SCIMBA: SCIentific Machine learning liBrAry

E. Franck

AG Exama, PEPR NUMPEX

01/2026

1. Scimba

Scimba

- **Scimba** is an OpenSource library integrating learning algorithms to solve PDE-type problems.
- **Two axes**:
  - Numerical methods and construction of reduced models based on neural networks.
  - Hybrid methods combining classical methods and learning.

Objectives: Propose approaches that are both accurate and faster than classical methods for solving high-dimensional PDEs and parametric PDEs.

Remark: Scimba development is primarily conducted within the framework of projects: Exama (PEPR Numpex), PDE-IA (PEPR IA) and the "scimba d'été" ADT (INRIA).

Who? Where? How?

- **Main developers**: E. Franck, V. Michel-Dansac and R. Imbach (Inria MACARON), M. Boileau (CNRS). More to come, we hope.

- **Other developers**:
  - N. Paillez (PhD Unistra PDE-IA), F. Lecourtier (PhD Inria MIMESIS), C. Schnoebelen (PhD Unistra), A. Belières (PhD Unistra Numpex), V. Italiano (PhD Unistra Numpex-Enact).
- **Users/collaborators**:
  - **Plasma**: V. Grandgirard (CEA), M. Campos Pinto and V. Fournet (IPP Garching), S. Pamela (UKAEA).
  - **Astrophysics**: P. Ocvrik (Obs Strasbourg)
  - **Waves and DD**: D. Hrebenshchykova and S. Lanteri (INRIA Atlantis), H. Barucq and F. Faucher (INRIA Makutu), V. Dolean (TU Eindhoven), A. Heinlein (TU Delft).
  - **Coupling**: C. Prud'homme (Unistra) and J. Aghili (Unistra).

Remark (Information):
- **Main version**: Pytorch + Pytest + Doc Sphinx
- **Jax version under development** (several algorithms already available).

1.1. So far

Nonlinear approximation spaces

Remark: Most numerical methods use linear approximation spaces. The first objective of **Scimba** is to propose **nonlinear spaces**.

- Approximation space

$$u_\theta(x) = \sum_{i=1}^{N} \alpha_i \varphi(x, \beta_i)$$

with $\theta = (\alpha, \beta)$.

- Projection by collocation

$$\theta^* = \arg\min_\theta \int_\Omega |u_\theta(x) - u(x)|^2 \, dx \approx \sum_{j=1}^{M} |u_\theta(x_j) - u(x_j)|^2$$

- In practice:
  - linear case (β fixed): least squares and matrix inversion
  - nonlinear case (β variable): nonlinear optimization (gradient descent, Adam, ...)
  - hybrid case: alternating between the two.

Example (possible spaces): Neural networks, kernel methods (with learnable kernels), nonlinear spectral methods)

PINNs and collocation method II

- How we solve the PDE: $\mathcal{R}(u) = f(x)$

Definition (Residual projection): We solve

$$\theta^* = \arg\min_{\theta} \int_{\Omega} |\mathcal{R}(u_\theta(x)) - f(x)\ |^2\ dx \approx \sum_{j=1}^{M} |\mathcal{R}\big(u_\theta\big(x_j\big)\big) - f\big(x_j\big)\ |^2$$

