Neural and hybrid methods for elliptic PDEs

H. Barucq[3], M. Duprez[1], F. Faucher[3], E. Franck[12], F. Lecourtier[3]
V. Lleras [4], V. Michel-Dansac[12], L. Navoret[12], N. Victorion[3]

Lille, 12/06/2024

New trends in the numerical analysis of PDEs

[1]Inria Nancy Grand Est, France
[2]IRMA, Strasbourg university, France
[3]Inria Bordeaux, Pau center, France
[4]Montpellier University, France

Outline

Machine learning and numerical methods

Link between ML and numerics

- Common objective of ML and numerical analysis.
- 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 function space.
- **Difficulty**: we want to find an infinite dimensional object.

Solution: parametric models

- We choose a known parametric function $f_\theta(x)$ with **unknown parameters** θ:
- The problem becomes

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

- ML approaches : we find θ constraining the approximation by the data
- We assume that we have examples $\{(x_1, f_1), \ldots, (x_N, f_N)\}$ 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 is a good approximation of the function on each data point. We solve:

$$\arg\min_\theta \sum_{i=1}^{N} d(u_i, u_\theta(x_i))$$

- Numerical methods: we construct θ, constraining the approximation by the physical equation
- Principle of a numerical method:

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

with L a differential or integral operator and A, b forming a linear or nonlinear system.

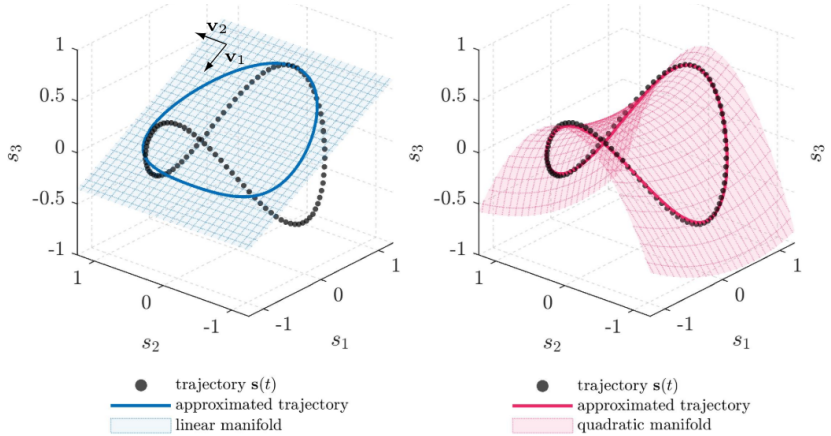
Deep learning revolution

- **Deep learning revolution** in signal and language processing: combination of huge numbers of data, massive GPU computing and efficient models in large dimensions.
- **Classical parametric model in ML**: linear, polynomial or kernel models.

Models

Main change: we have moved massively from linear models with respect to parameters to nonlinear models with respect to the parameters.

- Effects:
  - From a finite-dimensional vector space, the approximation space becomes a finite-dimensional manifold.
  - We move from convex quadratic optimization (mainly) to non-convex optimization
  - Problems in large dimensions are easier to solve.



- Linear Vs Manifold projection for reduced modeling (K. Willcox et al).

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 $L_l(\mathbf{x}_l) : \mathbb{R}^{d_l} \to \mathbb{R}^{d_{l+1}}$ 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 $\mathbf{b}_l$.

- **Goal**: using these models, we expect to require fewer DoFs, not to require a mesh, and to deal with larger dimensions.
- **Key point**: in the NN framework, derivatives can be exactly computed through automatic differentiation tools.

Numerical method and Galerkin projection

General method

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

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

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

  - Galerkin projection:

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

  - The problem is quadratic in θ. The parameters making the gradient vanish satisfy

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

  - Computing the derivative (exactly) and the integral (numerically) leads to

$$A\theta = b$$

- **Second approach**: Least square Galerkin projection

$$\min_{T_\theta \in V_\theta} \int_\Omega | - \Delta T_\theta - f|^2 dx$$

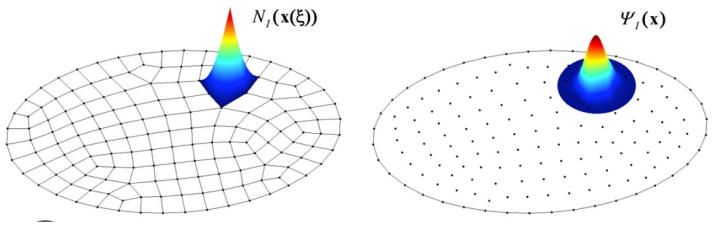
Computing the exact solution to this problem again yields linear system to solve.

Approximation vector spaces

Parametric models used by classical numerical methods

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

- Classical mesh-based methods (local basis functions):
  - **Finite elements**: C^p continuity between the cells (depending on the finite element) so is $\phi_i(x)$ piecewise polynomial.
  - **Finite differences**: pointwise values so $\phi_i(x) = \delta_{x_i}(x)$ with x_i a mesh node.
- Classical mesh-free methods (local or global basis functions):
  - **Spectral**: we use Hilbert basis, e.g. $\phi_i(x) = sin(2\pi k_i x)$ (same with Hermite, Laguerre, Legendre polynomials). Meshless depend of the BC.
  - **Radial basis**: we use radial basis, e.g. $\phi_i = \phi(|x - x_i|)$ with ϕ a Gaussian or $\frac{1}{1+\sigma^2 x^2}$.



- Except spectral methods, these approaches are local in space, and the number of DOFs increase exponentially with the dimension.

PINNs (Physics-Informed Neural Networks)

PINNs and Deep Ritz formulation

- The Galerkin/LS Galerkin methods rely on an L^2 projection of the equation in a finite vector space V_θ.
- The neural methods use the same principle, replacing V_θ by the manifold:

$$\mathcal{M}_\theta = \{u_\theta(x), \quad \theta \in \mathbb{R}^n\}$$

- PINNs (ref) use an LS Galerkin projection and Deep Ritz (ref) a Galerkin projection.
- For the equation $Lu = f(x)$ with non-homogeneous Dirichlet BC, with L a differential operator, the PINNs approach solve:

$$\min_{u_\theta \in \mathcal{M}_\theta} \left(J_r(\theta) + J_b(\theta)\right),$$

with

$$J_r(\theta) = \int_\Omega \|L(u_\theta) - f(x)\|_2^2 dx, \quad J_b(\theta) = \int_{\partial\Omega} \|u_\theta - g(x)\|_2^2 dx,$$

- Since the parametric model are nonlinear, this problem is non-convex.
- We can remove the BC loss J_b using the manifold

$$\mathcal{M}_{g,\theta} = \{u_\theta(x)\phi(x) + g(x), \quad \theta \in \mathbb{R}^n\}$$

with $\phi(x)$ a level set of the domain. Similar trick for Neumann/Robin BC [PinnsBC].

Monte-Carlo method

- **Last point**: we need to approximate the integrals. First approach: quadrature rules. To be accurate and valid on general geometries, a mesh is needed; furthermore, these methods scale poorly with the dimension.
- **Classical choice**: Monte Carlo. Scale well with the dimension, flexible, and compatible with stochastic gradient method classically used for NNs.
- General case:

$$\int_\Omega \|L(u_\theta) - f(x)\|_2^2 dx = \mathbb{E}_{\mathcal{U}(\Omega)}[\|L(u_\theta) - f(x)\|_2^2]$$

with $\mathcal{U}(\Omega)$ a uniform law on Ω.

$$\mathbb{E}_{\mathcal{U}(\Omega)}[\|L(u_\theta) - f(x)\|_2^2] = \mathbb{E}_{\mathcal{G}}\left[\frac{\|L(u_\theta) - f(x)\|_2^2}{g(x)}\right]$$

with $\mathcal{G}$ a probability law of density $g(x)$. With the law of large numbers, we obtain

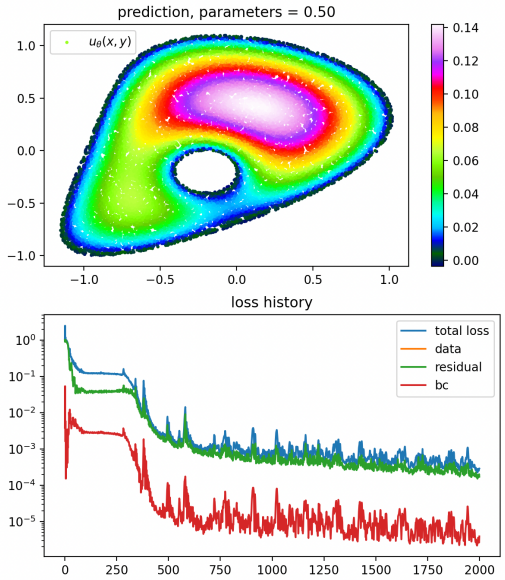
$$\int_\Omega \|L(u_\theta) - f(x)\|_2^2 dx \approx \frac{1}{N}\sum_{i=1}^{N} \frac{\|L(u_\theta(x_i)) - f(x_i)\|_2^2}{g(x_i)}$$

- In general, we take $g(x) = 1$ or $g(x) \sim \|L(u_\theta) - f(x)\|_2^2$.

