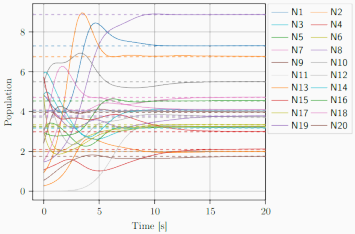
Physics-informed neural networks: what, how and why ?

Nicolas Jouvin - MIA Paris-Saclay (INRAE)

AG Mathnum - 08 octobre 2026

Motivation: dynamic modeling in life & environmental sciences

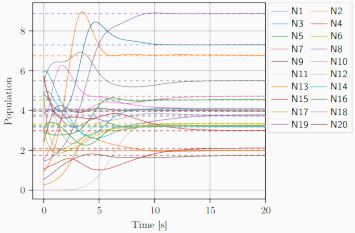
Biology



Hossie et al. (2024)

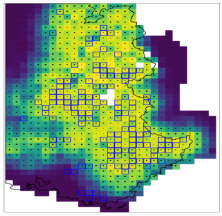
Motivation: dynamic modeling in life & environmental sciences

Biology



Hossie et al. (2024)

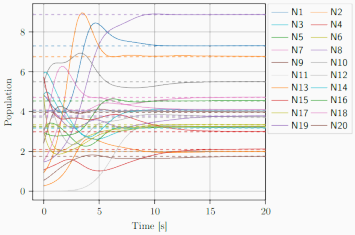
Ecology



Louvrier et al. (2020)

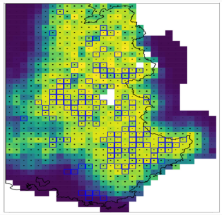
Motivation: dynamic modeling in life & environmental sciences

Biology



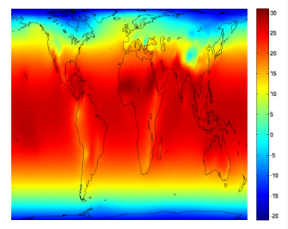
Hossie et al. (2024)

Ecology



Louvrier et al. (2020)

Climatology



Clarotto et al. (2024)

And many more domains related to MathNum/INRAe

Challenge: solve the underlying equations

Partial Differential Equations (PDEs) in a nutshell

"Physical" knowledge is encoded in a PDE, via a **differential operator**

$$\forall t, x \in I \times \Omega, \qquad \boxed{\mathcal{N}_\theta[u](t,x) = 0.} \tag{PDE}$$

$\rightsquigarrow$ you can have time independence (stationary PDEs) or space independence (ODE)

In addition, there usually are some

- initial condition

$$u(0,\cdot) = u_0(x), \quad \forall x \in \Omega \tag{IC}$$

- boundary condition

$$\mathcal{B}[u](t,\delta x) = f(t,\delta x), \quad \forall t \in I, \forall \delta x \in \partial\Omega \tag{BC}$$

PINNs are a machine-learning approach to solve PDEs

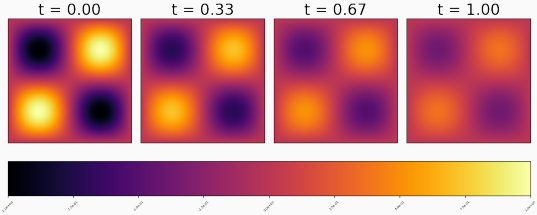
An example: the heat equation

Heat equation in 2D: spatio-temporal domain $I \times \Omega = [0, T] \times [-1, 1]^2$ defined by

$$\theta = D \in \mathbb{R}, \quad \mathcal{N}_\theta = \frac{\partial}{\partial t} - D\Delta,$$

$$\begin{cases} \partial_t u(t,x) - D\Delta u(t,x) = 0, & \text{(Heat PDE)} \\ u(0,x) = \sin(\pi x_1)\sin(\pi x_2), & \text{(IC)} \\ u(t,x) = 0, \ x \in \partial\Omega & \text{(Dirichlet BC)} \end{cases}$$

$$u^\star(t,x) = e^{-2D\pi^2 t} u_0(x)$$



Snapshots $u^\star(t_i, x)$ for $D = 0.05$

What are PINNs ?

