Neural implicit representation for PDE problems

Emmanuel Franck[*],

February 23 2024
MIA Seminar, La Rochelle

[*] MACARON project-team, Université de Strasbourg, CNRS, Inria, IRMA, France

Outline

Introduction to Neural methods for elliptic equations

Objectives

Linear elliptic PDEs

Here we consider elliptic and linear PDEs of the form:

$$\begin{cases} L(u(\mathbf{x})) = -\nabla \cdot (A(\mathbf{x})\nabla u(\mathbf{x})) + \nabla \cdot (\beta(\mathbf{x})u(\mathbf{x})) + c(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}), & \forall \mathbf{x} \in \Omega \subset \mathbb{R}^d \\ u(\mathbf{x}) = 0, & \forall \mathbf{x} \in \partial\Omega \end{cases}$$

Numeric Vs learning

Both learning and numerical methods seek to construct function approximations. In both cases, we use parametric functions. One is constrained by the data, the other by the physical equation.

Idea

Use neural networks as parametric models in numerical methods.

Approximation space

Linear space

- Approximation space:

$$V_n = \left\{ \sum_{i=1}^{N} \theta_i \phi_i(\mathbf{x}), \quad \theta \in V \subset \mathbb{R}^n \right\}$$

- Restriction operator $\mathcal{R}$:

$$\theta^* = \min_{\theta} \int_{\Omega} \mid u(\mathbf{x}) - \langle \boldsymbol{\theta}, \boldsymbol{\Phi}(\mathbf{x}) \rangle \mid^2 dx,$$

- Solving analytically this problem we obtain:

$$M\theta = b(u)$$

$$M = \int_{\Omega} \Phi(\mathbf{x}) \otimes \Phi(\mathbf{x}) d\mathbf{x}, \quad b(u) = \int_{\Omega} u(\mathbf{x}) \Phi d\mathbf{x}$$

- Reconstruction operator $\mathcal{I}$:

$$\mathcal{I}(u) = \sum_{i=1}^{N} \theta_i^* \phi_i(x)$$

- Projection operator: $\Pi_{V_n} = \mathcal{I} \circ \mathcal{R}$

Nonlinear space

- Approximation space:

$$M_n = \{ nn_\theta(\mathbf{x}), \quad \theta \in V \subset \mathbb{R}^n \}$$

- Restriction operator $\mathcal{R}$:

$$\theta^* = \min_{\theta} \int_{\Omega} \mid u(\mathbf{x}) - nn_\theta(\mathbf{x}) \mid^2 dx,$$

- Reconstruction operator $\mathcal{I}$:

$$\mathcal{I}(u) = nn_\theta(\mathbf{x})$$

- Projection operator: $\Pi_{V_n} = \mathcal{I} \circ \mathcal{R}$
- Properties of the projection operator ?

Approximation space II

Linear space

- We choose $f_1, f_2 \in V_n$:

$$f_1(\mathbf{x}) + f_2(\mathbf{x}) = \sum_{i=1}^{N} \theta_i \phi_i(\mathbf{x}) \in V_n$$

- V_n is a vectorial space.

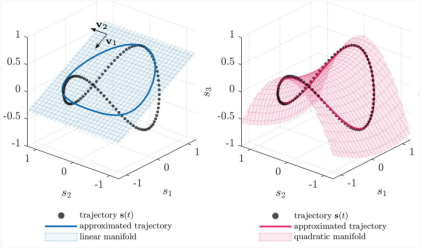
Nonlinear space

- We choose $f_1, f_2 \in M_n$:

$$f_1(\mathbf{x}) + f_2(\mathbf{x}) \notin M_n$$

- M_n is not a vectorial space but a manifold.

- Vectorial space Vs Manifold



- **Difficulty**: the projection on a manifold is not unique.

Approximation space III

Examples of linear space

- Fourier spectral functions (global):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \sin(2k\pi x)$$

- Orthogonal polynomiales spectral functions (global):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k P_k(\mathbf{x})$$

- Finite element basis (local):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi_{h,k}(\mathbf{x})$$

with $\phi_{h,k}$ piecewise polynomiales functions.

- Radial basis (local):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\epsilon \mid \mathbf{x} - \mathbf{x}_i \mid)$$

avec $\phi(r) = e^{-r^2}$, $\phi(r) = \sqrt{(1 + r^2)}$.

Approximation space III

Examples of linear space

- Fourier spectral functions (global):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \sin(2k\pi x)$$

- Orthogonal polynomiales spectral functions (global):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k P_k(\mathbf{x})$$

- Finite element basis (local):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \Phi_{h,k}(\mathbf{x})$$

with $\Phi_{h,k}$ piecewise polynomiales functions.

- Radial basis (local):

$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\epsilon \mid \mathbf{x} - \mathbf{x}_i \mid)$$

avec $\phi(r) = e^{-r^2}$, $\phi(r) = \sqrt{(1+r^2)}$.