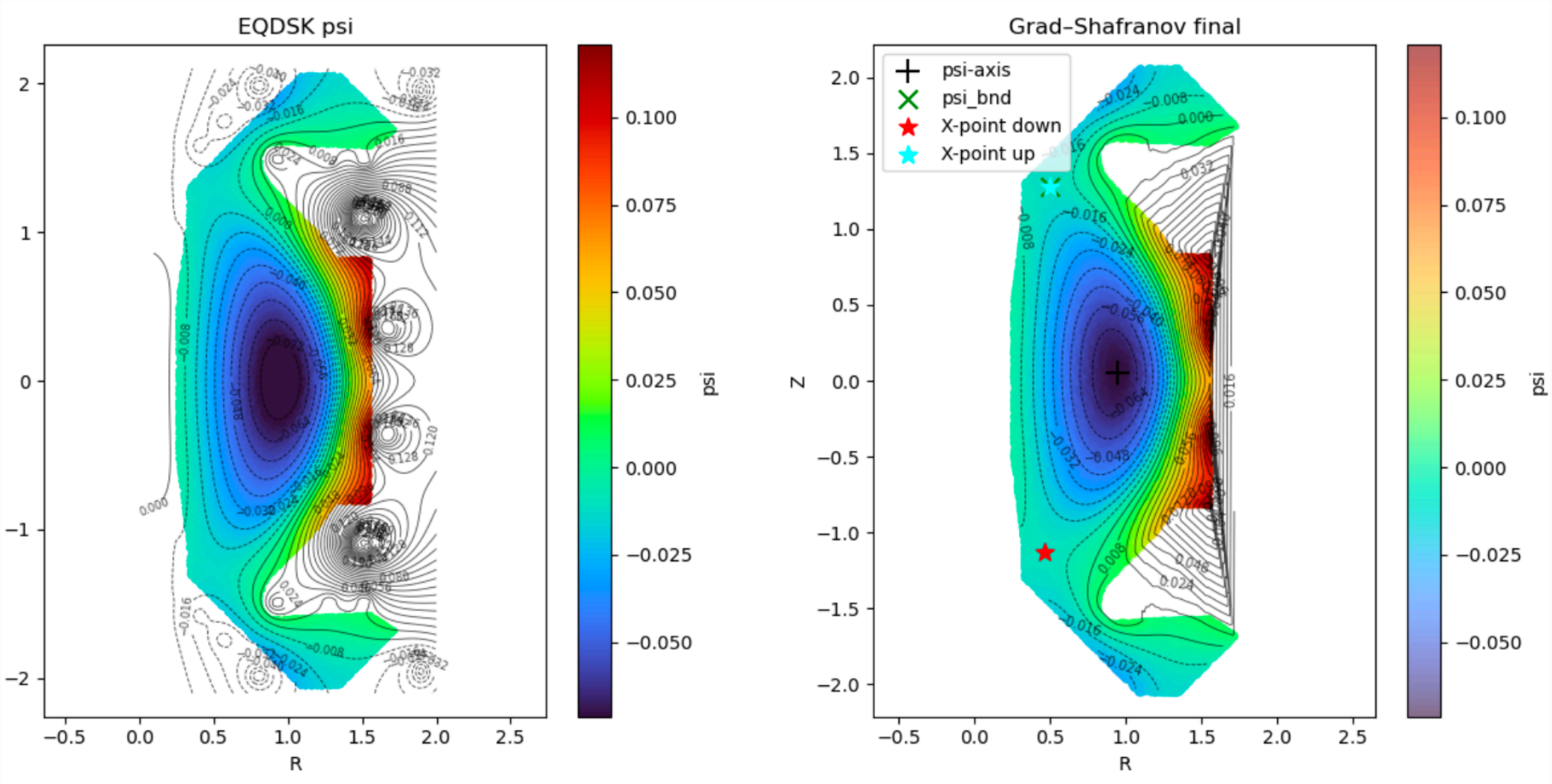
- If the space is linear, we fall back to classical collocation methods.
- If the space is nonlinear, we obtain **PINNs** type methods.

Remark: The code allows solving **parametric** problems in the domain $\Omega \times V$. Domains are sampled by **level set function** or **mapping**.

Result: To obtain good accuracy, we use quasi-Newton type optimizers. In Scimba: **Natural gradient** (requires linear algebra). SsBroyden also seems very good.