A primer/refresher on machine learning

PINNs are a specific machine-learning approach (Raissi et al. 2017)

Any machine learning method can be defined through

1) Some data $\mathcal{D}$

2) A loss function to be minimized (learning)

$$\hat{u} = \underset{u \in \mathcal{F}}{\arg\min}\, \mathcal{L}(u; \mathcal{D})$$

3) A space of functions $\mathcal{F}$. For example u_ν parametrized by $\nu \in \mathbb{R}^p$

- linear $u_\nu(x) = \nu^\top x$, polynomial, functional basis (splines, etc.)
- or ... neural networks ! more on this later

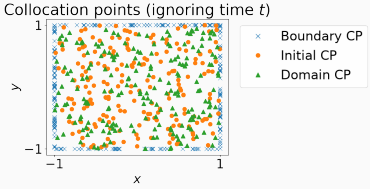
Training? Minimization of the loss

$$\mathcal{L}^\star(u) = \int \|\mathcal{N}_\theta[u](t,x)\|^2 \,\mathrm{d}t\,\mathrm{d}x\, (+ \text{IC} + \text{BC}) \qquad \rightsquigarrow \text{Monte-Carlo approximation}$$

1) Data: two sources of data for PINNs

The dataset $\mathcal{D}$ is composed of

- **Collocation points**
  1. Domain: $\{t_i, x_i\}$ sampled on $I \times \Omega$
  2. Initial: $\{(0, x_j)\}_{j=1}^{n_i}$ sampled on $\{0\} \times \Omega$
  3. Boundary: $\{t_k, x_k\}_{k=1}^{n_{bc}}$ sampled on $I \times \partial\Omega$



2) Loss: the PDE-part

3 loss components

- **Dynamic loss** $\rightsquigarrow$ PDE *residuals* on the collocation points

$$\mathcal{L}_{dyn}(u) = \frac{1}{n_c} \sum_{i=1}^{n_c} |\mathcal{N}_\theta[u](t_i, x_i)|^2$$

2) Loss: the PDE-part

3 loss components

- **Dynamic loss** $\rightsquigarrow$ PDE *residuals* on the collocation points

$$\mathcal{L}_{dyn}(u) = \frac{1}{n_c} \sum_{i=1}^{n_c} |\mathcal{N}_\theta[u](t_i, x_i)|^2$$

- **Initial condition**

$$\mathcal{L}_{ic}(u) = \frac{1}{n_i} \sum_{j=1}^{n_i} |u(0, x_j) - u_0(x_j)|^2$$

2) Loss: the PDE-part

3 loss components

- **Dynamic loss** $\rightsquigarrow$ PDE *residuals* on the collocation points

$$\mathcal{L}_{dyn}(u) = \frac{1}{n_c} \sum_{i=1}^{n_c} |\mathcal{N}_\theta[u](t_i, x_i)|^2$$

- **Initial condition**

$$\mathcal{L}_{ic}(u) = \frac{1}{n_i} \sum_{j=1}^{n_i} |u(0, x_j) - u_0(x_j)|^2$$

- **Boundary**

$$\mathcal{L}_{bc}(u) = \frac{1}{n_{bc}} \sum_{k=1}^{n_{bc}} |\mathcal{B}[u](t_k, x_k) - f(t_k, x_k)|^2$$

2) Loss: the PDE-part

3 loss components

- **Dynamic loss** $\rightsquigarrow$ PDE *residuals* on the collocation points

$$\mathcal{L}_{dyn}(u) = \frac{1}{n_c} \sum_{i=1}^{n_c} |\mathcal{N}_\theta[u](t_i, x_i)|^2$$

- **Initial condition**

$$\mathcal{L}_{ic}(u) = \frac{1}{n_i} \sum_{j=1}^{n_i} |u(0, x_j) - u_0(x_j)|^2$$

- **Boundary**

$$\mathcal{L}_{bc}(u) = \frac{1}{n_{bc}} \sum_{k=1}^{n_{bc}} |\mathcal{B}[u](t_k, x_k) - f(t_k, x_k)|^2$$