Examples of nonlinear functions

- Tensor methods:

$$f(\mathbf{x}) = \sum_{i=1}^{r} \left(\sum_{k=1}^{n} \alpha_{i,k} \Phi_k(\mathbf{x}_1) \right) \left(\sum_{k=1}^{n} \beta_{i,k} \Phi_k(\mathbf{x}_2) \right)$$

avec $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$.

Approximation space III

Examples of linear space

- Fourier spectral functions (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \sin(2k\pi x)$$
- Orthogonal polynomiales spectral functions (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k P_k(\mathbf{x})$$
- Finite element basis (local):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi_{h,k}(\mathbf{x})$$
with $\phi_{h,k}$ piecewise polynomiales functions.
- Radial basis (local):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\epsilon \mid \mathbf{x} - \mathbf{x}_i \mid)$$
avec $\phi(r) = e^{-r^2}$, $\phi(r) = \sqrt{(1+r^2)}$.

Examples of nonlinear functions

- Tensor methods:
$$f(\mathbf{x}) = \sum_{i=1}^{r} \left(\sum_{k=1}^{n} \alpha_{i,k} \phi_k(\mathbf{x}_1) \right) \left(\sum_{k=1}^{n} \beta_{i,k} \phi_k(\mathbf{x}_2) \right)$$
avec $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$.
- Fourier spectral functions (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \sin(2\omega_k \pi x)$$
- Radiales basis (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\epsilon_k \mid \mathbf{x} - \mathbf{x}_i \mid)$$
- Anisotropic radial basis (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\mid \Sigma_k^{-1}(\mathbf{x} - \mathbf{x}_i) \mid)$$
- MLP Neural network (global):
$$f(\mathbf{x}) = nn_\theta(\mathbf{x})$$
- KAN neural Network (global):
$$f(\mathbf{x}) = kan_\theta(\mathbf{x})$$

Approximation space III

Examples of linear space

- Fourier spectral functions (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \sin(2k\pi x)$$
- Orthogonal polynomiales spectral functions (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k P_k(\mathbf{x})$$
- Finite element basis (local):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi_{h,k}(\mathbf{x})$$
with $\phi_{h,k}$ piecewise polynomiales functions.
- Radial basis (local):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\epsilon \mid \mathbf{x} - \mathbf{x}_i \mid)$$
avec $\phi(r) = e^{-r^2}$, $\phi(r) = \sqrt{(1+r^2)}$.
- Random networks (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k nn_{\theta_k}(x)$$
with θ_k are randomly chosen.

Examples of nonlinear functions

- Tensor methods:
$$f(\mathbf{x}) = \sum_{i=1}^{r} \left(\sum_{k=1}^{n} \alpha_{i,k} \phi_k(\mathbf{x}_1) \right) \left(\sum_{k=1}^{n} \beta_{i,k} \phi_k(\mathbf{x}_2) \right)$$
avec $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$.
- Fourier spectral functions (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \sin(2\omega_k \pi x)$$
- Radiales basis (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\epsilon_k \mid \mathbf{x} - \mathbf{x}_i \mid)$$
- Anisotropic radial basis (global):
$$f(\mathbf{x}) = \sum_{i=k}^{n} \alpha_k \phi(\mid \Sigma_k^{-1}(\mathbf{x} - \mathbf{x}_i) \mid)$$
- MLP Neural network (global):
$$f(\mathbf{x}) = nn_\theta(\mathbf{x})$$
- KAN neural Network (global):
$$f(\mathbf{x}) = kan_\theta(\mathbf{x})$$

Parametric model: neural network

- Neural network are one of the most popular parametric models. There are parametric models nonlinear compared to the inputs but also compared to the parameters.

Layer

A layer is a function $L : \mathbf{x} \in \mathbb{R}^{d_i} \to \mathbf{y} \in \mathbb{R}^{d_{i+1}}$ defined by

$$L_{i,i+1}(\mathbf{x}) = \sigma(A\mathbf{x} + \mathbf{b})$$

with $A \in \mathcal{M}_{d_i,d_{i+1}}(\mathbb{R})$, $\mathbf{b} \in \mathbb{R}^{d_{i+1}}$ and $\sigma()$ a nonlinear function applied component by component. We call $\sigma()$ the activation function. The matrix A and vector $\mathbf{b}$ are the trainable parameters.

Neural Network

We call neural network a parametric function $N_\theta : \mathbf{x} \in \mathbb{R}^{d_{in}} \to \mathbf{y} \in \mathbb{R}^{d_o}$ defined by

$$N_\theta(\mathbf{x}) = L_{o,n} \circ ... \circ L_{i+1,i} \circ ... \circ L_{1,in}(x)$$

with θ the set of trainable parameters.

Integration

- To calculate the restriction, we need to integrate over the domain. Integration depends on the choice of space. In many case we use quadrature formula.
- We're going to look here at the case of nonlinear spaces, in particular based on neural networks whose characteristics are:
  - Global models which not use meshes.
  - Good approximation properties in large dimension

Integration

Given the qualities of NNs, the most suitable integration method is Monte Carlo.