Examples and complex geometries

How to deal with general geometries?

- Sample in a simple domain (circle) and apply a mapping to your domain:



- Use a level set (positive outside the domain, negative inside). For unknown level sets, we can learn the level set solving the Eikonal equation with PINNs.

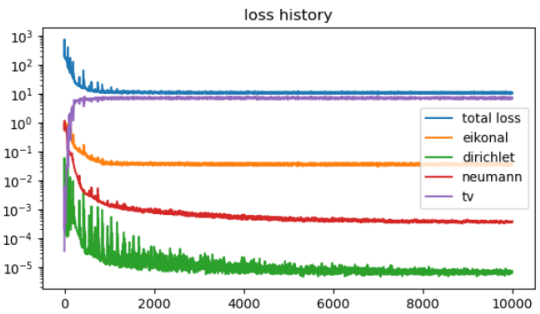


FIGURE 11 – Loss (tv=lap).

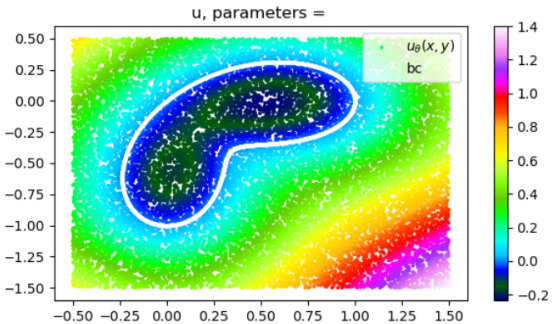


FIGURE 12 – Levelset ϕ_θ.

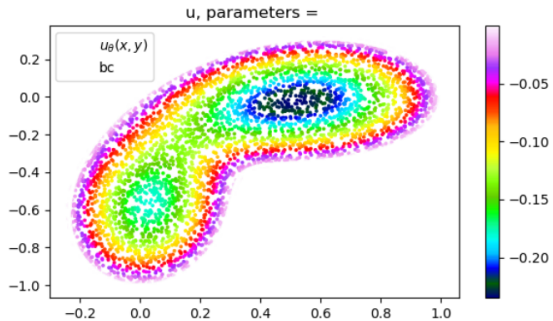


FIGURE 13 – Levelset $\phi_\theta < 0$.

Parametric problems

- Consider the problem

$$\begin{cases} Lu(x) = -\nabla \cdot (\mathbb{K}(x;\alpha)\nabla u) = f(x;\beta), & x \in \Omega \\ u(x) = g(x;\gamma), & x \in \partial\Omega \end{cases}$$

  with $\mu = (\alpha, \beta, \gamma) \in \mathbb{R}^p$ a set of parameters.
- We wish to solve the problem for many parameters (for applications in uncertainty propagation, optimal control, etc.).
- With PINNs it is possible in one training. For example, we solve:

$$\min_{u_\theta \in \mathcal{M}_\theta} J_r(\theta)$$

  with

$$J_r(\theta) = \int_{\mathbb{R}^p} \int_\Omega \|L(u_\theta) - f(x)\|_2^2 p(\mu) dx d\mu$$

  with $p(\mu)$ the distribution of parameters and

$$\mathcal{M}_{g,\theta} = \{u_\theta(x,\mu)\phi(x) + g(x;\gamma), \quad \theta \in \mathbb{R}^n\}$$

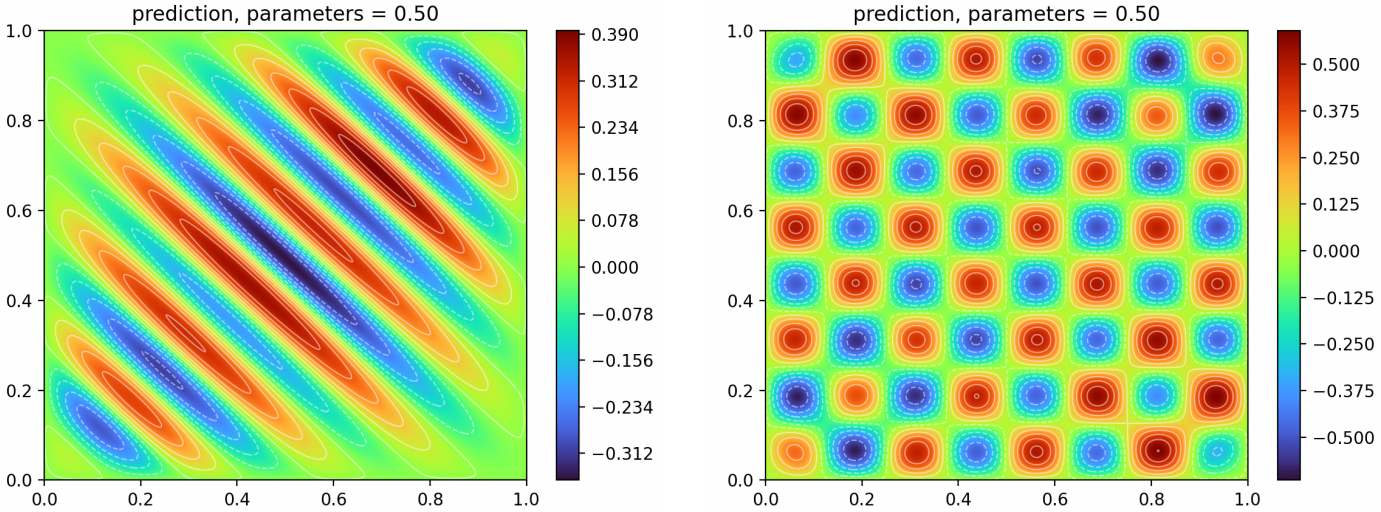
- The good behavior of NNs and Monte Carlo in large dimensions is essential.

Spectral biais and high frequencies

Spectral bias

Using the NTK theory makes it possible to study **Spectral bias of MLP**. MLPs first learn low frequencies, before learning the high frequencies (with difficulty).

- We solve $-\Delta u = 128 \sin(8\pi x)\sin(8\pi y)$. First try (left figure): classical MLP with sine activation functions (to help).



- To solve this problem for PINNs, we add Fourier features (right figure). We replace

$$NN_\theta(x) \quad \text{by} \quad NN_\theta(x, \sin(2\pi k_1 x), \ldots, \sin(2\pi k_n x))$$

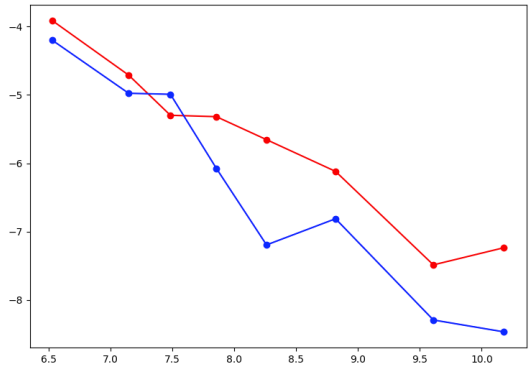
with $(k_1, \ldots . k_n)$ trainable parameters.

Advantages and disadvantages

Disadvantages

The main disadvantage of the Neural approach are the difficulty to obtain a good accuracy, and the fact that only asymptotic convergence results are available.

- Consider a 2D Laplacian solves with a 5-layer neural network and increase the size (685 weights for the smallest network and 26300 weights for the largest).
- Two learning rates:



Advantage

Mesh-free and ratio accuracy/degree of freedom less sensitive to the dimension.

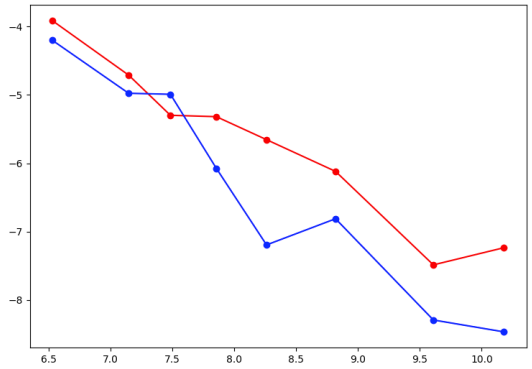
FE	N_{dof}	CPU	Error
1D	100	-	-
2D	$1E^4$	$\approx 10/20sec$	$\approx 2E^{-3}$
3D	$1E^6$	≈ 2h	$\approx 2E^{-3}$

Advantages and disadvantages

Disadvantages

The main disadvantage of the Neural approach are the difficulty to obtain a good accuracy, and the fact that only asymptotic convergence results are available.

