPs



High-frequency problem: with Fourier features

To overcome the spectral bias of MLPs, we can use Fourier features[10].

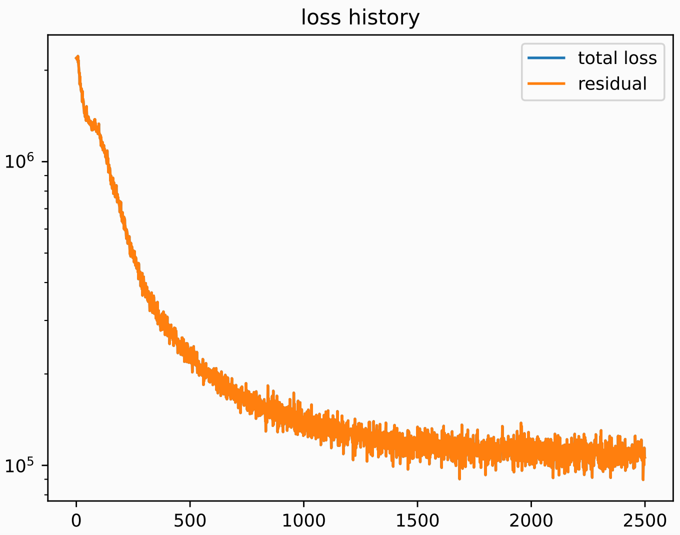
In this case, we replace the call to the neural network, going from $W_\theta(x; \mu)$ to

$$W_\theta(x; \mu, \sin(\pi a_1 x), \cos(\pi b_1 x), \dots, \sin(\pi a_K x), \cos(\pi b_K x)),$$

with $K \in \mathbb{N}$ the number of Fourier features and $(a_i)_i$, $(b_i)_i$ the trainable frequencies.

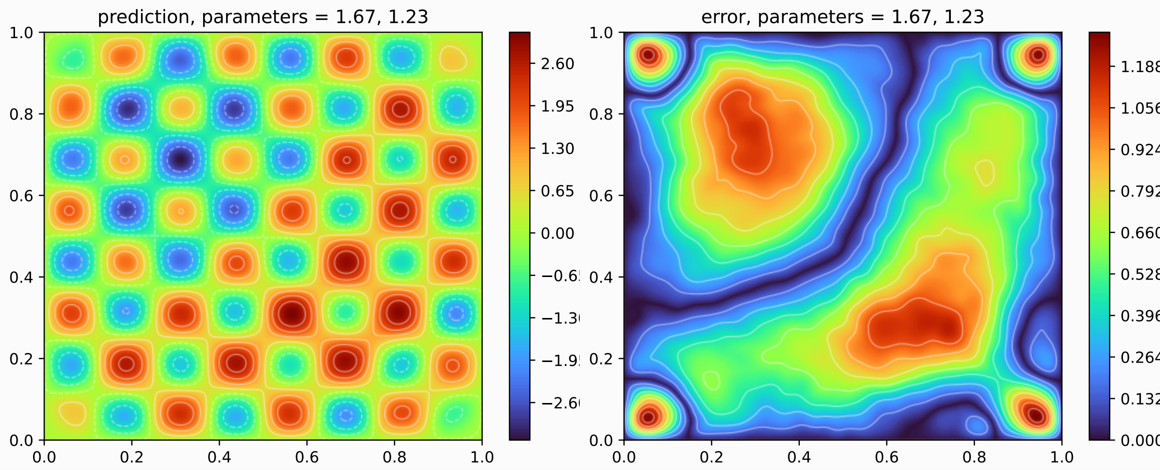
[10] See [M. Tancik et al, (2021)], but other methods exist, such as Finite Basis PINNs (FBPINNs, see [V. Dolean et al., *Comput. Method. Appl. M.* (2024)]).

