Neural representation for PDEs and hybrid numerical methods

H. Barucq[3],E. Franck[12], F. Foucher[3],
V. Michel-Dansac[12], L. Navoret[12], N. Victorion[3]

02 Février 2024
Séminaire analyse numérique, CNAM

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

Outline

Numerical Methods and implicit neural representation

Parametric models

- We consider a unknown function
$$y = f(x)$$
with $x \in V \subset \mathbb{R}^d$ and $y \in W \subset \mathbb{R}^p$.

Objective

- find $f_h \in H$ an approximation of f with H a functional space.
- **Difficulty**: it is a infinite dimensional problem.

Solutions parametric models

- We consider a function f_θ composed of known elementary functions and n unknown parameters θ_i
- The problem becomes : find $f_\theta \in H_n$ an approximation of f with H_n a finite dimensional functional space.
- It is equivalent to
$$\text{Find } \theta, \text{ such that } \parallel f_\theta - f \parallel_H \leq \epsilon$$

- **Main Question**: How determinate θ ?
- Example in the following. We want approximate the temperature in a Room:
$$T(t,x), \quad x \in \Omega \in \mathbb{R}^3, t \in \mathbb{R}^+$$

ML and regression

ML regression approach

- We have data and we use it to construct the parametric model which approach our function T

- We assume that we known: $\{(x_1, t_1, T_1), ...(x_N, t_N, T_N)\}$ such that

$$T_i = T(t_i, x_i) + \epsilon_i$$

  with ϵ_i a noise.
- To approximate the temperature function we propose to approximate correctly our data examples.
- It is equivalent to solve:

$$\min_\theta \sum_{i=1}^{N} d(T_i, f_\theta(t_i, x_i))$$

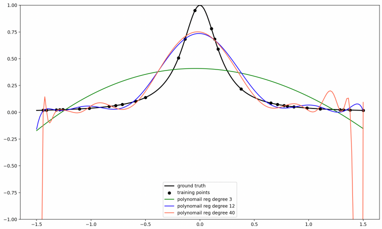
  with d a distance like euclidian norm.

Questions in ML

- Which parametric model ?
- Generalization for input outside of the data set (overfitting) ?
- Robustness to the noise ?
- How collect, process the date ?

Models and garanties

- We consider: $y = f(\mathbf{x})$ with $\mathbf{x} = (x^1, ..., x^d) \in \mathbb{R}^d$
- Models:
  - Linear model:
  $$\sum_{i=1}^{d} \theta_i x^i$$
  - Polynomial model:
  $$\sum_{i=1}^{n} \theta_i P_i(\mathbf{x})$$
  - Kernel model:
  $$\sum_{i=1}^{N} \theta_i K(\mathbf{x}, \mathbf{x}_i)$$
  with $\mathbf{x}_i$ a data and K a symmetric kernel.



- Polynomial regression of the Runge function

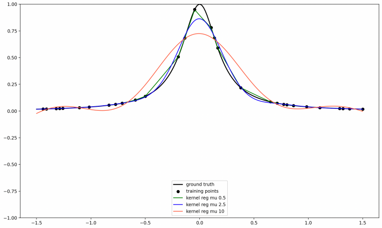
- **Garanties**: For $d = \| x - y \|_2^2$ the minimization problem is convex and admit a unique solution if you have sufficient number of data.
- For nonlinear models compared to the inputs more you have data and parameters more you will accurate.

Curse of dimensionality

The number of data needed to approximate well the function grows up exponentially with the dimension d

Models and garanties

- We consider: $y = f(\mathbf{x})$ with $\mathbf{x} = (x^1, ..., x^d) \in \mathbb{R}^d$
- Models:
  - Linear model:
$$\sum_{i=1}^{d} \theta_i x^i$$
  - Polynomial model:
$$\sum_{i=1}^{n} \theta_i P_i(\mathbf{x})$$
  - Kernel model:
$$\sum_{i=1}^{N} \theta_i K(\mathbf{x}, \mathbf{x}_i)$$
  with $\mathbf{x}_i$ a data and K a symmetric kernel.