- Consider a 2D Laplacian solves with a 5-layer neural network and increase the size (685 weights for the smallest network and 26300 weights for the largest).
- Two learning rates:



Advantage

Mesh-free and ratio accuracy/degree of freedom less sensitive to the dimension.

PINNs	N_{dof}	CPU	Error
1D	5081	30-55sec	$3E^{-4}$-$6E^{-4}$
2D	5121	80-100sec	$4E^{-4}$-$2E^{-3}$
3D	5161	110-140sec	$1E^{-3}$-$4E^{-3}$

Hybrid PINNs-FE approach

Prediction-correction method

Hybrid methods

In this context, hybrid methods combine classical numerical methods and numerical methods based on neural representations.

Objectives

Taking the best of both worlds: the accuracy of classical numerical methods, and the mesh-free large-dimensional capabilities of neural-based numerical methods [FEhybrid].

General Idea

- **Offline/Online process**: train a Neural Network (PINNs, NGs, or NOs) to obtain a large family of approximate solutions.
- **Online process**: predict the solution associated to our test case using the NN.
- **Online process**: correct the solution with a numerical method.

Additive and multiplicative formulation

- We consider the following elliptic problem:

$$\begin{cases} Lu = -\partial_{xx}u(x) + v\partial_x u(x) + ru(x) = f, & \forall x \in \Omega \\ u(x) = g(x), & \forall x \in \partial\Omega \end{cases}$$

- We assume that we have a continuous prior given by a parametric PINN $u_\theta(x;\mu)$
- We propose the following corrections of the finite element basis functions:

$$u(x) = u_\theta(x;\mu) + p_h(x), \quad u(x) = u_\theta(x;\mu)p_h(x),$$

with $p_h(x)$ a perturbation discretized using P_k Lagrange finite element.

- For the **first approach (additive prior)**, we solve in practice:

$$\begin{cases} Lp_h(x) = f - Lu_\theta(x;\mu), & \forall x \in \Omega \\ p_h(x) = g - u_\theta(x;\mu), & \forall x \in \partial\Omega \end{cases}$$

- For the **second approach (multiplicative prior)**, we need $u_\theta(x) \neq 0$, so we take $C_m > 0$ and we solve:

$$\begin{cases} L(u_\theta(x;\mu)p_h(x)) = f, & \forall x \in \Omega \\ p_h(x) = \frac{g}{u_\theta(x;\mu)} + C_m, & \forall x \in \partial\Omega \end{cases}$$

- **Additional cost**: increase the quadrature rule degree where the network is integrated

Error estimates

Additive approach

- We rewrite the Cea lemma for $u_h(x) = u_\theta(x) + p_h(x)$. We obtain

$$\|u - u_h\| \leq \frac{M}{\alpha}\|u - u_\theta - I_h(u - u_\theta)\|$$

with I_h the interpolator. Using the classical result of P_k Lagrange interpolator we obtain

$$\|u - u_h\|_{H^m} \leq \frac{M}{\alpha} C h^{k+1-m} \underbrace{\left(\frac{|u - u_\theta|_{H^m}}{|u|_{H^m}}\right)}_{gain} |u|_{H^m}$$

- It is equivalent to a Petrov-Galerkin method with affine trial space and P_k test space.

Key point

The prior must give a good approximation of the m^{th} derivative.

- We can also make the proof for multiplicative approach (rewriting the modified interpolation operator using the usual one). In practice the additive approach is more efficient in a large majority of cases.

Results I

- Test 1:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. For the test case the solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(2x)\,\sin(2y)\,e^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2 \sim \mathcal{U}(-0.5, 0.5)$.

- Gain at fixed size.
- First we use a classical PINNs (called L^2 PINNs)

	Gains on PINNs				Gains on FEM			
N	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
20	15.7	48.35	33.64	5.57	134.31	377.36	269.4	43.67
40	61.47	195.75	135.41	23.21	131.18	362.09	262.12	41.67

	Gains on PINNs				Gains on FEM			
N	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
20	244.81	996.23	655.08	153.63	67.12	165.13	135.21	21.37
40	2,056.2	8,345.4	5,504.89	1,287.16	66.52	159.73	132.05	20.38

	Gains on PINNs				Gains on FEM			
N	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
20	2,804.27	11,797.23	7,607.51	1,780.7	39.72	72.99	61.85	7.05
40	50,989.23	212,714.99	137,711.77	32,125.57	40.02	73	61.98	6.92

Results I

- Test 1:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. For the test case the solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(2x)\,\sin(2y)\,e^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2 \sim \mathcal{U}(-0.5, 0.5)$.

- Gain at fixed size.
- First we use a H_1 PINNs

	Gains on PINNs				Gains on FEM			
N	min	max	mean	std	min	max	mean	std
20	18.28	66.19	43.42	12.47	243.79	874.3	633.45	137.97
40	73.45	272.36	176.52	51.82	241.8	843.29	621.68	132.89

	Gains on PINNs				Gains on FEM			
N	min	max	mean	std	min	max	mean	std
20	362.57	2,052.78	1,025.28	409.17	177.74	476.76	376.16	75.9
40	3,081.22	17,532.62	8,725.57	3,494.26	177.16	472.55	371.93	74.85

	Gains on PINNs				Gains on FEM			
N	min	max	mean	std	min	max	mean	std
20	4,879.13	32,757.68	14,646.89	6,699.18	116.52	298.33	208.35	43.62
40	88,736.63	587,716.86	264,383.45	120,240.85	117.46	296.34	208.29	43.16

Results I

- Test 1:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. For the test case the solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(2x)\,\sin(2y)\,e^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2 \sim \mathcal{U}(-0.5, 0.5)$.

- Gain at fixed error (Finite element P_1)

	N_{dof}	CPU	Error
Pinns L^2	x	4min15	5.21×10^{-3}
Pinns H^1	x	x	2.0×10^{-3}
Correction 20^2 (L^2)	400	1.1sec	1.42×10^{-4}
Correction 20^2 (H^1)	400	1.1sec	5.8×10^{-5}
FE 160^2	25600	1min20sec	5.46×10^{-4}
FE 320^2	102400	5min22sec	1.36×10^{-4}

- The error is the average error on a set of 10 parameters.
- CPU time for 100 simulations varying parameters: 355sec for our method (L^2 version), 32200 sec for FE. CPU divided by 90.7.
- CPU time for 100 simulations varying parameters: 1450sec for our method (L^2 version), 322000 sec for FE. CPU divided by 2220.

Results II

- Test 2:

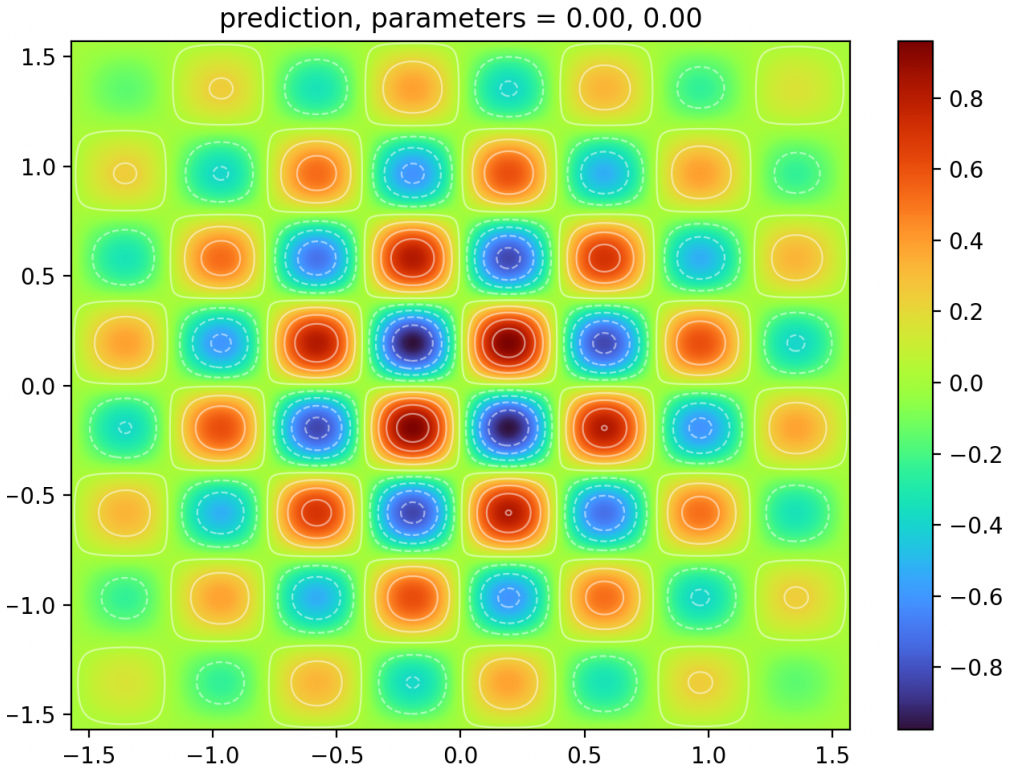
$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. For the test case the solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(8x)\,\sin(8y)\,\times 10^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2 \sim \mathcal{U}(-0.5, 0.5)$.