Whole PDE loss with balancing weights w_{dyn}, w_{ic}, w_{bc}

$$\mathcal{L}_{PDE} = w_{dyn}\mathcal{L}_{dyn} + w_{bc}\mathcal{L}_{bc} + w_{ic}\mathcal{L}_{ic}$$

3) The function class: neural network

As the name suggests, we look for

$$u \in \mathcal{F} = \left\{ u_\nu = u_{\nu,L} \circ \ldots \circ u_{\nu,1}, \text{ NNs with parameters } \nu \right\}$$

3) The function class: neural network

As the name suggests, we look for

$$u \in \mathcal{F} = \left\{ u_\nu = u_{\nu,L} \circ \ldots \circ u_{\nu,1}, \text{ NNs with parameters } \nu \right\}$$

3) The function class: neural network

As the name suggests, we look for

$$u \in \mathcal{F} = \left\{ u_\nu = u_{\nu,L} \circ \ldots \circ u_{\nu,1},\ \text{NNs with parameters } \nu \right\}$$

- **Why neural networks ?** They are both
  1. suitable approximation (asymptotically with the number of neurons/layers)
  2. suitable for automatic differentiation $\rightsquigarrow$ **easy-and-exact** $\mathcal{N}_\theta[u_\nu]$ at any (t, x)

- **Optimization** Non-convex program $\hat{\nu} \in \arg\min_\nu \mathcal{L}(u_\nu) \rightsquigarrow$ need good optimizers!

In practice, preconditioned variants of gradient descent (Müller et al. 2023)

$$\nu_{k+1} \leftarrow \nu_k - \eta P_k^\dagger \nabla \mathcal{L}(\nu_k)$$

$\rightsquigarrow$ all derivatives are computed via autodiff.

Why ? Three types of PINN problems

1) PDE solver (forward problem)

Goal: solve the PDE prescribed by $\mathcal{N}_\theta$ using the PDE loss only:

$$\hat{\nu} \in \arg\min_{\nu} \mathcal{L}_{PDE}(u_\nu).$$

$\rightsquigarrow$ PDE parameters θ are **fixed and known** + no observations

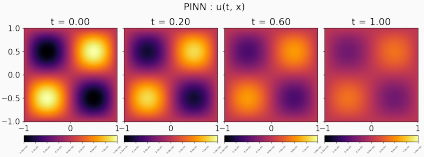
1) PDE solver (forward problem)

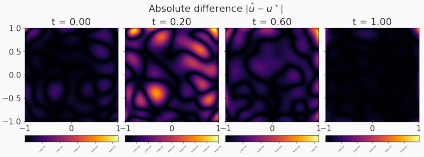
Goal: solve the PDE prescribed by $\mathcal{N}_\theta$ using the PDE loss only:

$$\hat{\nu} \in \arg\min_{\nu} \mathcal{L}_{PDE}(u_\nu).$$

$\rightsquigarrow$ PDE parameters θ are **fixed and known** + no observations

Example: 2D heat equation $\mathcal{N}_D = \frac{\partial}{\partial t} - D\Delta$, with $D = 0.05$ + sinus IC & Dirichlet BC





- Small & simple architecture: $(3, 16, 16, 1)$-MLP with 320 parameters (shallow)
- Running time $\sim 1s$ on this laptop (CPU).[1]

[1] Implement this yourself in this step-by-step tutorial.

1bis) hybrid-modeling

Regression with noisy observations: $y_l = u^\star(t_l, x_l) + \epsilon_l$ with MSE

$$\mathcal{L}_{obs}(u) = \frac{1}{n_{obs}} \sum_{l=1}^{n_{obs}} |u(t_l, x_l) - y_l|^2$$

1bis) hybrid-modeling

Regression with noisy observations: $y_l = u^\star(t_l, x_l) + \epsilon_l$ with MSE

$$\mathcal{L}_{obs}(u) = \frac{1}{n_{obs}} \sum_{l=1}^{n_{obs}} |u(t_l, x_l) - y_l|^2$$

Goal: regress over all u satisfying a prescribed equation

$$\hat{u} \in \underset{\mathcal{N}_\theta[u]=0}{\arg\min}\, \mathcal{L}_{obs}(u)$$