Examples

- Fixed boundary Grad Shafranov equilibrium for tokamak:



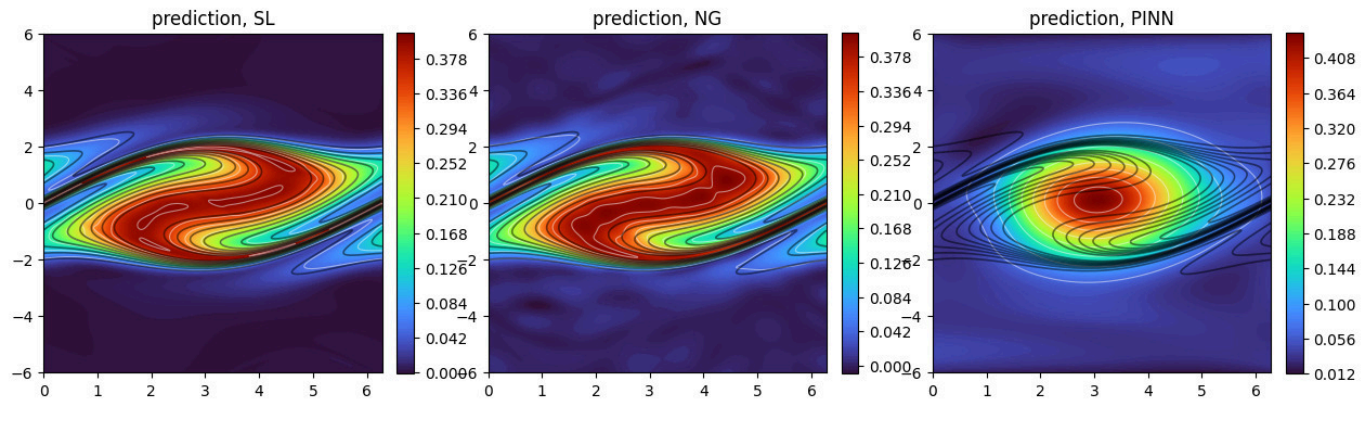
So far

Sequential time methods

- **PINNs for time-dependent problems**: one optimization. Equivalent to "space-time" approximations.
- **Sequential methods**:
  - we approximate the solution at each time step t_n by a linear (classical) or nonlinear space.
  - we solve optimizations at each time step.