- Example of solution



Results II

- Test 2:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. For the test case the solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(8x)\,\sin(8y)\,\times 10^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2 \sim \mathcal{U}(-0.5, 0.5)$.

- Gain at fixed size

	Gains on PINNs				Gains on FEM			
N	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
20	9.17	36.13	19.79	6.63	112.2	454.43	349.41	82.75
40	26.14	111.44	58.86	19.8	106.01	388.96	308.49	71.81

	Gains on PINNs				Gains on FEM			
N	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
20	35.47	166.68	87.44	29.18	65.7	206.07	157.83	37.13
40	207.56	1,102.21	524.38	181.75	52.97	141.53	111.17	22.44

	Gains on PINNs				Gains on FEM			
N	**min**	**max**	**mean**	**std**	**min**	**max**	**mean**	**std**
20	75.86	499.24	215.89	79.51	28.91	64.9	52.36	8
40	999.27	6,317.61	2,665.31	1,003.72	20.09	42.2	34.3	5.19

Results II

- Test 2:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. For the test case the solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(8x)\,\sin(8y)\,\times 10^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2 \sim \mathcal{U}(-0.5, 0.5)$.

- Gain at fixed error (Finite element P_1)

	N_{dof}	CPU	Error
Pinns	28045	13min	2.4×10^{-2}
Correction 20^2	400	2sec	1.1×10^{-3}
FE 160^2	25600	1min54	7.8×10^{-3}
FE 320^2	102400	7m29	1.95×10^{-3}

- The error is the average error on a set of 10 parameters.
- CPU time for 100 simulations varying parameters: 980sec for our method, 44900 sec for FE. CPU divided by 45.8.
- CPU time for 1000 simulations varying parameters: 2780sec for our method, 449000 sec for FE. CPU divided by 161.

Results III

- Test 3:

$$\begin{cases} -\nabla \cdot (\mathbb{K} \nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. The source is given by

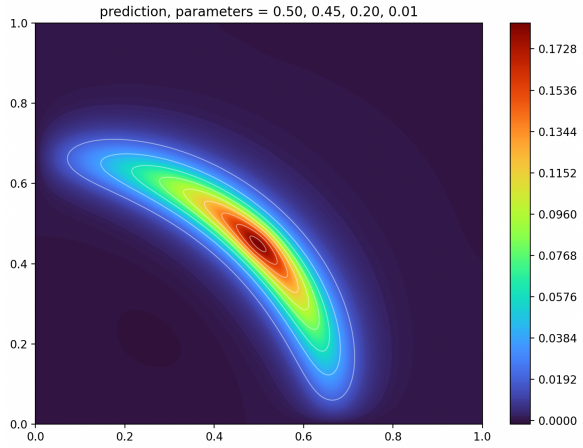
$$f(x, y) = 10 \exp(-((x1 - c1)^2 + (x2 - c2)^2)/(0.025\sigma^2))$$

and the anisotropy matrix is given by

$$K = \begin{pmatrix} \epsilon x^2 + y^2 & (\epsilon - 1)xy \\ (\epsilon - 1)xy & x^2 + \epsilon y^2 \end{pmatrix}$$

with $c_1, c_2 \sim \mathcal{U}(-0.5, 0.5)$, $\sigma \sim \mathcal{U}(0.1, 0.8)$ and $\epsilon \sim \mathcal{U}(0.01, 0.9)$.

- Example of solution (no analytic solution: we will compare with a fine solution)



- Results less good for small ϵ.

Results III

- Test 3:

$$\begin{cases} -\nabla \cdot (\mathbb{K} \nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. The source is given by

$$f(x, y) = 10 \exp(-((x1 - c1)^2 + (x2 - c2)^2)/(0.025\sigma^2))$$

and the anisotropy matrix is given by

$$K = \begin{pmatrix} \epsilon x^2 + y^2 & (\epsilon - 1)xy \\ (\epsilon - 1)xy & x^2 + \epsilon y^2 \end{pmatrix}$$

with $c_1, c_2 \sim \mathcal{U}(-0.5, 0.5)$, $\sigma \sim \mathcal{U}(0.1, 0.8)$ and $\epsilon \sim \mathcal{U}(0.01, 0.9)$.

- Gain at fixed error:

	N_{dof}	CPU	Error
Pinns		30min	2.86×10^{-2}
Correction 20^2	400	1sec	1.40×10^{-3}
Correction 40^2	400	3sec	3.3×10^{-4}
FE 80^2	6400	6sec	2.13×10^{-3}
FE 240^2	57600	55sec	2.38×10^{-4}

- CPU time for 100 simulations varying parameters (precision $\approx 2 \times 10^{-3}$): 1900sec for our method, 600 sec for FE. CPU multiplied by 3.1.
- CPU time for 100 simulations varying parameters (precision $\approx 2 \times 10^{-3}$): 2800sec for our method, 3000 sec for FE. CPU divided by 1.1.
- Results less good for small ϵ.

Results III

- Test 3:

$$\begin{cases} -\nabla \cdot (\mathbb{K} \nabla u) = f, & \text{in } \Omega, \\ u = 0, & \text{on } \Gamma. \end{cases}$$

We define Ω by the square $\Omega = [-0.5\pi, 0.5\pi]^2$. The source is given by

$$f(x, y) = 10 \exp(-((x1 - c1)^2 + (x2 - c2)^2)/(0.025\sigma^2))$$

and the anisotropy matrix is given by

$$K = \begin{pmatrix} \epsilon x^2 + y^2 & (\epsilon - 1)xy \\ (\epsilon - 1)xy & x^2 + \epsilon y^2 \end{pmatrix}$$

with $c_1, c_2 \sim \mathcal{U}(-0.5, 0.5)$, $\sigma \sim \mathcal{U}(0.1, 0.8)$ and $\epsilon \sim \mathcal{U}(0.01, 0.9)$.

- Gain at fixed error:

	N_{dof}	CPU	Error
Pinns		30min	2.86×10^{-2}
Correction 20^2	400	1sec	1.40×10^{-3}
Correction 40^2	400	3sec	3.3×10^{-4}
FE 80^2	6400	6sec	2.13×10^{-3}
FE 240^2	57600	55sec	2.38×10^{-4}

- CPU time for 100 simulations varying parameters (precision $\approx 2 \times 10^{-4}$): 2100sec for our method, 5500 sec for FE. CPU divided by 2.62.
- CPU time for 100 simulations varying parameters (precision $\approx 2 \times 10^{-4}$): 4800sec for our method, 55000 sec for FE. CPU divided by 11.5.
- Results less good for small ϵ.

Results IV

- Test 4:

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

We define Ω by the cube $\Omega = [-0.5\pi, 0.5\pi]^3$. The analytic solution u_{ex} is given by

$$u_{ex}(x, y) = \sin(2x)\,\sin(2y)\,\sin(2z)\,\times 10^{-\frac{1}{2}((x-\mu_1)^2+(y-\mu_2)^2+(z-\mu_3)^2)},$$

with homogeneous BC on Ω (i.e. $g = 0$) and $\mu_1, \mu_2, \mu_3 \sim \mathcal{U}(-0.5, 0.5)$.

- Gain at fixed error (Finite element P_1)

	N_{dof}	CPU	Error
Pinns		2min30sec	1.4×10^{-2}
Correction 20^2	400	1min30sec	6.6×10^{-4}
FE 80^3	5.12×10^4	1h1min	3.6×10^{-3}
FE 100^3	1×10^6	2h21sec	2.3×10^{-3}

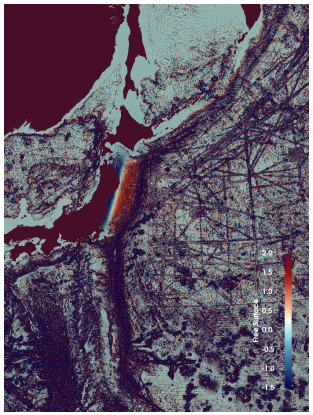
- The error is the error on 1 parameter set.
- **Pinns + correction**: 4min vs 2h for FE 100^2 with better error.
- The FE uses Skyline/diagonal storage (made for the LU decomposition), which is not efficient here, as well as an iterative solver.