- Kernel regression of the Runge function

- **Garanties**: For $d = \| x - y \|_2^2$ the minimization problem is convex and admit a unique solution if you have sufficient number of data.
- For nonlinear models compared to the inputs more you have data and parameters more you will accurate.

Curse of dimensionality

The number of data needed to approximate well the function grows up exponentially with the dimension d

Numerical methods

Principe of numerical methods

- **Same objective than ML**: construct a parametric model approaching T.
- no data but a strong constrain on the function: the equation

- Equation for temperature evolution:

$$\begin{cases} L_{t,x}u = \partial_t T - \Delta T = f(x) \\ T(t=0,x) = T_0(x) \\ T(x) = g \text{ on } \partial\Omega \end{cases}$$

- **Numerical method**: choose a parametric model, transform the equation/constrain on the function on a equation/constrain on the parameters.

Important: convergence

For numerical methods, we want that $\| f_\theta - f \|_h \rightarrow 0$, when , $n \rightarrow \infty$ with n the number of parameters (call degrees of freedom).

- For the three next slides, i consider only a spatial problem like $-\Delta T = f(x)$

Parametric models

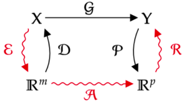
- In all the classical numerical method we choose: $f_\theta = \sum_{i=1}^n \theta_i \phi_i(x)$
- How construct ϕ_i ?

Polynomial Lagrange interpolation

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

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

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



First step: choose a parametric function

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

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

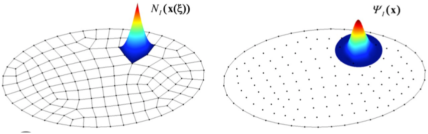
This is a piecewise polynomial representation.

Classical numerical methods, encoder and decoder

Parametric model for all numerical methods;

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

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



How determinate the degree of freedom

General method

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

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

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

  - Galerkin projection:

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

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

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

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

$$A\theta = b$$

- Second approach: Least square Galerkin projection

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

Time case

Space time methods

We use the parametric model:

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

- The time equation have no equivalent minimization form so we use the Least square Galerkin projection.

Space methods

We use the parametric model:

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

- To obtain the parameters we must find a way to write a ODE on these parameters.
- If we plug the parametric model in the equation we have

$$\partial_t T_\theta(t, x) - \Delta T_\theta(t, x) = f(x), \Longleftrightarrow (\nabla_\theta T_\theta) \frac{d\theta(t)}{dt} - \Delta T_\theta(t, x) = f(x)$$

- Not possible to invert $(\nabla_\theta T_\theta) = \Phi = (\phi_1(x), ...\phi_n(x))^t$. So we solve

$$\frac{d\theta(t)}{dt} = \min_\eta \int_\Omega \mid \Phi \cdot \eta - \Delta T_\theta(t, x) - f(x) \mid^2$$

- The problem is quadratic we can compute the ODE on $\theta(t)$.

Garanties

Essential point

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

Convergence

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

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

Curse of dimensionality

For mesh based approaches

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

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

Neural representation in ML and numerics

Deep ML, nonlinear model and manifold

Key point

All the parametric models introduced for ML or numerical methods are linear compared to the parameters and gives finite dimension function vectorial space

Deep learning

The rupture associated to the deep learning is to use massively nonlinear compared to the parameters which gives finite dimension function manifold

Projection on manifold

How project on manifold ? Not uniqueness ? The convex optimisation problem are replaced by non-convex problem. So there is less guaranties on the results.

Nonlinear models

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

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

- Neural networks (NN).

Layer

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

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

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

Neural network

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

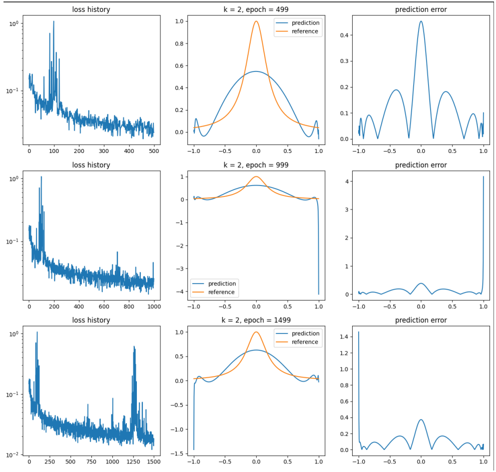
$$f_\theta(\mathbf{x}) = L_n \circ \ldots \circ L_1(\mathbf{x})$$