High-frequency problem: with Fourier features

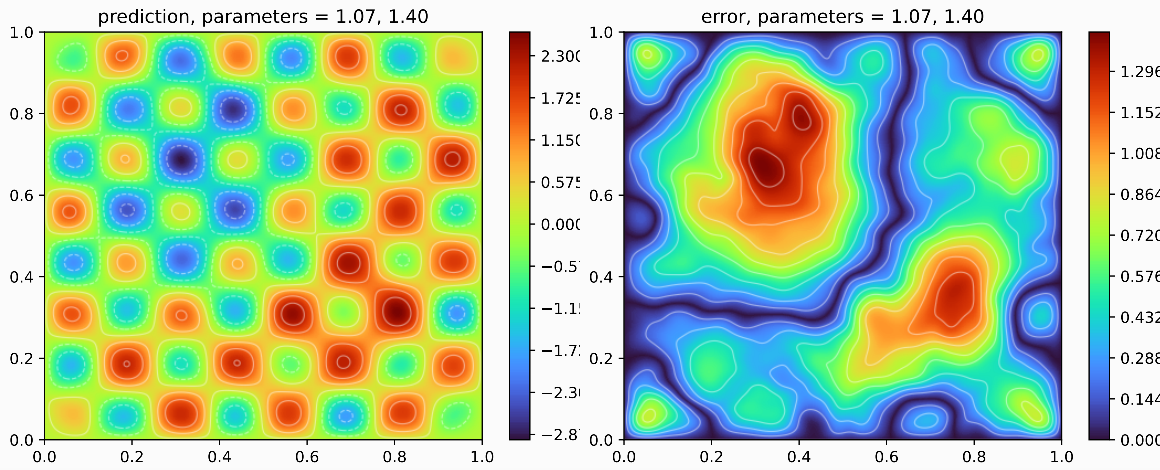


elapsed time: ~ 50 seconds

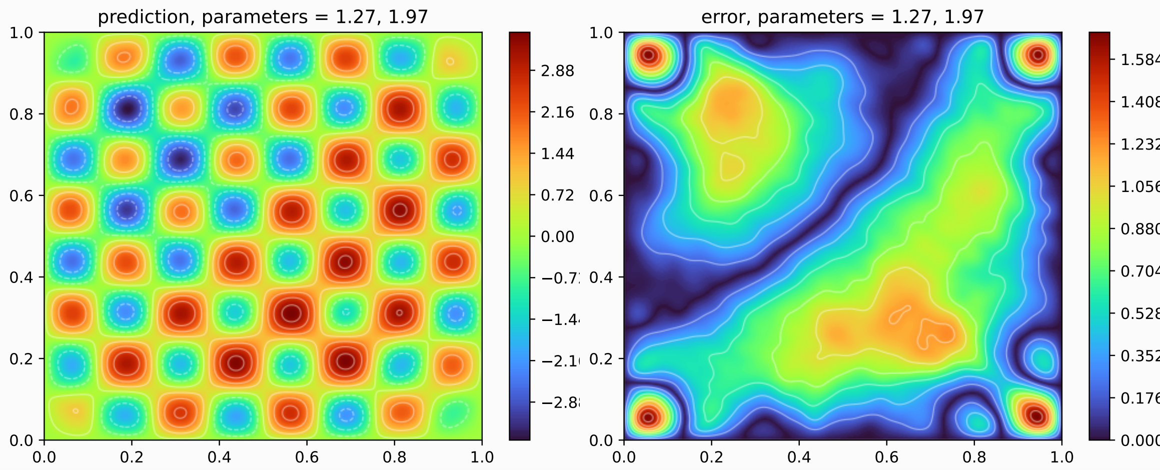
High-frequency problem: with Fourier features



High-frequency problem: with Fourier features



High-frequency problem: with Fourier features



Monte-Carlo integration: convergence

Consider an integrable and bounded function $f : \Omega \times (0,T) \to \mathbb{R}$, and define $(X_k, T_k)_k$ a sequence of independent random variables, uniformly sampled in $\Omega \times (0,T)$.

We wish to give an approximation to

$$I = \int_\Omega \int_0^T f(x,t)\, dt\, dx.$$

An estimator of I is the following:

$$\widehat{I}^n = \frac{|\Omega| T}{n} \sum_{k=1}^{n} f(X_k, T_k).$$

Since the $(X_k, T_k)_k$ are uniformly sampled, we get

$$\mathbb{E}[f(X_k, T_k)] = \frac{1}{|\Omega| T} \int_\Omega \int_0^T f(x,t)\, dt\, dx.$$

Hence, applying the **law of large numbers** tells us that, with probability 1, $\widehat{I}^n \to I$.