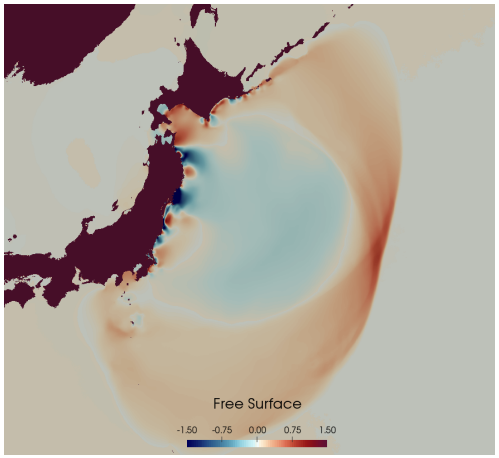
Coupling with hyperbolic systems

- In the team, most of us are interested in hyperbolic systems:

$$\partial_t \boldsymbol{U} + \nabla \cdot \boldsymbol{F}(\boldsymbol{U}) = \boldsymbol{S}(\boldsymbol{U})$$

- It is important to have a good preservation of the steady state $\nabla \cdot \boldsymbol{F}(\boldsymbol{U}) = \boldsymbol{S}(\boldsymbol{U})$.
- **Example**: Lake at rest for shallow water:





- **Exactly Well-Balanced schemes**: exact preservation of the steady state.
  Approximately Well-Balanced schemes: preserve the steady state with a higher accuracy than the scheme's.
- Building exact WB schemes is difficult for some equilibria, or for 2D flows.

Steady solutions

General steady solutions are solutions of:

$$-\nabla(D(\boldsymbol{U})\nabla \boldsymbol{U}) + \nabla \cdot \boldsymbol{A}(\boldsymbol{U}) + \boldsymbol{C}(\boldsymbol{U}) = 0$$

Coupling with hyperbolic systems

Idea

We want to compute a family of solutions with NN-based methods and plug it in the DG scheme to increase the accuracy close to the equilibrium [DGHybrid].

- The classical modal DG scheme uses the local representation:

$$u_{|\Omega_k}(x) = \sum_{l=0}^{q} \alpha_l \phi_l(x)^k, \text{ with } \quad [\phi_1^k, ...\phi_q^k] = [1, (x - x_k), ...(x - x_k)^q]$$

- If $u_\theta(x)$ is an approximation of the equilibrium, we propose the basis

$$V_1 = [u_\theta(x), (x - x_k), ...(x - x_k)^q], \text{ or } V_2 = u_\theta(x)[1, (x - x_k), ...(x - x_k)^q]$$

Estimate on the projection error

Assume that the prior u_θ satisfies

$$u_\theta(x; \mu)^2 > m^2 > 0, \quad \forall x \in \Omega, \quad \forall \mu \in \mathbb{P}$$

and consider the vector space V_2. For any function $u \in H^{q+1}(\Omega)$,

$$\|u - P_h(u)\|_{L^2(\Omega)} \lesssim \left| \frac{u}{u_\theta} \right|_{H^{q+1}(\Omega)} (\Delta x_k)^{q+1} \|u_\theta\|_{L^\infty(\Omega)}.$$

- Proofs made for the scalar case.

Euler-Poisson system in spherical geometry

- We consider the Euler-Poisson system in spherical geometry

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

- **First application**: we consider the barotropic pressure law $p(\rho; \kappa, \gamma) = \kappa \rho^\gamma$ such that the steady solutions satisfy

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

- The PINN yields an approximation of $\rho_\theta(x, \kappa, \gamma)$
- **Second application**: we consider the ideal gas pressure law $p(\rho; \kappa, \gamma) = \kappa \rho T(r)$, with $T(r) = e^{-\alpha r}$, such that the steady solutions satisfy

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

- The PINN yields an approximation of $\rho_\theta(x, \kappa, \alpha)$
- To simulate a flow around a steady solution, we need a scheme that is very accurate on the steady solution.

Results

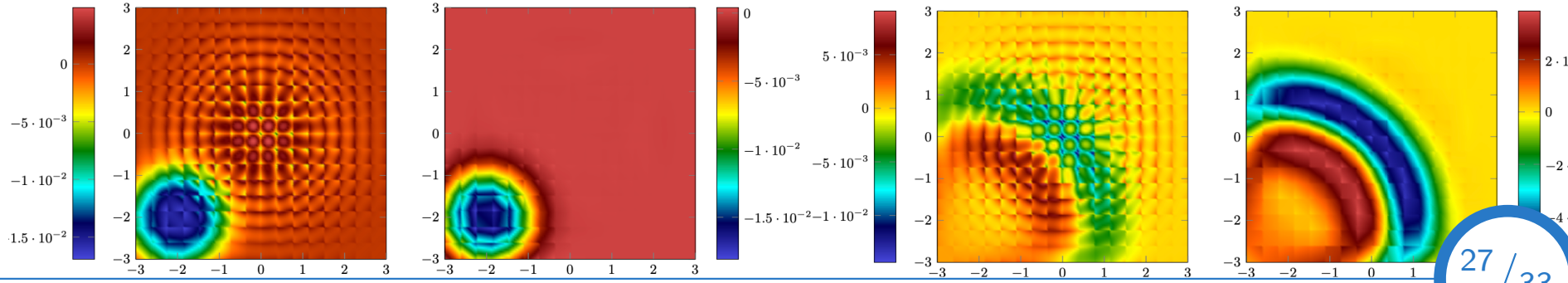
- Training takes about 10 minutes on an old GPU,with **no data**, only the PINN loss.
- We take a quadrature of degree $n_Q = n_G + 1$ (sometimes, more accurate quadrature formulas are needed).
- Barotropic case:

	minimum gain			average gain			maximum gain		
q	ρ	Q	E	ρ	Q	E	ρ	Q	E
0	19.14	2.33	17.04	233.48	3.73	197.28	510.42	4.48	371.87
1	7.61	8.28	6.98	158.25	188.92	130.57	1095.68	1291.90	1024.59
2	0.14	0.22	2.99	12.11	16.55	23.73	89.47	109.93	169.28

- ideal gas case:

	minimum gain			average gain			maximum gain		
q	ρ	Q	E	ρ	Q	E	ρ	Q	E
0	13.30	1.05	16.24	151.96	1.88	150.63	600.13	2.91	473.83
1	6.30	7.53	5.40	72.63	77.20	51.09	321.20	302.58	257.19
2	3.35	3.45	2.20	18.96	22.58	13.56	55.47	63.45	47.83

- 2D shallow water equations: equilibrium with $\boldsymbol{u} \neq 0$ + small perturbation. Plot the deviation to equilibrium:



Greedy PINNs

Principle

Objectives

Solve, with good accuracy, large-dimensional parametric elliptic problems. We wish to use an approach with only neural networks. How to increase the accuracy ?

Idea

Correct the first network with a second one, iterate. Refs: [mlevel] [mStage] -[GalNeu].

- We can write that as a greedy algorithm [Greedy11].
  - We consider the following submanifold approximation $\mathcal{M}_i, \quad 1 \leq i \leq d$
  - We initialize the greedy basis: $\mathcal{B} = \emptyset$, $u_h(x, \mu) = 0$
  - While $k < K$ and $\mid R(u_h) \mid > \epsilon$
    - We solve

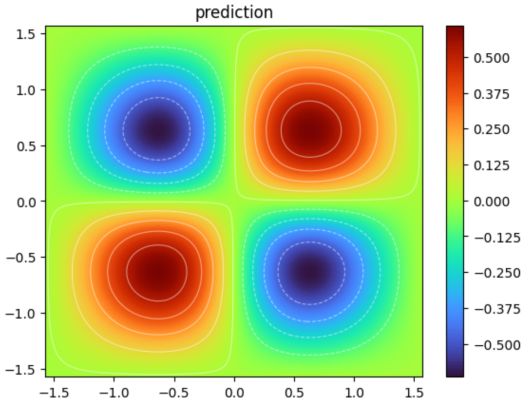
$$\text{argmin}_{\theta_k} \left(\int_{\mathcal{P}} \int_{\Omega} R(u_h(x, \mu), u_k(x, \mu)) dx + \lambda \int_{\mathcal{P}} \int_{\partial\Omega} B(u_h(x, \mu), u_k(x, \mu)) dx \right)$$

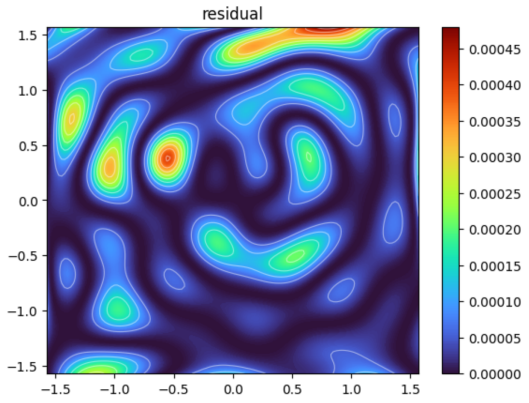
    - We compute $(\alpha_0, ...\alpha_k)$ with a Galerkin projection. Gives global approximation $u_h(x, \mu) = \sum_{i=0}^{k} \alpha_i u_i(x, \mu)$.