1bis) hybrid-modeling

Regression with noisy observations: $y_l = u^\star(t_l, x_l) + \epsilon_l$ with MSE

$$\mathcal{L}_{obs}(u) = \frac{1}{n_{obs}} \sum_{l=1}^{n_{obs}} |u(t_l, x_l) - y_l|^2$$

Goal: regress over all u satisfying a prescribed equation

$$\hat{u} \in \underset{\mathcal{N}_\theta[u]=0}{\arg\min}\, \mathcal{L}_{obs}(u)$$

Hard $\rightsquigarrow$ uses a two-component loss with relaxed constraint

$$\mathcal{L}(u) \coloneqq \underbrace{w_{obs}\mathcal{L}_{obs}}_{\text{statistical information}} + \underbrace{\mathcal{L}_{dyn} + w_{bc}\mathcal{L}_{bc} + w_{ic}\mathcal{L}_{ic}}_{\text{relaxed “physics” constraint}}$$

2) Inverse problem

Goal: find $\theta^\star$ from observations of PDE solution $u^\star = u^{\theta^\star}$

- **Inverse problem** $\rightsquigarrow$ find the best PDE parameters that fit through the data

$$(\hat{u}, \hat{\theta}) \in \arg\min_{u,\theta} \mathcal{L}(u, \theta) = w_{obs}\mathcal{L}_{obs}(u) \quad + \mathcal{L}_{PDE}(u; \theta)$$

2) Inverse problem

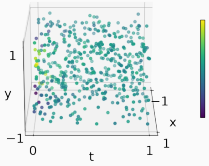
Goal: find $\theta^\star$ from observations of PDE solution $u^\star = u^{\theta^\star}$

- **Inverse problem** $\rightsquigarrow$ find the best PDE parameters that fit through the data

$$(\hat{u}, \hat{\theta}) \in \arg\min_{u,\theta} \mathcal{L}(u, \theta) = w_{obs}\mathcal{L}_{obs}(u) \quad + \mathcal{L}_{PDE}(u; \theta)$$

- **Example** estimation of the diffusion coefficient $\theta = D$

Observations: $u_l = u^\star(t_l, x_l) + \sigma\varepsilon_l$



D unkown!

500 observations

+

Heat equation model

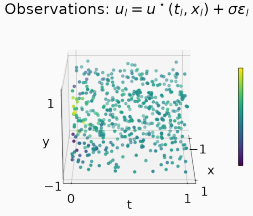
2) Inverse problem

Goal: find $\theta^\star$ from observations of PDE solution $u^\star = u^{\theta^\star}$

- **Inverse problem** $\rightsquigarrow$ find the best PDE parameters that fit through the data

$$(\hat{u}, \hat{\theta}) \in \arg\min_{u,\theta} \mathcal{L}(u, \theta) = w_{obs}\mathcal{L}_{obs}(u) \quad + \mathcal{L}_{PDE}(u; \theta)$$

- **Example** estimation of the diffusion coefficient $\theta = D$

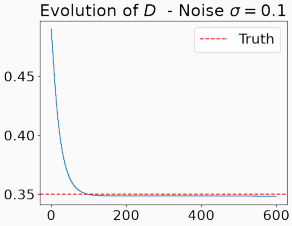


D unkown!

500 observations

+

Heat equation model

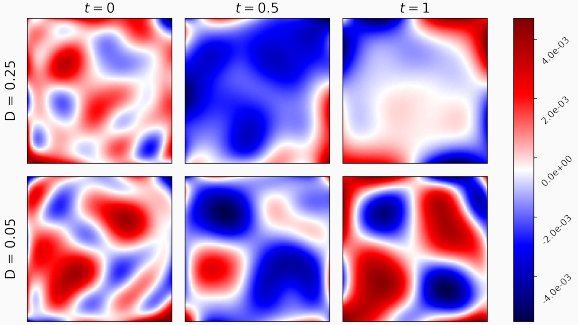


3) Meta-modeling (aka parametric problem, surrogate modeling)

Goal learn a neural net $\hat{u}_\nu(\cdot;\theta)$ solution of PDE for a range of θ

- **Approximate** the map $\theta \mapsto u^\theta \in \mathcal{M}$ where $\mathcal{M} = \{\text{manifold of PDE solutions}\}$

- **Example** heat equation with $D \in [0.05, 0.5]$



- **Augmented** training set $\{\theta_m\}_{m=1}^{n_p}$

$$\mathcal{L}_{dyn}(\nu) = \frac{1}{n_p n_c} \sum_{m=1}^{n_p} \sum_{i=1}^{n_c} \left| \mathcal{N}_{\theta_m}[u_\nu](t_i, x_i; \theta_m) \right|^2$$

$\rightsquigarrow$ no simulation required before-hand!