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

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

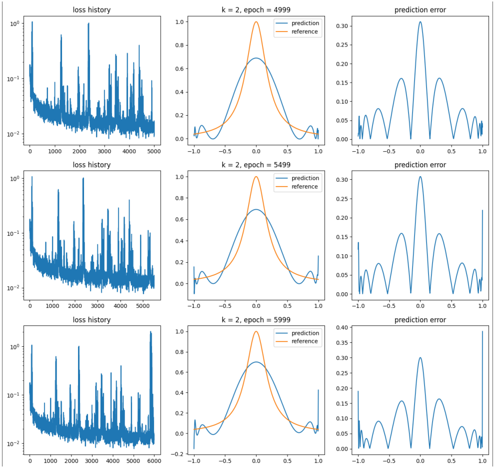
NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



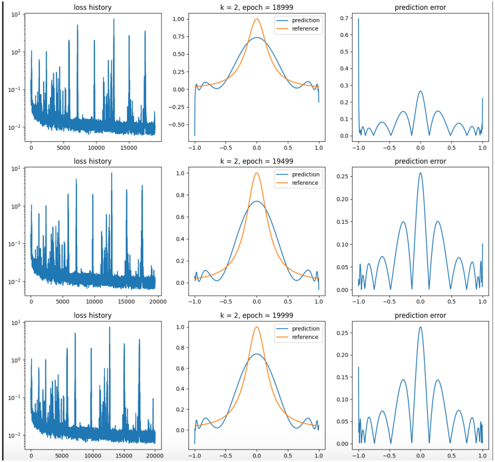
NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



NN vs Polynomial

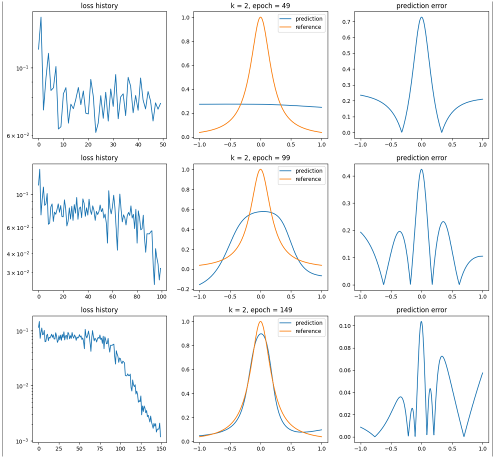
- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



- The polynomial model tends to oscillate in the over parameterized regime. Problematic for overfitting.

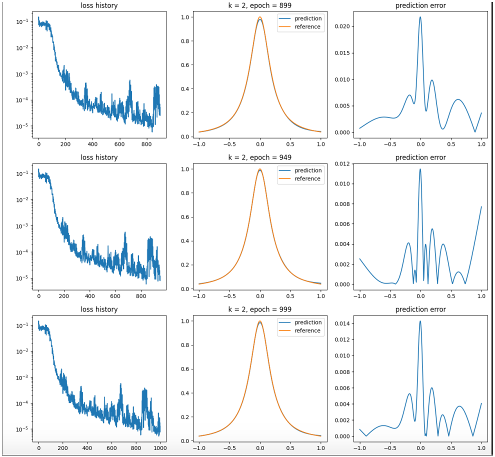
NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



NN vs Polynomial

- We compare over-parametrized NN and polynomial regression on the Runge function.
- **Regression**: 120 data and approximately 800 parameters in each model.



- The ANN generates very smooth/low frequency approximations.
- It is related to the spectral bias. The low frequencies are learned before the high frequencies.
- Seems very helpful to use it for global and high dimensional representation.