Monte-Carlo integration: convergence speed

We can also determine the convergence speed of the Monte-Carlo method, assuming that f^2 is integrable.

The **central limit theorem** allows us to state that

$$\sqrt{n}(\widehat{I^n} - I) \xrightarrow[n\to\infty]{} \mathcal{N}(0, \sigma^2),$$

and so, for large enough n and with probability 1,

$$\left|\widehat{I^n} - I\right| = \mathcal{O}\left(n^{-\frac{1}{2}}\right).$$

This result is **independent of the dimension** d of Ω! Contrast this with, for instance, the trapezoidal rule, with an error in $\mathcal{O}(n^{-\frac{2}{d}})$.

Error analysis – proof

To prove this result, we adapt the proof of Céa's lemma to the additive prior case.

The numerical solution u_h^+ is given for all $x \in \Omega$ by

$$u_h^+(x) = u_\theta(x) + p_h^+(x),$$

with $p_h^+ \in V_h \subset V$ solution of the new discrete variational problem. We have

$$\begin{aligned}
a(u - u_h^+, u - u_h^+) &= a\big(u - u_h^+, (u - u_\theta) - p_h^+\big) \\
&= a\big(u - u_h^+, (u - u_\theta) - p_h^+ - v_h + v_h\big), && \forall v_h \in V_h \\
&= a\big(u - u_h^+, (u - u_\theta) - v_h\big) + a\big(u - u_h^+, v_h - p_h^+\big), && \forall v_h \in V_h.
\end{aligned}$$

We will estimate both terms, one by one.

Error analysis – proof (cont'd)

Let us first estimate the second term: $a\big(u - u_h^+, v_h - p_h^+\big)$.

Using that $V_h \subset V$, we have, by Galerkin orthogonality,

$$a(u - u_h^+, v_h) = 0, \quad \forall v_h \in V_h.$$

The above equality is valid for all $v_h \in V_h$, and $v_h - p_h^+ \in V_h$. Therefore, we obtain

$$a\big(u - u_h^+, v_h - p_h^+\big) = 0, \quad \forall v_h \in V_h.$$

The second term therefore vanishes, and we are left with the first one:

$$a(u - u_h^+, u - u_h^+) = a\big(u - u_h^+, (u - u_\theta) - v_h\big), \quad \forall v_h \in V_h.$$

Error analysis – proof (cont'd)

Denoting by α and γ the coercivity and continuity constants of a, we have

$$\begin{aligned}\alpha\left\|u-u_h^+\right\|_{H^m}^2 &\leqslant a\left(u-u_h^+,u-u_h^+\right)=a\left(u-u_h^+,(u-u_\theta)-v_h\right), && \forall v_h\in V_h,\\ &\leqslant \gamma\left\|u-u_h^+\right\|_{H^m}\left\|(u-u_\theta)-v_h\right\|_{H^m}, && \forall v_h\in V_h,\end{aligned}$$

which immediately leads to

$$\|u-u_h^+\|_{H^m}\leqslant\frac{\gamma}{\alpha}\left\|(u-u_\theta)-v_h\right\|_{H^m},\quad \forall v_h\in V_h.$$

Applying the above relation to $v_h=\mathcal{I}_h(u-u_\theta)$ with $\mathcal{I}_h$ the Lagrange interpolator, and invoking classical interpolation results from [A. Ern and J.-L. Guermond, (2004)], we get

$$\|u-u_h^+\|_{H^m}\lesssim\frac{\gamma}{\alpha}h^{q+1-m}|u-u_\theta|_{H^{q+1}}.$$

Rewriting the above equation to introduce the error of the classical FEM, we get

$$\|u-u_h^+\|_{H^m}\lesssim C_{\mathrm{gain}}\,h^{q+1-m}|u|_{H^{q+1}}\qquad\text{with}\qquad C_{\mathrm{gain}}=\frac{|u-u_\theta|_{H^{q+1}}}{|u|_{H^{q+1}}},$$

which completes the proof.