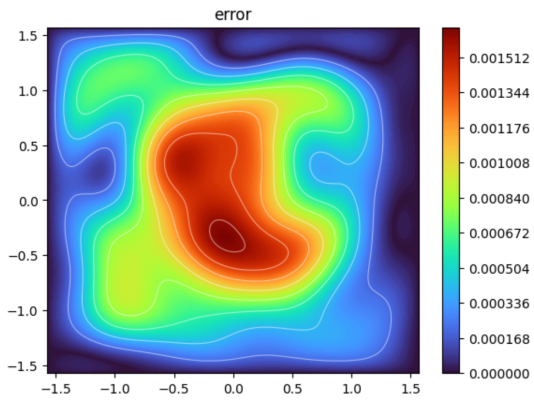
- The frequencies increase at each step so we need to use Fournier neural networks.
- **Key points**: normalize each problem to have a solution in $O(1)$ (better for training), estimate the maximal frequency if the solution to calibrate the Fourier Networks.
- Prove the Convergence of the method. Current work with Ehrlacher :).

Results on test 1

- Test1: 4D problem (2D spatial + 2 parameters).
- Classical network ($\approx$ 9k parameters). 4000 epochs. 25k points. 45 min CPU.

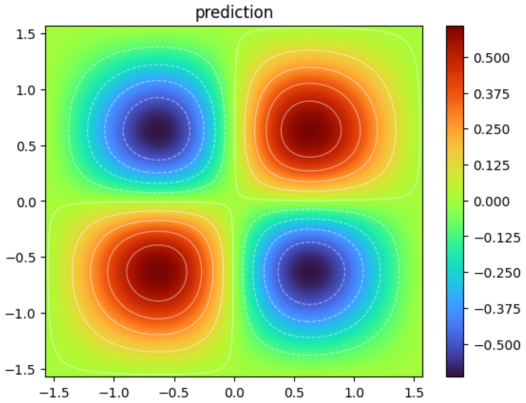


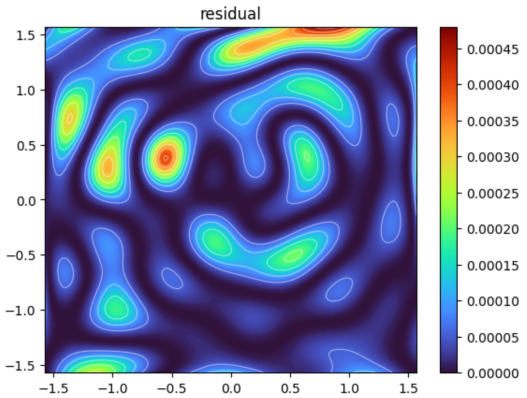


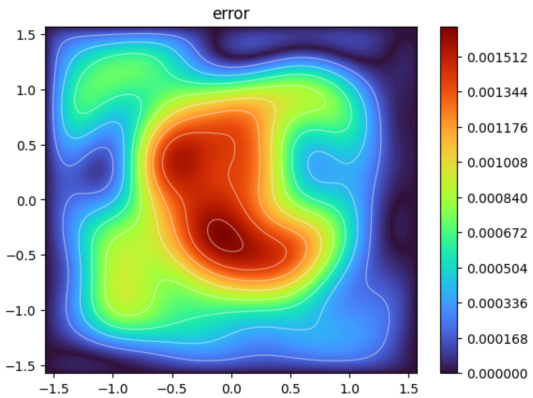


Results on test 1

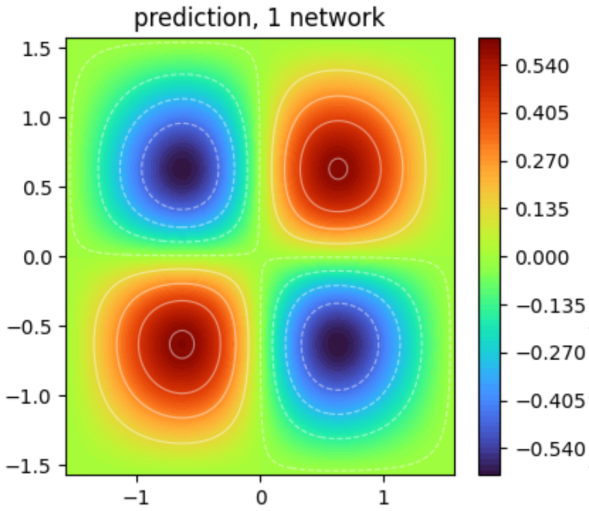
- Test1: 4D problem (2D spatial + 2 parameters).
- Classical network ($\approx$ 9k parameters). 4000 epochs. 25k points. 45 min CPU.

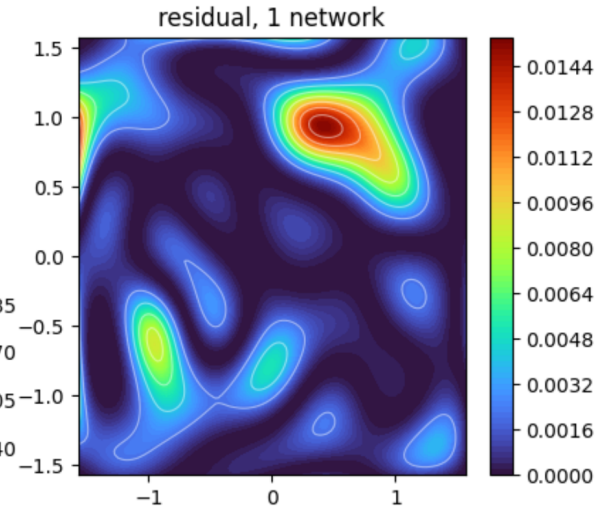


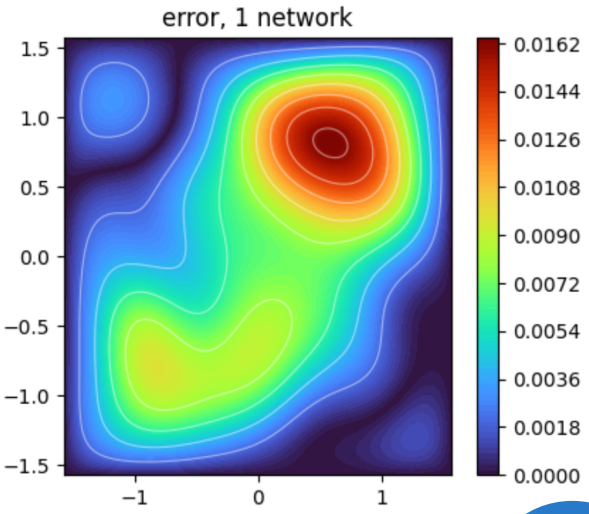




- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).