Space-time approach: PINNs I

Neural methods

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

Idea of PINNs

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

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

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

Which projection

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

Space-time approach: PINNs II

- We define the residual of the PDE:

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

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

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

with

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

and

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

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

Space-time approach: PINNs II

- We define the residual of the PDE:

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

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

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

with

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

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

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

- If these residuals are all equal to zero, then $u_{nn}(t,x;\theta)$ is a solution of the PDE.
- To complete the determination of the method, we need a way to compute the integrals. In practice we use Monte Carlo.

- **Important point**: the derivatives are computed exactly using automatic differentiation tools and back propagation. Valid for any decoder proposed.

PINNs for parametric PDEs

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

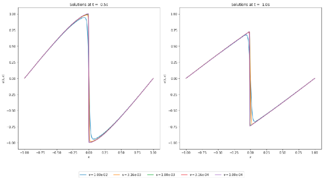
New Optimization problem for PINNs

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

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

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

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

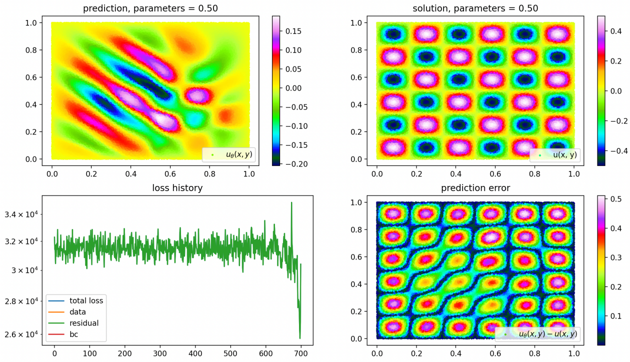


- Training for $\mu = 10^{-4}$: 2h. Training for the full viscosity subset: 2h.

- **Spectral bias of MLP**: the MLP learn firstly the low frequencies and after the high frequencies (with difficulty)
- To solve this problem for PINNs we add Fourier features. We replace

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

  with $(k_1, ...k_n)$ trainable parameters.
- Example for $-\Delta u = \sin(6\pi x)$



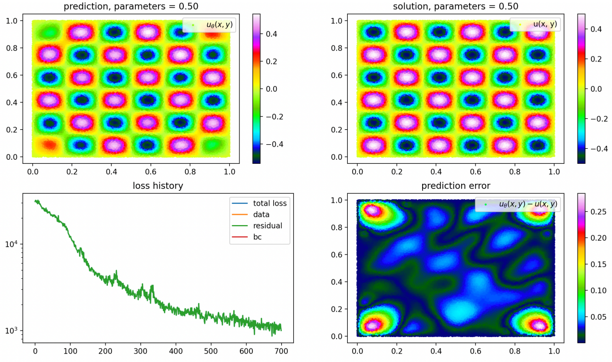
- MLP classic

Interesting tools I

- **Spectral bias of MLP**: the MLP learn firstly the low frequencies and after the high frequencies (with difficulty)
- To solve this problem for PINNs we add Fourier features. We replace

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

  with $(k_1, ...k_n)$ trainable parameters.
- Example for $-\Delta u = \sin(6\pi x)$



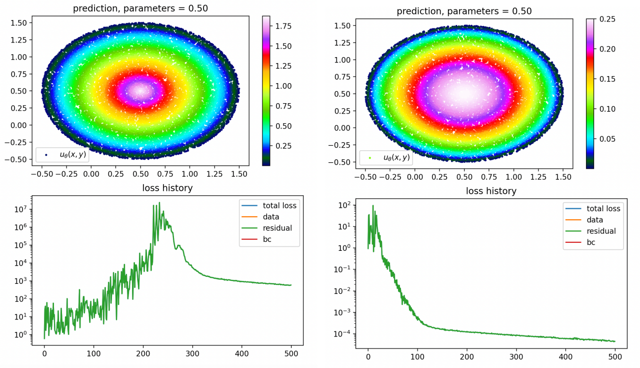
- MLP with Fourier features

Interesting tools II

- How treat the general geometry ?
- We define the model by a level set $\phi(x)$ which satisfy

$$\phi(x) = 0, x \in \partial\Omega, \quad \phi(x) < 0, x \in \Omega, \quad \phi(x) > 0, x \in \mathbb{R}^n/\Omega,$$

- Sample is easy in this case.
- Restriction on ϕ ? Classic level set: the signed distance function.