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

with $\mathcal{U}(\Omega)$ a uniform law on Ω. Applying the law of large numbers, we have

$$\int_\Omega \|u_\theta(\mathbf{x}) - u(\mathbf{x})\|_2^2 d\mathbf{x} \approx \frac{1}{N} \sum_{i=1}^{N} \|u_\theta(\mathbf{x}_i)) - u(\mathbf{x}_i)\|_2^2$$

Level-set function

Given an Ω domain with Γ boundary, we call a level function a ϕ function such that

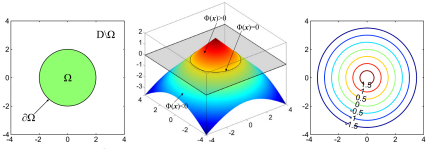
$$\phi(x) = \begin{cases} < 0, & \mathbf{x} \in \Omega \\ = 0, & \mathbf{x} \in \Gamma \\ > 0, & \mathbf{x} \in \mathbb{R}^d / \Omega \end{cases}$$

- How to sample ?
  - We draw a point randomly in $[a, d]^d$ such that Ω is included.
  - If $\phi(\mathbf{x}) < 0$ we keep the point otherwise we start again.
- No level function uniqueness. Example: the disk:

$$\phi_1(\mathbf{x}) = \sqrt{x_1^2 + x_2^2} - r, \quad \phi_1(\mathbf{x}) = x_1^2 + x_2^2 - r^2$$

- The first is called The signed distance function because it gives the distance between each point and Γ. It is a C^0 function, not a C^1 one.

- **Domains sum**: $\phi_1(\mathbf{x}) < 0$ ou $\phi_2(\mathbf{x}) < 0$
- **Domains intersection**: $\phi_1(\mathbf{x}) < 0$ et $\phi_2(\mathbf{x}) < 0$
- **Domains with holes**: $\phi_d(\mathbf{x}) < 0$ et $\phi_h(\mathbf{x}) > 0$



(a) A circular domain Ω embedded in a domain D. (b) Higher-dimensional LSF of the circular domain Ω. (c) Iso-contours of the LSF.

How compute the restriction operator ?

Linear spaces

- **Gradient computation**: analytic
- **Solving of** $\nabla J = 0$: normal equation.
  - In the linear case we have:

$$\nabla J = 0 \longleftrightarrow A\theta\mathbf{b} = 0$$

  - We solve a linear system with LU, CG, GMRES.
- **Computation of the model derivatives**: analytic

Nonlinear space

- **Gradient computation**: Automatic differentiation.
- **Solving of** $\nabla J = 0$: Gradient method and quasi-Newton method
- **Computation of the model derivatives**: Automatic differentiation.

Gradient and Newton methods

- We therefore want to determine θ^* solution of

$$\nabla_\theta \mathcal{J}(\theta^*) = \sum_{i=1}^{N} \nabla_\theta \mathcal{J}_i(\theta^*) = 0$$

  with $\mathcal{J}_i$ the local cost function (here a L^2 norm) for each sample.
- Since $\mathcal{J}$ is nonlinear we potentially have several solutions.
- The gradient is calculated by automatic differentiation.
- **Gradient method**:

$$\nabla_\theta \mathcal{J}(\theta) = 0 \iff -\eta \nabla_\theta \mathcal{J}(\theta) = 0 \iff -\eta \nabla_\theta \mathcal{J}(\theta) + \theta = \theta$$

  using fixed point method we obtain: $\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{J}(\theta_k)$.
- **Newton method**:

$$\nabla_\theta \mathcal{J}(\theta) = 0 \underbrace{\Longrightarrow}_{\text{linéarisation}} Jac(\nabla_\theta \mathcal{J}(\theta_0))(\theta - \theta_0) + \nabla_\theta \mathcal{J}(\theta_0) \approx \nabla_\theta \mathcal{J}(\theta) = 0$$

  so

$$H_\theta(J(\theta_k))(\theta_{k+1} - \theta_k) = -\nabla_\theta \mathcal{J}(\theta_k) \iff \theta_{k+1} = \theta_k - H_\theta^{-1}(\mathcal{J}(\theta_k))\nabla_\theta \mathcal{J}(\theta_k)$$

  with $H_\theta(\mathcal{J}(\theta_k))$ the Hessian of $\mathcal{J}$.
- Gauss-Newton, Levenberg-Marquardt or L-BFGS use a approximation of the Hessian.

Approximation methods

Linear spaces

- Ritz-Galerkin:

$$\theta^* = \min_{v \in V_n} (a(v,v) - f(x)v)$$

- Least square Galerkin:

$$\theta^* = \min_{v \in V_n} \int_\Omega |\, L(u) - f \,|^2$$

Nonlinear spaces

- Deep-Ritz:

$$\theta^* = \min_{v \in M_n} (a(v,v) - f(x)v)$$

- PINNs:

$$\theta^* = \min_{v \in M_n} \int_\Omega |\, L(u) - f \,|^2$$

- The idea is the same. We restrict the functions to be minimized to the approximation space.
- The difference between classical and neural methods is the approximation space.
- The choice of integral approximation and resolution follows from this.