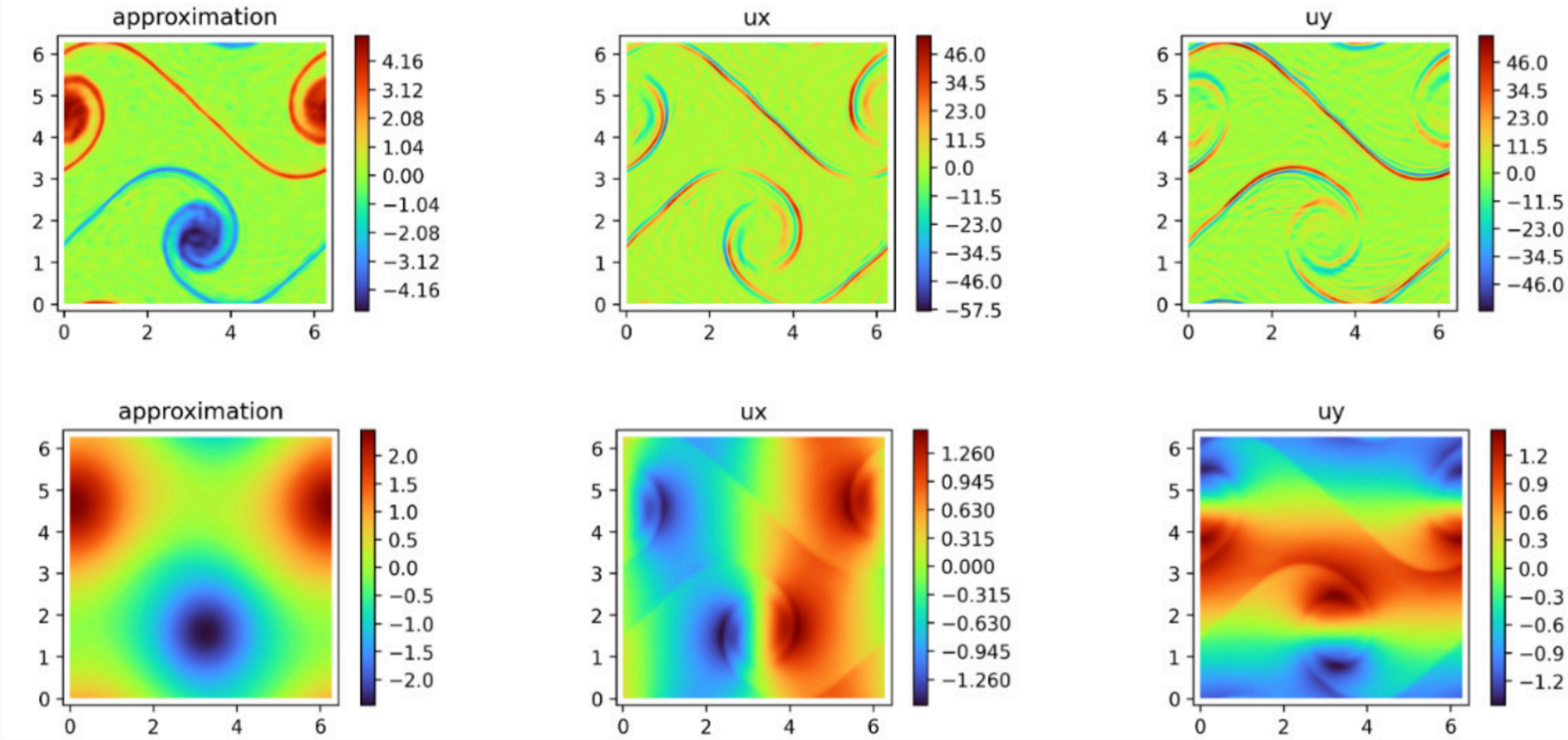
Remark: We can easily generalize implicit, explicit, splitting or semi-Lagrangian schemes to nonlinear spaces.

- Many more optimizations but shorter because:
  - we start from a good initialization (solution at the previous time step)
  - we can use smaller spaces (because we approximate over a short time step).



Examples

- Incompressible Euler:



Flow learning/model learning

- Let a parametric ODE:

$$\begin{cases} \frac{\mathrm{d}}{\mathrm{dt}}\mathrm{u}(\mathrm{t},\boldsymbol{\theta}) = \mathcal{F}(\mathrm{u}(\mathrm{t},\boldsymbol{\theta}),\boldsymbol{\theta}) \\ \mathrm{u}(0,\boldsymbol{\theta}) = \mathrm{u}_0(\boldsymbol{\theta}) \end{cases}$$

Definition (Flow): The flow $\Phi_{\mathrm{t},\boldsymbol{\theta}}$ is defined by $\mathrm{u}(\mathrm{t},\boldsymbol{\theta}) = \Phi_{\mathrm{t},\boldsymbol{\theta}}(\mathrm{u}_0(\boldsymbol{\theta}))$. It is the resolvent of the system.

- In Scimba we can learn a nonlinear approximation of a parametric flow:
  - discrete or continuous flow in time.
  - classical or **symplectic** flow (associated with Hamiltonian systems).
  - optimization over complete trajectories (to be validated).

Remark (Application): These flow learning can be used to learn latent variable dynamics in order reduction.