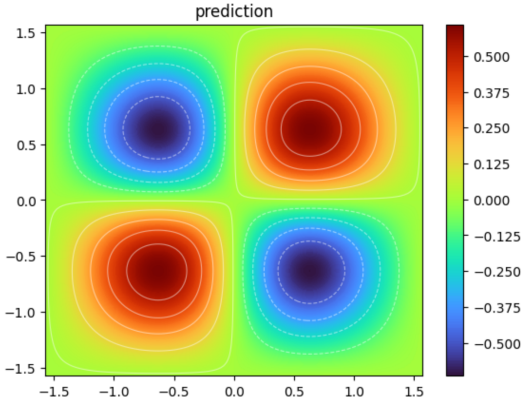


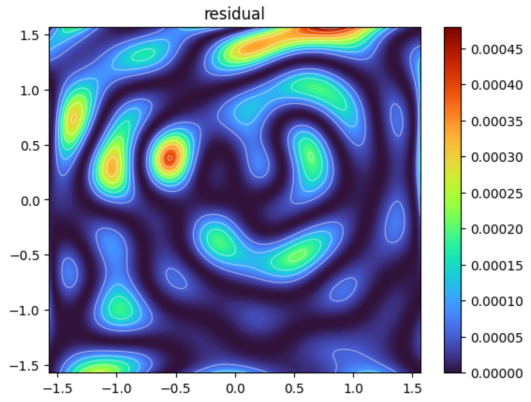


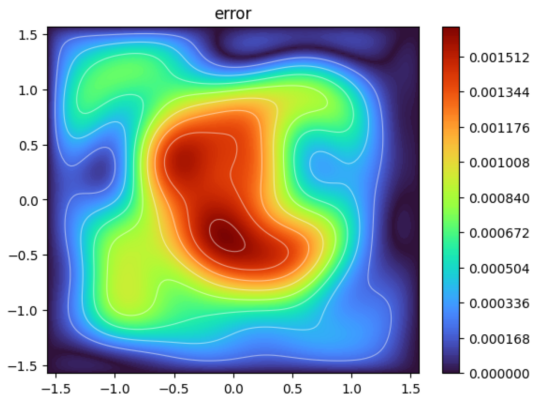


Results on test 1

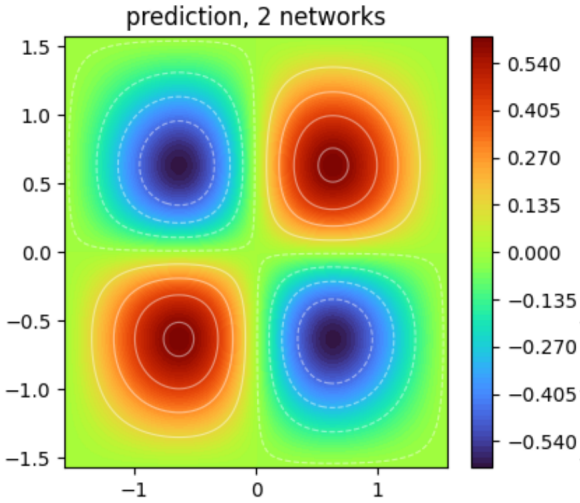
- Test1: 4D problem (2D spatial + 2 parameters).
- Classical network ($\approx$ 9k parameters). 4000 epochs. 25k points. 45 min CPU.

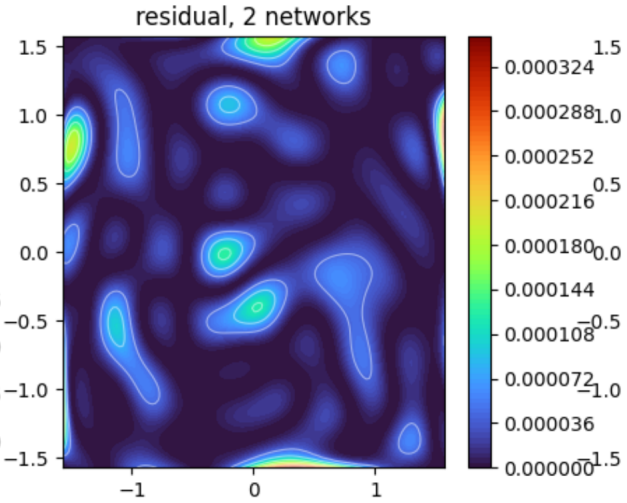


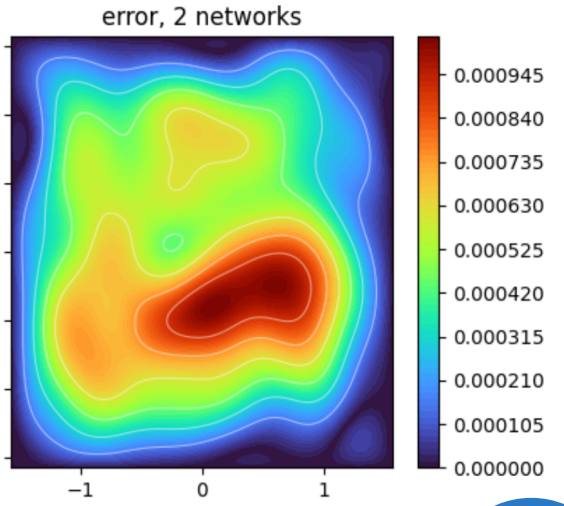




- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).

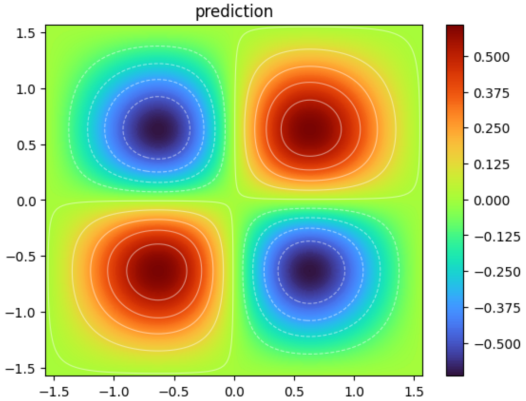


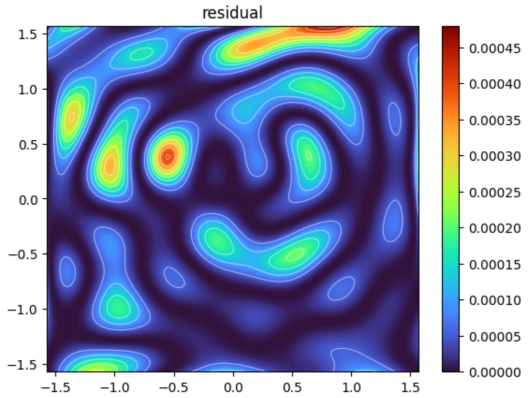


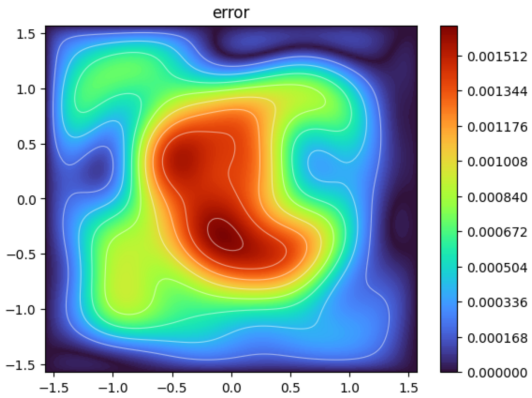


Results on test 1

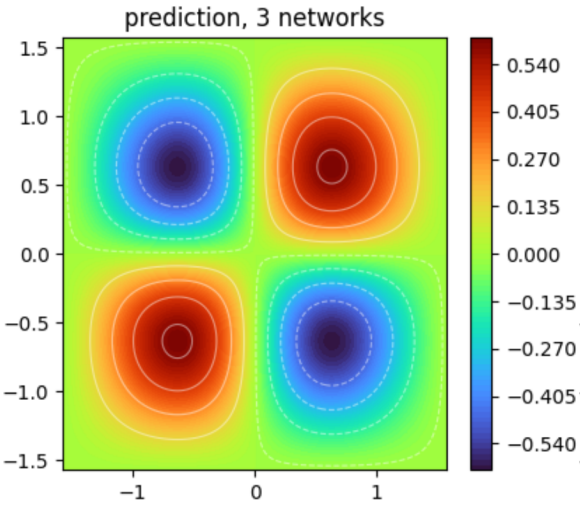
- Test1: 4D problem (2D spatial + 2 parameters).
- Classical network ($\approx$ 9k parameters). 4000 epochs. 25k points. 45 min CPU.

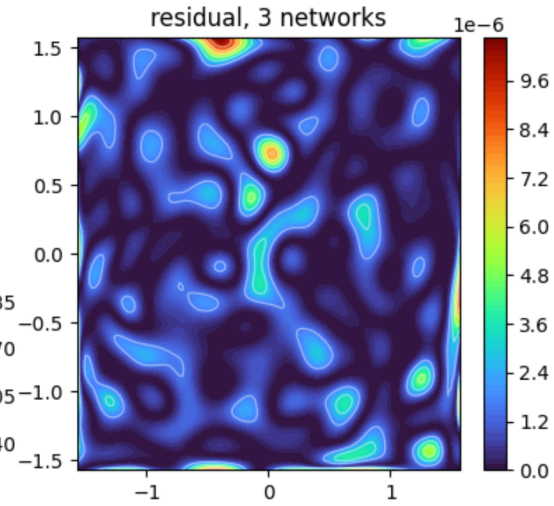


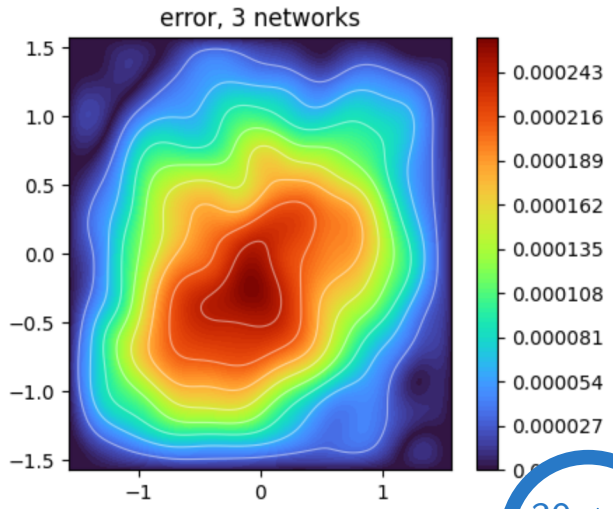




- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).