Solving and integration

Solving

To solve the PINNs or Deep-Ritz minimization problems, we use the same methods as for calculating the restriction operator. as for calculating the restriction operator:

- classic/preconditionned Stochastic gradient methods (Adam, ResProp, etc)
- Quasi-Newton methods (L-BFGS, Leverberg-Marquardt).

- It's quite common to combine two methods. Newton/quasi-Newton methods converge slowly but are less robust to poor initialization and more expensive.
- **Usual method**: We start with a gradient method and end with a quasi-Newton algorithm.

Integration and geometry

The strategies for managing cost functions, handling complex geometries and adapting are the same as for the restriction operator.

Weak boundary conditions

- As with the usual linear methods, we can impose weak boundary conditions by penalization.
- We name $\mathcal{J}_r(u)$ the functional to be minimized (PINNs or Deep-Ritz). We call the residue of the boundary conditions $B(u) = 0$ (Dirichlet, Neumann or other).

Weak BC for neural based methods

The minimization problem becomes

$$\min_{u_\theta \in W_n} \left(\mathcal{J}r(u_\theta) + \lambda_{bc} \int_\Omega \| B(u_\theta) \|_2^2 \, d\mathbf{x} \right)$$

- **Fails**:If $\| \nabla_\theta \mathcal{J}_r(u_\theta) \|_{L^\infty} >> \| \nabla_\theta \mathcal{J}_{bc}(u_\theta) \|_{L^\infty}$ the training can learn mainly the PDE, ignore the BC and compute trivial solution.

Solution

Add an algorithm to adapt the weights of each loss function to avoid dominant gradient.

Strong boundary conditions

- In the linear methods we can impose BC in the space. We can make the same here.
- We assume that we have a level set function $\phi(\mathbf{x})$ of the domain.

Dirichlet BC

To impose $g(\mathbf{x})$ at the bc we use the space

$$M_n = \left\{ g(\mathbf{x}) + \phi(\mathbf{x}) nn_\theta(\mathbf{x}), \quad \theta \in \Theta \subset \mathbb{R}^d \right\}$$

Neumann BC

To impose $\frac{\partial u_\theta}{\partial \mathbf{n}} = h(\mathbf{x})$ at the bc we use the space

$$M_n = \left\{ \left(1 + \phi(\mathbf{x}) \frac{\partial u_\theta}{\partial \mathbf{n}} \right) nn_{\theta,1}(\mathbf{x}) - \phi(\mathbf{x}) h(\mathbf{x}) + \phi^2(\mathbf{x})(\mathbf{x}) nn_{\theta,2}(\mathbf{x}), \quad \theta_1, \theta_2 \in \Theta \subset \mathbb{R}^d \right\}$$

- We can make the same for Robin and multiple BC.
- We need a regular approximation of the signed distance function.

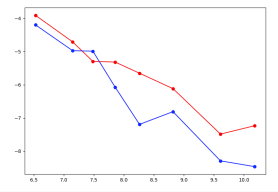
Advantages and disadvantages

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:



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

Advantage

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

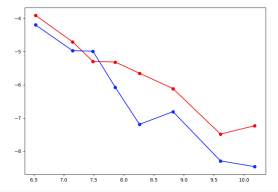
Advantages and disadvantages

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:



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	$1E^{-3}$-$4E^{-3}$

Advantage

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

Parametric problems

- In optimization, uncertainty propagation etc., we want to solve problems such as

$$L_\alpha(u(\mathbf{x})) - f(\mathbf{x}, \beta)$$

with $\mu = (\alpha, \beta)$ parameters that live in a space V_μ.

- The usual methods are too expensive in high dimension so we don't solve this problem in V_μ space.
- In general, we run simulations for different μ and build a reduced model.

Parametric neural methods

Since neural network spaces are more efficient in high dimensions, we can try to solve in V_μ space.

- In this case the restriction operator is defined by

$$\theta^* = \min_\theta \int_{V_\mu} \int_\Omega \mid u(\mathbf{x}, \mu) - nn_\theta(\mathbf{x}, \mu) \mid^2 dx,$$

- The PINNs method becomes:

$$\theta^* = \min_\theta \int_{V_\mu} \int_\Omega \mid L_\alpha(u(\mathbf{x}, \mu)) - f(\mathbf{x}, \beta) \mid^2 dx,$$

Greedy approaches

Greedy Method

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 (multistage, multlevel PINNs).

- We can write that as a greedy algorithm.
  - We consider the following submanifold approximation $\mathcal{M}_i, \quad 1 \leqslant i \leqslant d$
  - 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 $(\alpha_0, \ldots . \alpha_k)$ with a Galerkin projection or with a estimation.
    - Gives global approximation $u_h(x, \mu) = \sum_{i=0}^{k} \alpha_i u_i(x, \mu)$.