Enhancing the approximation space – multiplicative prior

Another possible modification of the FEM approximation space, is to replace V_h by $V_h^\times$:

$$V_h^\times = \left\{ v_h = u_\theta p_h^\times, \quad p_h^\times \in V_h \right\}.$$

The discrete variational problem becomes:

$$\begin{pmatrix} \text{Find } u_h^\times \in V_h^\times \text{ such that} \\ \forall v_h \in V_h,\ a(u_h, v_h) = \ell(v_h) \end{pmatrix} \iff \begin{pmatrix} \text{Find } p_h \in V_h \text{ such that} \\ \forall v_h \in V_h,\ a(u_\theta p_h, v_h) = \ell(v_h). \end{pmatrix}$$

Enhancing the approximation space – multiplicative prior

Another possible modification of the FEM approximation space, is to replace V_h by $V_h^\times$:

$$V_h^\times = \{ v_h = u_\theta p_h^\times, \quad p_h^\times \in V_h \}.$$

The discrete variational problem becomes:

$$\begin{pmatrix} \text{Find } u_h^\times \in V_h^\times \text{ such that} \\ \forall v_h \in V_h,\ a(u_h, v_h) = \ell(v_h) \end{pmatrix} \iff \begin{pmatrix} \text{Find } p_h \in V_h \text{ such that} \\ \forall v_h \in V_h,\ a(u_\theta p_h, v_h) = \ell(v_h). \end{pmatrix}$$

Theorem: Let u be the exact solution of the BVP, $u_\theta \in \mathcal{H}_0^m(\Omega)$ a prior on u, and $u_h^\times \in V_h^\times$ the enhanced FEM solution (considering $\mathbb{P}_q$ polynomials, with $m \leqslant q$). Then:

$$\| u - u_h^\times \|_{H^m} \lesssim C_{\text{gain}}^\times \, h^{q+1-m} |u|_{H^{q+1}}.$$

Enhancing the approximation space – multiplicative prior

Another possible modification of the FEM approximation space, is to replace V_h by $V_h^\times$:

$$V_h^\times = \{ v_h = u_\theta p_h^\times , \quad p_h^\times \in V_h \}.$$

The discrete variational problem becomes:

$$\left(\begin{array}{c} \text{Find } u_h^\times \in V_h^\times \text{ such that} \\ \forall v_h \in V_h,\ a(u_h, v_h) = \ell(v_h) \end{array} \right) \iff \left(\begin{array}{c} \text{Find } p_h \in V_h \text{ such that} \\ \forall v_h \in V_h,\ a(u_\theta p_h, v_h) = \ell(v_h). \end{array} \right)$$

Theorem: Let u be the exact solution of the BVP, $u_\theta \in \mathcal{H}_0^m(\Omega)$ a prior on u, and $u_h^\times \in V_h^\times$ the enhanced FEM solution (considering $\mathbb{P}_q$ polynomials, with $m \leqslant q$). Then:

$$\| u - u_h^\times \|_{H^m} \lesssim C_{\text{gain}}^\times \, h^{q+1-m} |u|_{H^{q+1}}.$$

In this result, the gain constant is $C_{\text{gain}}^\times = \left| \dfrac{u}{u_\theta} \right|_{H^{q+1}} \dfrac{\| u_\theta \|_{W^{m,\infty}}}{|u|_{H^{q+1}}}$. Beware of the division!

PINNs as a DG prior: perturbed steady solution

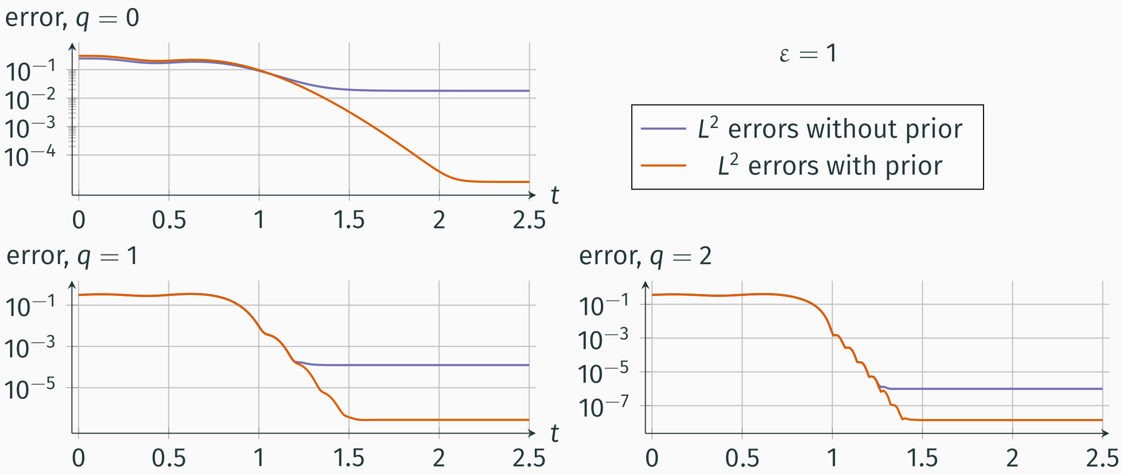
We use the DG scheme to solve the advection equation with a **perturbation of the steady solution as initial condition**:

$$\begin{cases} \partial_t W + \partial_x W = aW + bW^2 & \text{for } x \in (0,1),\ t \in (0,T), \\ W(0,x) = \left(1 + \varepsilon \sin(2\pi x)\right) W_{\text{eq}}(x) & \text{for } x \in (0,1), \\ W(t,0) = u_0 & \text{for } t \in (0,T). \end{cases}$$

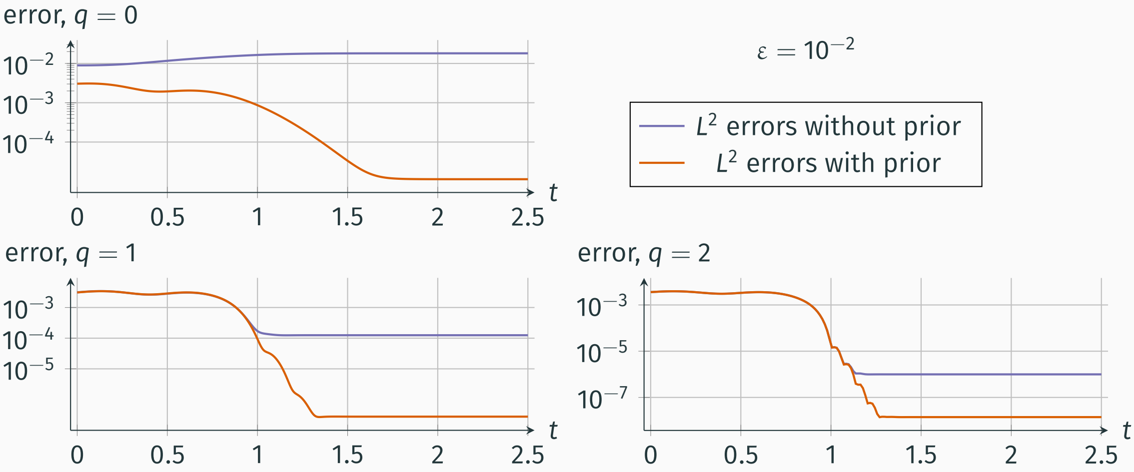
We expect:

- both schemes to **converge (in time)** towards the original, unperturbed steady solution;
- the DG scheme with prior to provide a **better approximation of the unperturbed steady solution** than the classical DG scheme.

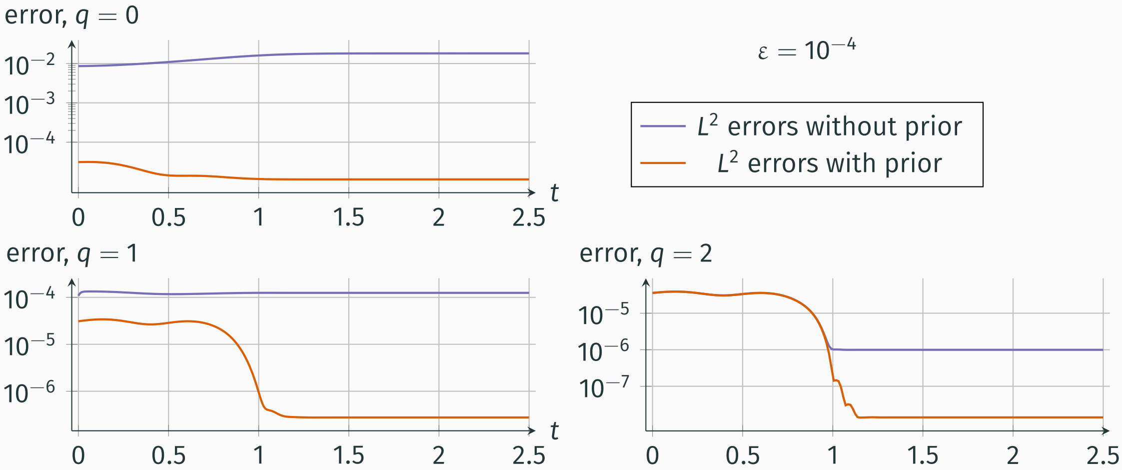
PINNs as a DG prior: perturbed steady solution