- Left: exact distance function, right: smooth levelset.
- **Solution**: approximated distance function, learned signed distance function.
- How learn a distance function ? A PINNs which approximate the Eikonal equation.

Spatial approach: Neural Galerkin I

- We solve the following PDE:

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

- Classical representation: $u(t,x) = \sum_{i=1}^{N} \theta_i(t)\phi_i(x)$
- Deep representation: $u(t,x) = u_{nn}(x;\theta(t))$ with u_{nn} a neural network, with parameters $\theta(t)$, taking x as input.
- We want that:

$$F(u_{nn}(x;\theta(t))) = \partial_t u_{nn}(x;\boldsymbol{\theta(t)}) = \left\langle \nabla_\theta u_{nn}(x;\boldsymbol{\theta}), \frac{d\theta(t)}{dt} \right\rangle$$

- How to find an equation for $\frac{d\theta(t)}{dt}$?
- We solve the minimization problem:

$$\frac{d\theta(t)}{dt} = \arg\min_{\boldsymbol{\eta}} J(\boldsymbol{\eta}) = \arg\min_{\boldsymbol{\eta}} \int_\Omega | \langle \nabla_\theta u_{nn}(x;\theta), \boldsymbol{\eta} \rangle - F(u_{nn}(x;\theta(t)))|^2 dx.$$

- The solution is given by

$$\boxed{M(\theta(t))\frac{d\theta(t)}{dt} = F(x,\theta(t))}$$

with

$$M(\theta(t)) = \int_\Omega \nabla_\theta u_{nn}(x;\theta) \otimes \nabla_\theta u_{nn}(x;\theta) dx, \quad F(x,\theta(t)) = \int_\Omega \nabla_\theta u_{nn}(x;\theta) F(u_{nn}(x;\theta)) dx.$$

Spatial approach: Neural Galerkin II

- How to estimate $M(\theta(t))$ and $F(x,\theta(t))$?
- **Firstly**: we need to differentiate the network with respect to θ and to x (in the function F). This can easily be done with automatic differentiation.
- **Secondly**: How to compute the integrals? Monte Carlo approach.
- So, we use:

$$M(\theta(t)) \approx \sum_{i=1}^{N} \nabla_\theta u_{nn}(x_i;\theta) \otimes \nabla_\theta u_{nn}(x_i;\theta)$$

  and the same for $F(x,\theta(t))$.
- **Summary**: we obtain an ODE in time (as usual) and a mesh-less method in space.
- Like in the case of PINNs, we can apply this framework to parametric PDEs and larger dimensions.
- We solve the following PDE:

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

- Deep representation: $u(t,x,\mu) = u_{nn}(x,\mu;\theta(t))$
- The solution is given by

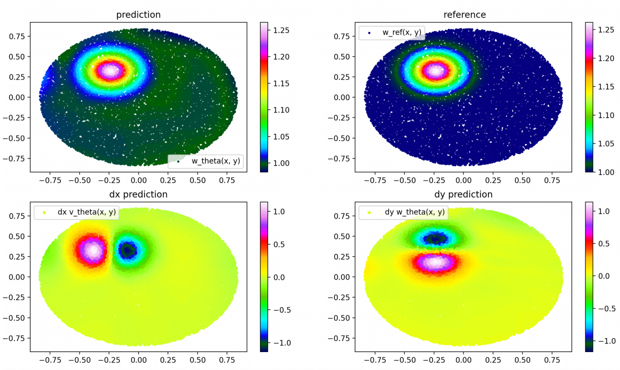
$$\boxed{M(\theta(t))\frac{d\theta(t)}{dt} = F(x,\theta(t),\mu)}$$

  with

$$M(\theta(t)) = \int_{V_\mu}\int_\Omega \nabla_\theta u_{nn}(x,\mu;\theta) \otimes \nabla_\theta u_{nn}(x,\mu;\theta) dx d\mu.$$