Which space ?

Interesting point: each approximation space $\mathcal{M}_i$ can be different. Examples: NN, radial basis, finite element etc.

Full NN greedy method I

Full NN approach

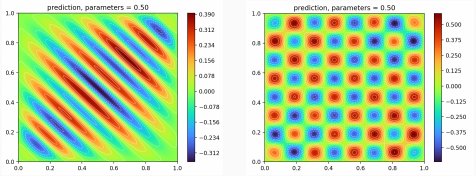
How choose the model at each step:

- One layer hidden-NN where we double the number of parameters at each step.
- Deep NN at each step with increase ability to capture 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).

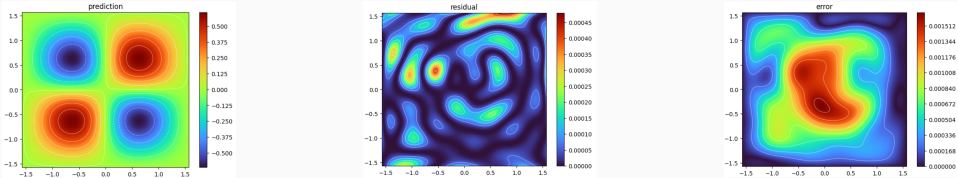
- We solve $-\Delta u = 128 \sin(8\pi x)\sin(8\pi y)$. First try (left figure): classical MLP vs Fourier NNs.



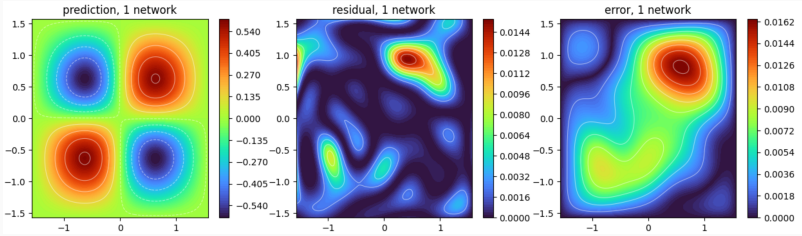
- FNN: we add Fourier features. We replace $NN_\theta(x)$ by $NN_\theta(x, \sin(2\pi k_1 x), ..., \sin(2\pi k_n x))$ with $(k_1, ...k_n)$ trainable parameters.

Full NN greedy method II

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

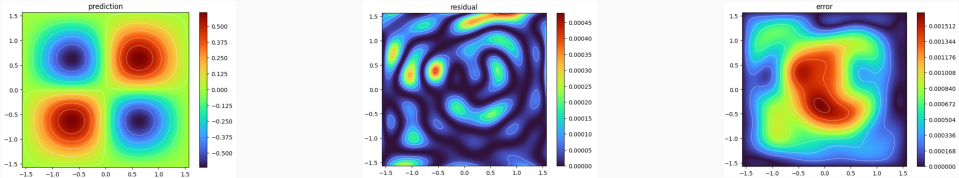


- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).

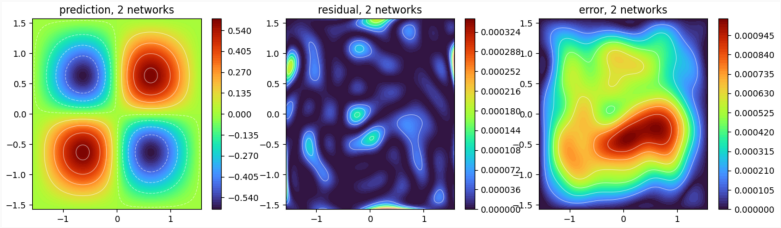


Full NN greedy method II

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

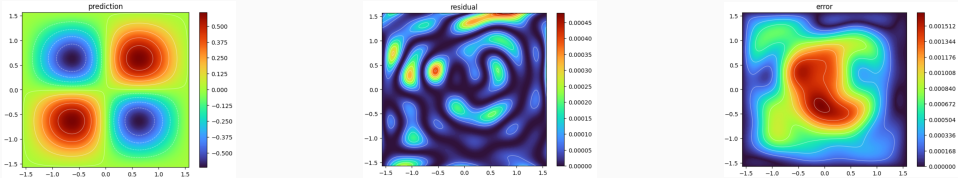


- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).