So far

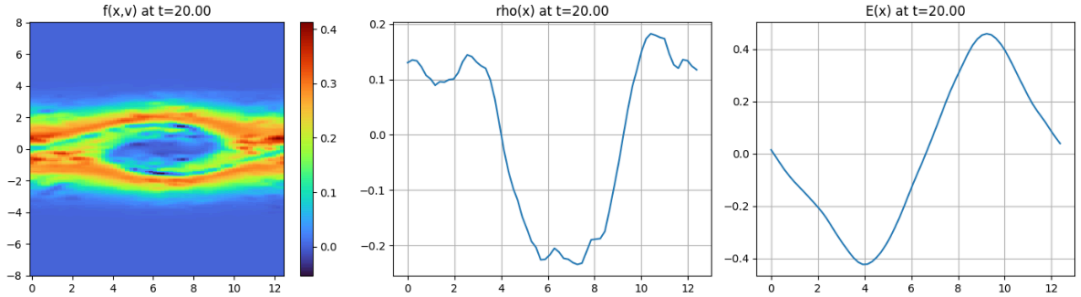
Examples

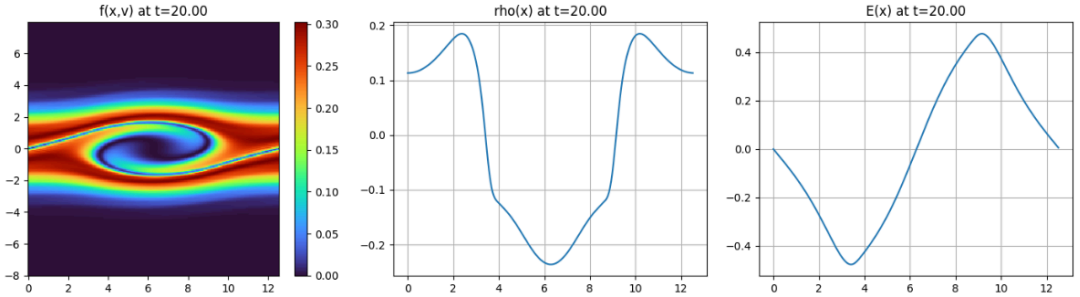
- Classique δ-PIC

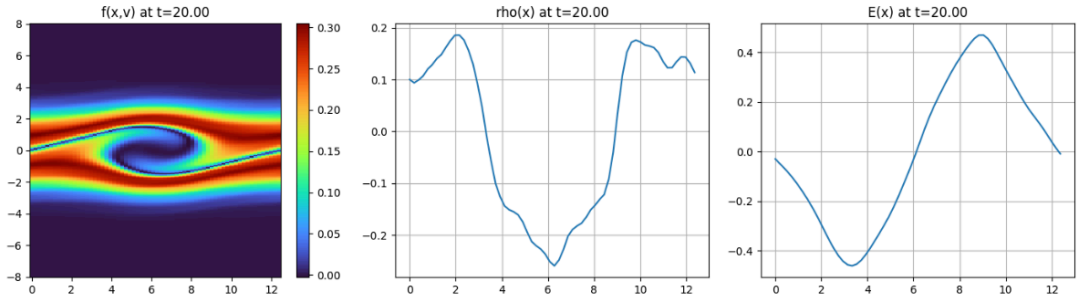
$$f(t,x,v) = f^{init}(x,v) + \sum_{k=1}^{N} w_k \delta(\mathbf{x} - \mathbf{X}_k(t))\delta(\mathbf{v} - \mathbf{V}_k(t))$$

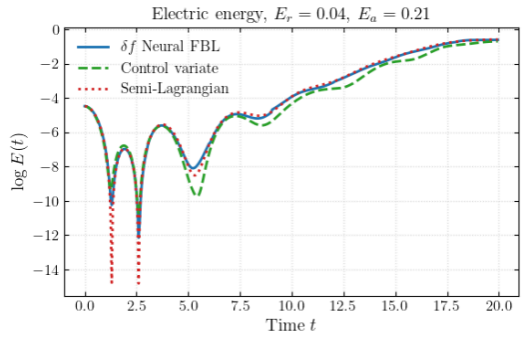
- Hybrid PIC: add a NN approximation

$$f(t,x,v) = f^{init}\left(\left(\varphi_{\boldsymbol{\theta}_n} \circ \ldots \circ \varphi_{\boldsymbol{\theta}_1}\right)(x,v)\right) + \sum_{k=1}^{N} w_k \delta(\mathbf{x} - \mathbf{X}_k(t))\delta(\mathbf{v} - \mathbf{V}_k(t))$$









Example code

- link(“https://www.scimba.org”)

1.2. Future

Ongoing work in torch version

Remark (Adaptive sampling in Torch): Learning can be difficult if the PDE has very localized phenomena (shocks, interfaces, ...). To avoid too many collocation points, we can use adaptive sampling strategies.