Conclusion

Conclusion on PINNs

Pros:

- Mesh-less (no curse of dimensionality)
- Versatile hybrid-ML framework for data-integration

Cons:

- Non convexity: good optimizers are crucial !
- Few guarantees compared to well-studied numerical schemes $\|\hat{u}_\nu - u^\star\| = \mathcal{O}(?)$

Broader landscape: "scientific machine-learning" (SciML)

- Hybridization - Neural numerical methods: Galerkin, time-discrete, FEM, etc.
- *Neural operators* ($\neq$ PINNs)
- Neural ODE: learn unknown physics (*e.g.* a drift) from data.

Thank you for your attention

Collaborators: Lucia Clarotto, Hugo Gangloff, Béatrice Laroche, Julien Papaïx, Lorenzo Sala

Click on the magic lamp
(or teapot)

Checkout the jinns Python package

PINNs with JAX: inverse problem, metamodeling,
inhomogeneous models, and much more!

References

- Clarotto, Lucia et al. (2024). **“The SPDE approach for spatio-temporal datasets with advection and diffusion”.** In: Spatial Statistics 62, p. 100847.
- Hossie, Paguiel Javan et al. (2024). **“Simulating interactions in microbial communities through Physics Informed Neural Networks: towards interaction estimation”.** In.
- Louvrier, Julie et al. (2020). **“A mechanistic–statistical species distribution model to explain and forecast wolf (Canis lupus) colonization in South-Eastern France”.** In: Spatial Statistics 36, p. 100428.
- Müller, Johannes and Marius Zeinhofer (2023). **“Achieving High Accuracy with PINNs via Energy Natural Gradient Descent”.** In: Proceedings of the 40th International Conference on Machine Learning. Ed. by Andreas Krause et al. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 25471–25485.
- Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis (2017). **“Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations”.** In: arXiv preprint arXiv:1711.10561.

Automatic differentiation

Automatic differentiation $\implies$ exact derivative

Automatic differentiation $\neq$ finite-differences

Main ideas

1. u is a composition of simple functions

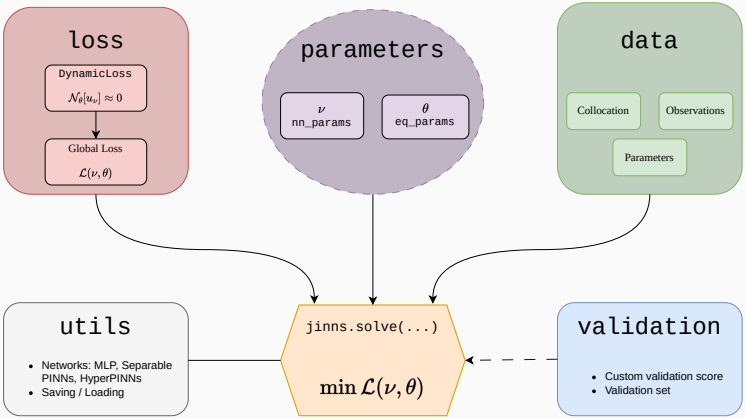
$$u = u_L \circ \ldots \circ u_1.$$

2. Each ∂u_l is easy to compute
3. Then, chain rule yields a product of Jacobians

$$\partial u(y) = \partial u_L(u_{1:(L-1)}(z)) \ldots \partial u_1(z).$$

Take-home message

- **Exact** jacobians at any point z (not a symbolic expression though)
- This a *technology*: softwares implement chain rule & $v^\top \partial u_l(\cdot)$ for many u_l $(\exp, \log, z^\lambda, \ldots)$
- For PINNs, z can both be parameters ν or input (t, x)

Jinns package

Flowchart for users