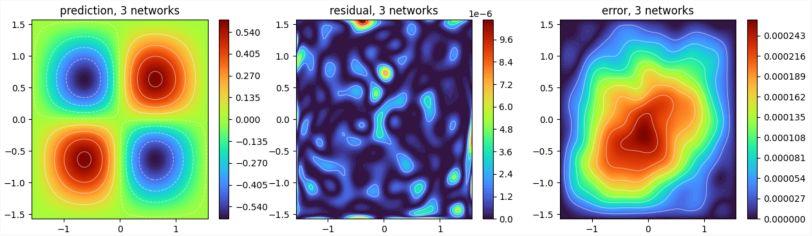


Full NN greedy method II

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

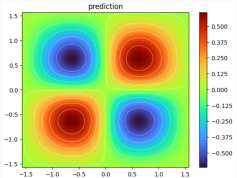


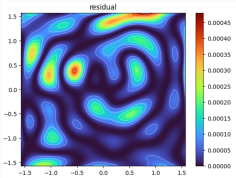
- Greedy network (4 sub-networks) (2 MLPs, 2 Fourier MLPs). 1k, 1k, 3k and 4k parameters (total: 9k). Each trained for 1000 epochs. 5k, 5k, 25k and 50k points by epoch (1h05 CPU).

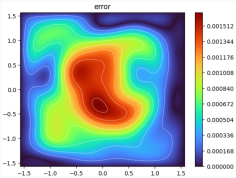


Full NN greedy method II

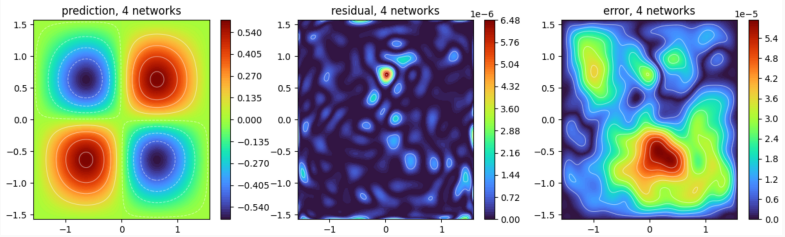
- Test: 4D problem (2D spatial + 2 parameters).
- Classical network ($\approx$ 9k parameters). 4000 epochs. 25k points. 45 min CPU.







- 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).



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**: correct the solution with a numerical method.
- Can be view as a two step Greedy method. The first with NNs on $\Omega \times V_\mu$ and the second with finite element on $\Omega \times \{\mu_1, ..., \mu_n\}$.

Additive and multiplicative formulation

- We consider the following elliptic problem:

$$\begin{cases} Lu(\mathbf{x}) = -\nabla \cdot (A(\mathbf{x})\nabla u(\mathbf{x})) + \mathbf{v} \cdot \nabla u(\mathbf{x}) + ru(\mathbf{x}) = f(\mathbf{x}), & \forall \mathbf{x} \in \Omega \\ \partial_{\mathbf{n}} u(\mathbf{x}) + \beta u(\mathbf{x}) = g(\mathbf{x}), & \forall x \in \partial\Omega \end{cases}$$

- 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:

$$u_h(\mathbf{x}) = u_\theta(\mathbf{x};\mu) + p_h(\mathbf{x}), \quad u(\mathbf{x}) = u_\theta(x;\mu)p_h(\mathbf{x}),$$

with $p_h(\mathbf{x})$ a perturbation discretized using P_k Lagrange finite element.

- For the **first approach**, we solve in practice:

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

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

Error estimates

Additive approach

- We define $I_h()$ the interpolator operator on the finite element space.
- We rewrite the Cea lemma for $u_h(\mathbf{x}) = u_\theta(\mathbf{x}) + p_h(\mathbf{x})$. We obtain

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

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

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

Key point

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

Results I

- For finite element version we use a old Fenics version (probably too slow).
- **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

- For finite element version we use a old Fenics version (probably too slow).
- **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

- For finite element version we use a old Fenics version (probably too slow).
- **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:

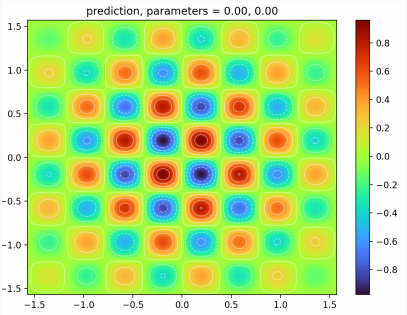
$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \Gamma. \end{cases}$$

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

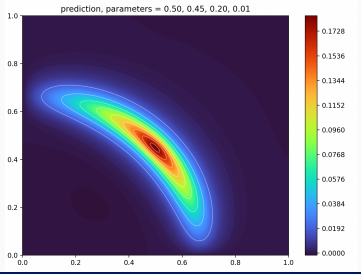
$$f(x, y) = 10 \exp(-((x1 - c1)^2 + (x2 - c2)^2)/(0.025\sigma^2))$$

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.

Shape Optimization

Problem solved

PINNs and inverse problem

One of the advantages often mentioned is their ability to easily handle inverse problems and optimal control problems, since we're already solving a nonlinear optimization problem.