Spatial approach: Neural Galerkin III

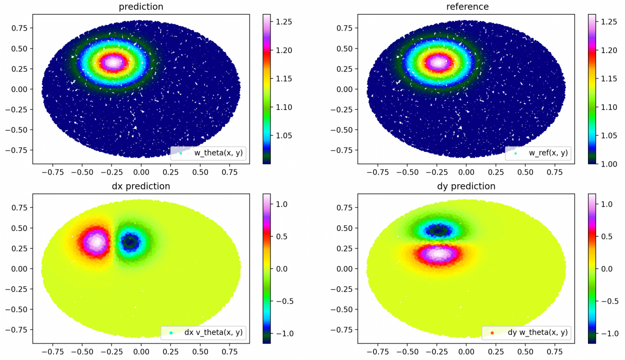
- We solve the advection-diffusion equation $\partial_t \rho + \boldsymbol{a} \cdot \nabla \rho = D \Delta \rho$ with a Gaussian function as initial condition.
- Case 1: with a neural network (2200 DOF)



- 5 minutes on CPU, MSE error around 0.0045.

Spatial approach: Neural Galerkin III

- We solve the advection-diffusion equation $\partial_t \rho + \boldsymbol{a} \cdot \nabla \rho = D \Delta \rho$ with a Gaussian function as initial condition.
- Case 2: with a Gaussian mixture (one Gaussian):



- 5 sec on CPU. MSE around 1.0^{-6}. Decoder perfect to represent this test case.

Hybrid numerical methods

Hybrid predictor-corrector methods

Hybrid methods

In this context, hybrid methods combine classical numerical methods and numerical methods based on Implicit Neural representation (IRM).

Objectives

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

General Idea

- **Offline 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.

Predictor-Corrector: using PINNs in a FE method

- 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 of the solution given by a parametric PINN $u_\theta(x)$
- We propose the following corrections of the finite element basis functions:

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

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

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

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

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

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

Theory for hybrid EF

- **Approach one**: 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

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

- **Approach two**: $u_h(x) = u_\theta(x) p_h(x)$. We use a modified interpolator:

$$I_{mod,h}(f) = \sum_{i=1}^{N} \frac{f(x_i)}{u_\theta(x_i)} \phi_i(x) u_\theta(x)$$

using $I_{mod,f}(f) = I_h(\frac{f}{u_\theta}) u_\theta(x)$, the Cea lemma and interpolation estimate we have:

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

Key point

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

EF for elliptic problems

- First test:

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

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

method:	average gain	variance gain
additive prior with PINNs	273	13000
Multiplicative prior $M = 3$ with PINNs	92	4000
Multiplicative prior $M = 100$ with PINNs	272	13000
additive prior with NN	15	18
Multiplicative prior $M = 3$ with NN	11	17.5
Multiplicative prior $M = 100$ with NN	15	18

- The PINN is trained with the physical loss, the NN with only data, no physics.
- The NN is able to better learn the solution itself, but the approximation of derivatives is less accurate than with the PINN.

EF for elliptic problems

- Second test:

$$v\partial_x u - \frac{1}{P_e}\partial_{xx} u = r$$

We train with $r \in [1,2]$, $Pe \in [10,100]$. We test with $(r, Pe) = (1.2, 40)$ and $(r, Pe) = (1.5, 90)$

Case 1	Classical FE		Additive prior			Multiplicative prior		
	error	order	error	order	gain	error	order	gain
10	$1.07e^{-1}$	–	$2.70e^{-3}$	–	40	$2.29e^{-4}$	–	467
20	$3.36e^{-2}$	1.97	$8.00e^{-4}$	1.76	42	$9.06e^{-5}$	1.93	371
40	$9.09e^{-3}$	1.89	$2.01e^{-4}$	2.00	45	$2.63e^{-5}$	1.97	345
80	$2.32e^{-3}$	1.97	$5.01e^{-5}$	1.99	46	$6.37e^{-6}$	1.99	365
160	$5.82e^{-4}$	1.99	$1.30e^{-6}$	1.97	45	$1.77e^{-6}$	2.0	289