- In practice, we use a generative model to dynamically learn $\mathrm{p}_{\boldsymbol{\theta}}(\mathbf{x}) \sim \mid \partial_{\mathbf{x}}^{\mathrm{p}} \mathrm{R}(\mathbf{u}_{\boldsymbol{\theta}}) \mid^2$ (or other)
- Models:
  - flow matching
  - normalizing flow
  - optimal transport

Ongoing and futur works in Jax

- Neural operators works:
  - Neural operators (Fourier, transformer, Green's function) accurate and general in mesh and geometry.
  - Structure-preserving neural operators (symplectic, dissipative) + PDE flow learning
  - Rom based on Invertible networks + POD

Example: Accurate NO for Helmholtz in heterogeneous medium + generative NO for the inverse problem (with V. Italiano Numpex)

Objectives (Current development in Jax):

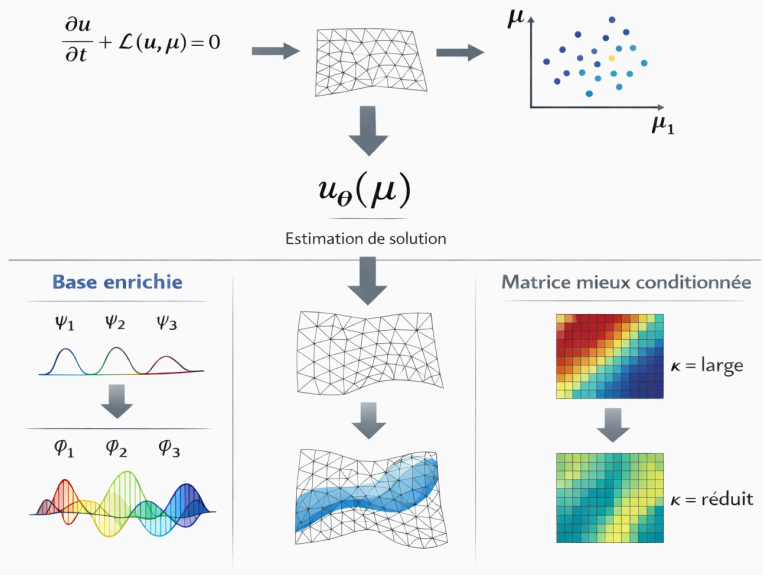
- Neural operator
- Adding optimizers
- Adding features: multi-subdomain, multiple BC, DD etc
- Domain decomposition: One PINNs per subdomain with overlaps (FBPINNs methods). Partitioning with GMSH
- Matrix-free + block preconditionning Natural gradient for FBPINNs. Extension to other optimizers
- Efficient linear algebra? Coupling with C++?

Hybrid DG

Remark: uses significantly fewer degrees of freedom than a conventional code starting from dimension 3.

- fewer calculations? only in larger dimensions
- less memory ==> in an HPC context, less communication

Remark: The lack of guarantees of these methods is a limitation. Weak control on the error and **stability**.



Objectives: hybrid approaches and restore convergence guarantees. Focus on coupling with multi-patch DG methods.

- **Idea 1**: The network roughly captures a set of solutions (parametric PINNs, NO. We enrich the DG space (basis, mesh, PC) with the prediction.
- **Idea 2**: DG between macro cells with PINNs.
- **Idea 3**: Invertible change of coordinates/variables which simplifie the solution for the DG space