- Here we consider Shape optimization problems:
- **Energy Dirichlet**:

$$\mathcal{E}(\Omega) := \inf_{u \in H_0^1(\Omega)} \frac{1}{2} \int_\Omega \left(|\nabla u|^2 - fu \right) d\mathbf{x}$$

- **Problem solved**:

$$\inf\{\mathcal{E}(\Omega), \Omega \text{ bounded open set of } \mathbb{R}^n, \text{ such that } |\Omega| = V_0\}$$

- it is equivalent to solve:

$$\inf_\Omega \left(\frac{1}{2} \int_\Omega \left(|\nabla u|^2 - fu \right) d\mathbf{x} \right), \quad \text{with the constrains} \begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega. \end{cases}$$

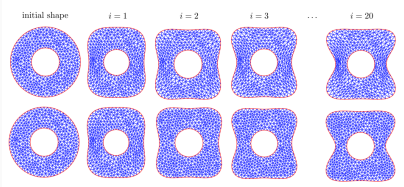
Classical method

- Here we details the classical methods to solve this problem.

One step of the algorithm

- We solve the PDE problem a Finite element or orher method on the mesh Ω_h
- We solve the adjoint PDE problem a Finite element or other method on the mesh Ω_h
- We compute the shape derivative using the primal and adjoint state.
- We use this shape derivative to move the boundary of the shape
- If the mesh becomes too degenerate we remesh.

- Picture of *Parameter-Free Shape Optimization: Various Shape Updates for Engineering Applications.*



- Immersed boundary finite element method avoid the remeshing but need to compute the shape derivative computing level set moving.

PINNs method

- Our approach:
  - We use two networks: $u_\theta(\mathbf{x}, \boldsymbol{\mu})$ for the parametric solution of the PDE and $\phi_{\theta_f}(\Omega_0)$ a diffeomorphism which deform the original space.
  - We solve:
$$\min_{\theta,\theta_f}\left(\int_{\phi_{\theta_f}(\Omega)\times \mathrm{M}}\left(\frac{1}{2}|\nabla u_\theta(\mathbf{x};\boldsymbol{\mu})|^2 - f(\mathbf{x};\boldsymbol{\mu})u_\theta(\mathbf{x};\boldsymbol{\mu})\right)\mathrm{d}\mathbf{x}\,\mathrm{d}\boldsymbol{\mu}\right)$$
    with M the parameter space.
  - By a change of variable we find a equivalent problem to solve on Ω_0. We sample on Ω_0.
- Difficulties:
  - How obtain a invertible neural network ?
  - How treat the volumes constrains ?

Advantages

One single loss function to consider.

- A penalization loss for the volume does not work.
- So we propose to impose in hard in the network: invertibility and volume preservation

Symplectic map

Key idea

Use to ϕ_{θ_f} a neural network called SympNet

- Hamiltonian ODE:

$$\frac{d\mathbf{x}}{dt} = \mathcal{J}^{-1}\nabla_{\mathbf{x}}H(\mathbf{x})$$

  with $\mathbf{x} \in \mathbb{R}^n$ and $\mathcal{J} = \begin{pmatrix} 0 & -I_N \\ I_n & 0 \end{pmatrix}$

- Flow $\phi_H(t, \mathbf{x})$ of Hamiltonian ODE:
  - **Symplectic**: $(\partial\phi_H)^t(\mathbf{x})\mathcal{J}(\partial\phi_H)(\mathbf{x}) = \mathcal{J}$
  - **Volume preservation**: $Vol(\phi_H(t, \Omega))) = Vol(\Omega)$
- If we split the Hamiltonian H into $H_1 + ... + H_K$ the flow can be approximate by

$$\phi_H(\Delta t, \mathbf{x}) \approx \phi_{H_K}(\Delta t, \mathbf{x}) \circ ... \circ \phi_{H_1}(\Delta t, \mathbf{x})$$

  and this approximation is symplectic since each subflow is associated to hamiltonian ODE and composition of symplectic map is symplectic.

Symplectic map II

Go to NNs

It look like a neural network if we learn the H_i. How assure that each subflow is symplectic ? With an exact flow. **Idea**: parametrize H_i such that we can compute exact flow.

Idea of Sympnet

If $\mathbf{x} = (\mathbf{q}, \mathbf{p})$ we choose for parametric model :

$$H_{\theta_i}(\mathbf{q}, \mathbf{p}) = T_{\theta_i}(\mathbf{q}) + K_{\theta_i}(\mathbf{p})$$

and we split agains the Hamiltonian and write the exact flo on each part.

Symplectic layer

The layer is obtained using $K_{\theta_i}(\mathbf{p}) = diag(\mathbf{a})\Sigma(K\mathbf{p} + \mathbf{b})$ et $U_{\theta_i}(\mathbf{q}) = diag(\mathbf{a})\Sigma(K\mathbf{q} + \mathbf{b})$. It gives $\phi_{La}(\mathbf{q}, \mathbf{p}) = L^2 \circ L^1$ with