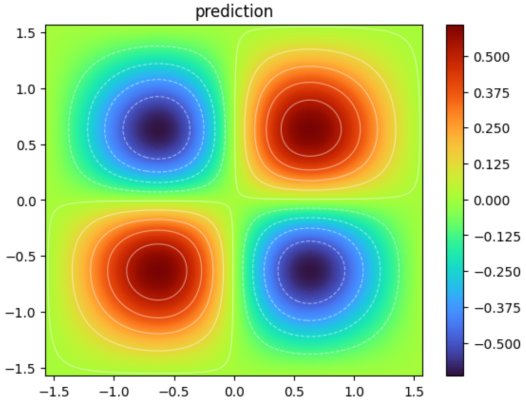


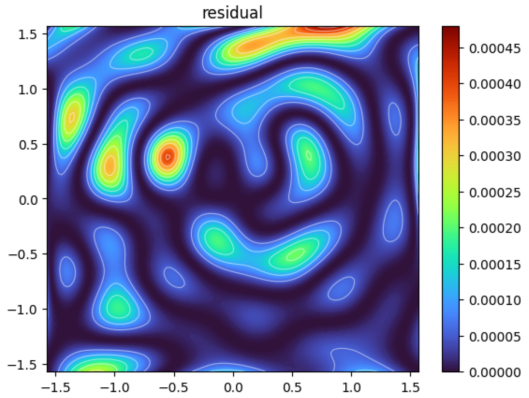


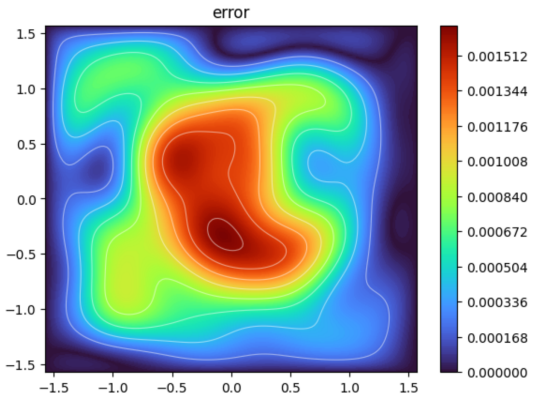


Results on test 1

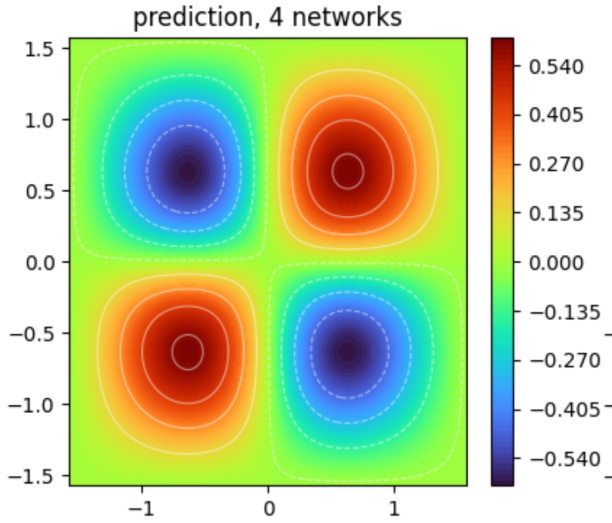
- Test1: 4D problem (2D spatial + 2 parameters).
- Classical network ($\approx$ 9k parameters). 4000 epochs. 25k points. 45 min CPU.

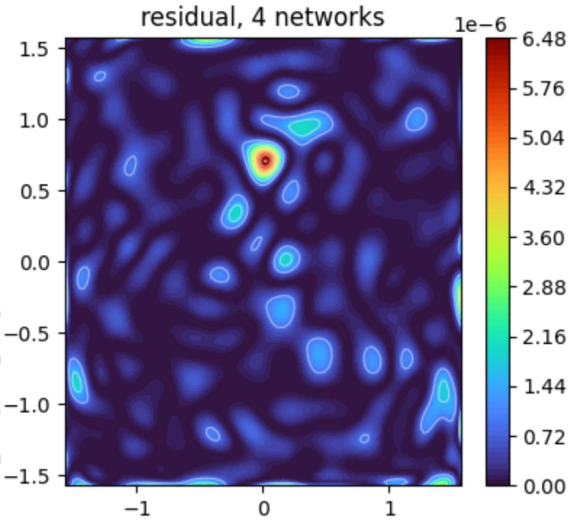


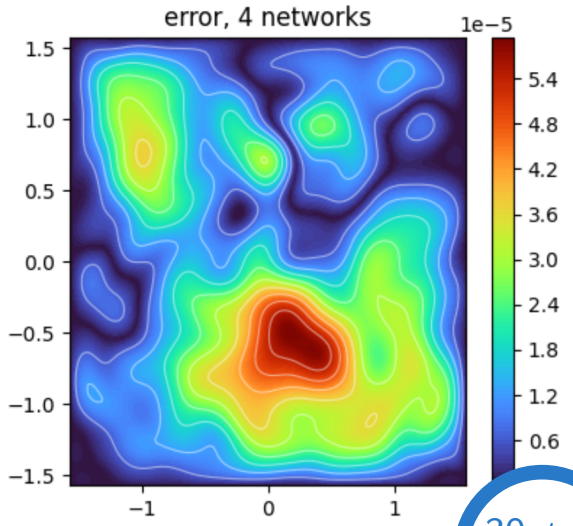




- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).







Conclusion

Conclusion

- NN-based methods (global models) are not accurate but scale well with the dimension.
- Classical methods (local models) are very accurate and provably convergent but scale very poorly with the dimension.
- We propose a convergent, simple and weakly intrusive approach where the neural network computes a coarse approximation corrected by classical methods (here, FE or DG).
- For physical/parametric dimension>3, our approach becomes very interesting in terms of CPU time and memory.

Future work

- Adapt the mesh of the correction using the residual of the solution obtained by the neural methods.
- Extend this to time-dependent problems with two approaches:
  - A PINN predicts a space/time solution later corrected by the numerical scheme.
  - A Neural Galerkin method (discrete in time, neural in space) predicts a time step, corrected by a classical method.
- **Applications**: hyperbolic-kinetic PDE (prove CV, respect physical properties), 3D reduced MHD for Tokamak and Grad-Shafranov (PhD with CulHam Fusion center next year).
- **Greedy PINNs**: extend the approach to time, transport, Hamiltonian and nonlinear problems. Prove the convergence. (Post doc for 2025 with V. Ehrlacher).

Main references

- PINNs:

PINNs *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, M. Raissi, P. Perdikaris, G.E. Karniadakis

PInnsEx *An Expert's Guide to Training Physics-informed Neural Networks*, S. Wang, S. Sankaran, H. Wang, P. Perdikaris

mevel *Multi-level neural networks for accurate solutions of boundary-value problems*, Z. Aldirany, R. Cottereau, M. Laforest, S. Prudhomme

mStage *Multi-stage neural networks: Function approximator of machine precision*, Yongji Wang, Ching-Yao Lai

GalNeu *Galerkin neurlal networks: a framework for approximating variational equations with error control*, M. Ainsworth, J. Dong

PinnsBC *Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks*, N. Sukumar and Ankit Srivastava

- Greedy Methods:

Greedy11 *Convergence of a greedy algorithm for high-dimensional convex nonlinear problems*, E Cances, V Ehrlacher, T Lelievre

- Hybrid methods:

FEhybrid *Enhanced Finite element by neural networks for elliptic problems*, H. Barucq, M. Duprez,, F. Faucher, E Franck, F. Lecourtier, V. Lleras, V. Michel-Dansac, N. Victorion. En cours de rédaction.

DGHybrid *Approximately well-balanced Discontinuous Galerkin methods using bases enriched with Physics-Informed Neural Networks*, E. Franck, V. Michel-Dansac, L. Navoret. JCP mai 2024.