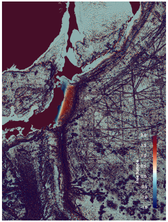
Case 2	Classic		additive prior			Multiplicative prior		
	error	order	error	order	gain	error	order	gain
10	$2.65e^{-1}$	–	$1.51e^{-1}$	–	1.7	$9.33e^{-4}$	–	284
20	$1.06e^{-1}$	1.32	$6.04e^{-2}$	1.33	1.7	$3.84e^{-4}$	1.28	276
40	$3.46e^{-2}$	1.62	$1.96e^{-2}$	1.62	1.8	$1.13e^{-4}$	1.76	305
80	$9.50e^{-3}$	1.86	$5.32e^{-3}$	1.87	1.8	$3.26e^{-5}$	1.80	291
160	$2.43e^{-3}$	1.86	$2.43e^{-3}$	1.86	1.8	$8.67e^{-6}$	1.91	280

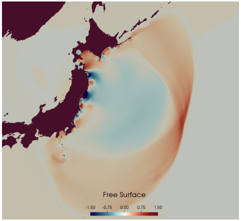
Hyperbolic systems with source terms

- 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 with a high-accuracy than the scheme the steady state.
- Building exact WB schemes is difficult for some equilibria, or for 2D flows.





Idea

Compute offline a family of equilibria with parametric PINNs (or NOs) and plug the equilibrium in the DG basis to obtain a more accurate scheme around steady states.

Theory for hybrid DG

- Theory for the scalar case.
- 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 to take as 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 projector for V2

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

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

- Adding a stability estimate, we can also prove the convergence. **Important**: The prior must give a good approximation of the m^{th} derivative.

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

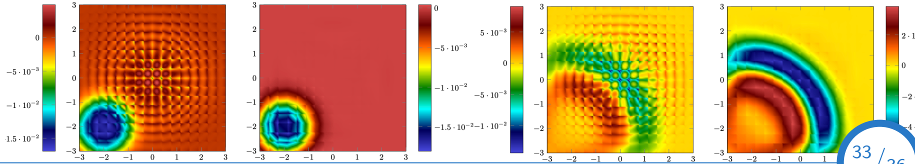
- Training takes about 10 minutes on an old GPU,with **no data**, only the PINN loss.
- We take a quadrature of degree $n_Q = n_G + 1$ (sometimes, more accurate quadrature formulas are needed).
- Barotropic case:

	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:



Conclusion

Conclusion

Short conclusion

Using nonlinear implicit representations, we proposed new numerical/reduced modeling methods whose advantages/drawbacks are very different to those of classical approaches. We will continue to investigate hybrid approaches.

Macaron

- Our Inria team MACARON becomes specialize in the hybridation between ML and numerical methods for PDEs.
- We regularly have PhD, post-doc and even permanent positions open on these subjects. If you are interested, contact us :)

Main references

- 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
  - *An Expert's Guide to Training Physics-informed Neural Networks*, S. Wang, S. Sankaran, H. Wang, P. Perdikaris
  - *Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs*, S. Mishra, R. Molinaro
- Neural Galerkin:
  - *Neural Galerkin Scheme with Active Learning for High-Dimensional Evolution Equations*, J. Bruna, B. Peherstorfer, E. Vanden-Eijnden
  - *A Stable and Scalable Method for Solving Initial Value PDEs with Neural Networks*, M. Finzi, A. Potapczynski, M. Choptuik, A. Gordon Wilson
- Neural Operator:
  - *Fourier Neural Operator for Parametric Partial Differential Equations*, Z.i Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar
  - *Neural Operator: Learning Maps Between Function Spaces*, N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, A. Anandkumar
  - *MOD-Net: A Machine Learning Approach via Model-Operator-Data Network for Solving PDE*, L. Zhang, T. Luo, Y. Zhang, Weinan E, Z. Xu, Z. Ma
- Hybrid methods:
  - *Enhanced Finite element by neural networks for elliptic problems*, H. Barucq, E Franck, F. Faucher, N. Victorion. En cours de rédaction
  - *Approximately well-balanced Discontinuous Galerkin methods using bases enriched with Physics-Informed Neural Networks*, E. Franck, V. Michel-Dansac, L. Navoret. Arxiv preprint.