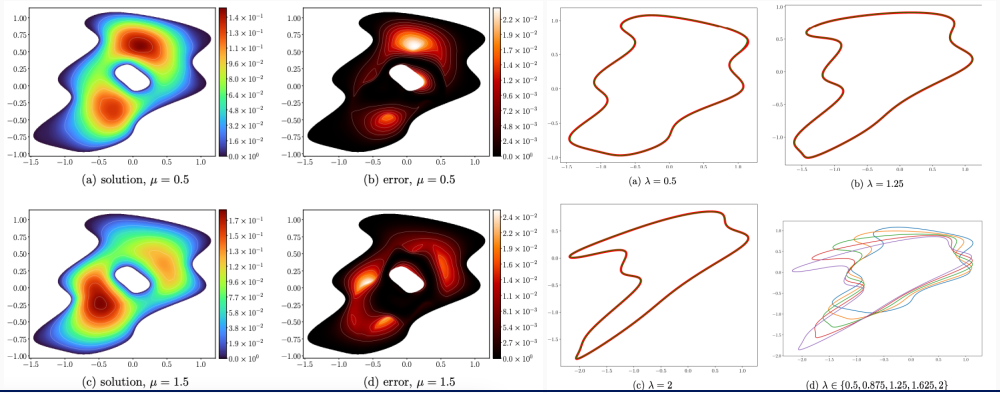
$$L^2 = \begin{pmatrix} \mathbf{q} \\ \mathbf{p} + K^t diag(\mathbf{a})\sigma(K\mathbf{q} + \mathbf{b}), \end{pmatrix} \quad L^1 = \begin{pmatrix} \mathbf{q} + K^t diag(\mathbf{a})\sigma(K\mathbf{p} + \mathbf{b}), \\ \mathbf{p} \end{pmatrix}$$

with K, $\mathbf{b}$ et $\mathbf{a}$ learnable parameters.

Results I

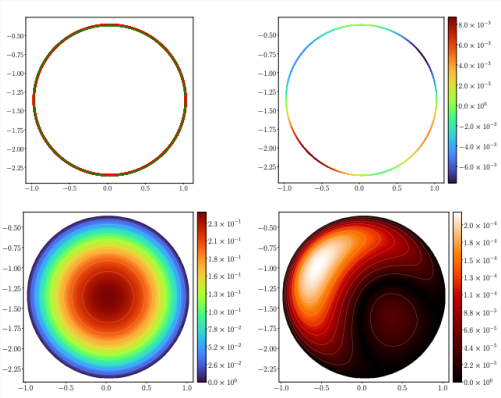
- **Left**: We solve a parametric problem $-\Delta u = f(x_1, x_2; \mu) = \exp\left(1 - \left(\frac{x_1}{\mu}\right)^2 - (\mu x_2)^2\right)$ on a domain obtain applying a analytic symplectic map:
- **Right**: We learn a parametric symplectic map:

$$\begin{cases} \mathcal{S}^1_\lambda : (x_1, x_2) \mapsto \left(x_1 - \lambda x_2^2 + 0.3\sin\left(\frac{x_2}{\lambda}\right) - 0.2\sin(8x_2), x_2\right), \\ \mathcal{S}^2_\lambda : (x_1, x_2) \mapsto (x_1, x_2 + 0.2\lambda x_1 + 0.12\cos(x_1)). \end{cases}$$



Result II

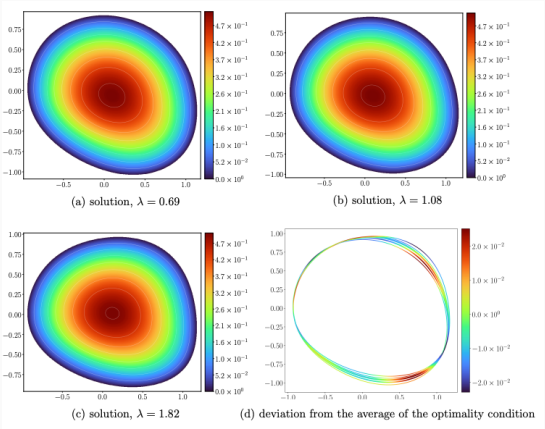
- Optimization problem with $f = 1$ and Ω_0 an ellipse.



method	FEM ($R = 100$)	FEM ($R = 250$)	FEM ($R = 500$)	GeSONN
Computational time (s)	53.3	509	3020	22.5
ℓ^2 error	7.20×10^{-2}	2.83×10^{-2}	2.21×10^{-2}	1.99×10^{-3}

Result III

- Optimization problem with $f(x, y; \lambda) = \exp\left(1 - \|\mathcal{T}_\lambda(x, y)\|^2\right)$ with T is a the previous symplectic map and Ω_0 an ellipse.



Conclusion

Conclusion

PINNs

PINNs look like a Least-Square Galerkin method on finite dimension submanifold. It is global model (no need mesh) able to tackle large dimensional smooth problems.

Greedy approaches

allows to increase the accuracy of the PINNs. Using a two step greedy method coupling PINNs and FE we can obtain a convergent method more accurate for parametric problems.

Optimization

Since we use nonlinear optimization it is a natural framework for inverse problem and control. NNs are also very useful to parametrize geometries (mapping, signed distance function) and avoid mesh in shape optimization.