PINNs as a DG prior: perturbed steady solution



PINNs as a DG prior: perturbed steady solution



PINNs as a DG prior: unsteady solution

We use the DG scheme to solve the advection of a Gaussian bump:

$$\begin{cases} \partial_t W + \partial_x W = aW + bW^2 & \text{for } x \in (0,1),\ t \in (0,T), \\ W(0,x) = 0.1(1 + e^{-100(x-0.5)^2}) & \text{for } x \in (0,1), \\ W(t,0) = 0.1(1 + e^{-25}) & \text{for } t \in (0,T). \end{cases}$$

We expect the prior not to alter the convergence:

- both schemes to converge with the **same error rate**;
- the DG scheme with prior to provide a **similar approximation** to the classical DG scheme.

PINNs as a DG prior: unsteady solution

We compute the errors in x between the exact and approximate solutions:

- for several numbers of basis elements and discretization cells,
- using $a = 0.75$; $b = 0.75$; $u_0 = 0.15$.

	without prior		with prior		
cells	error	order	error	order	gain
10	4.04e-02	—	5.04e-02	—	0.80
20	3.46e-02	0.22	4.28e-02	0.24	0.81
40	2.84e-02	0.28	3.50e-02	0.29	0.81
80	2.15e-02	0.40	2.64e-02	0.40	0.81
160	1.47e-02	0.55	1.81e-02	0.55	0.81

(a) Errors with a basis composed of one element.

PINNs as a DG prior: unsteady solution

We compute the errors in x between the exact and approximate solutions:

- for several numbers of basis elements and discretization cells,
- using $a = 0.75$; $b = 0.75$; $u_0 = 0.15$.

	without prior		with prior		
cells	error	order	error	order	gain
10	1.92e-02	—	1.93e-02	—	1.00
20	6.26e-03	1.62	6.27e-03	1.62	1.00
40	1.19e-03	2.39	1.20e-03	2.39	1.00
80	1.99e-04	2.59	1.99e-04	2.59	1.00
160	4.19e-05	2.24	4.20e-05	2.24	1.00

(b) Errors with a basis composed of two elements.

PINNs as a DG prior: unsteady solution

We compute the errors in x between the exact and approximate solutions:

- for several numbers of basis elements and discretization cells,
- using $a = 0.75$; $b = 0.75$; $u_0 = 0.15$.

	without prior		with prior		
cells	error	order	error	order	gain
10	5.15e-03	—	5.15e-03	—	1.00
20	4.56e-04	3.50	4.56e-04	3.50	1.00
40	4.55e-05	3.32	4.55e-05	3.32	1.00
80	5.42e-06	3.07	5.42e-06	3.07	1.00
160	6.75e-07	3.01	6.75e-07	3.01	1.00

(c) Errors with a basis composed of three elements.

PINNs as a DG prior: unsteady solution

We compute the errors in x between the exact and approximate solutions:

- for several numbers of basis elements and discretization cells,
- using $a = 0.75$; $b = 0.75$; $u_0 = 0.15$.

	without prior		with prior		
cells	error	order	error	order	gain
10	4.72e-04	—	4.72e-04	—	1.00
20	2.87e-05	4.04	2.87e-05	4.04	1.00
40	1.81e-06	3.99	1.81e-06	3.99	1.00
80	1.14e-07	3.98	1.14e-07	3.98	1.00
160	7.20e-09	3.99	7.20e-09	3.99	1.00

(d) Errors with a basis composed of four elements.