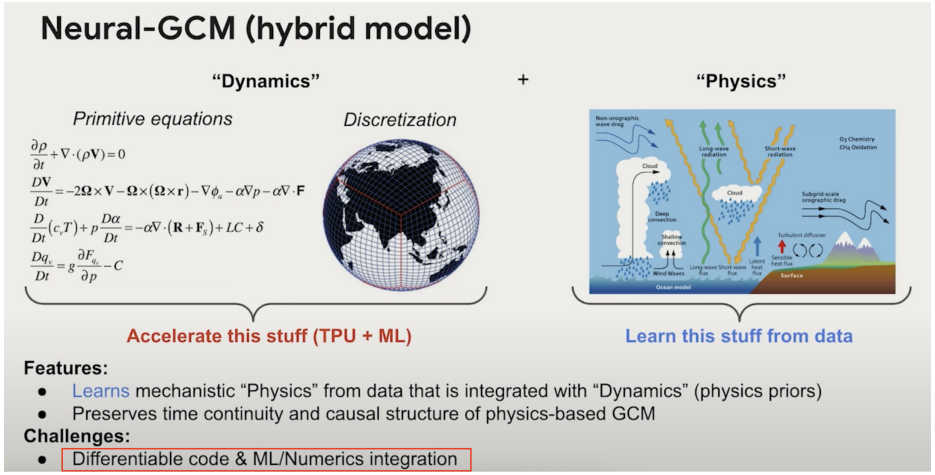
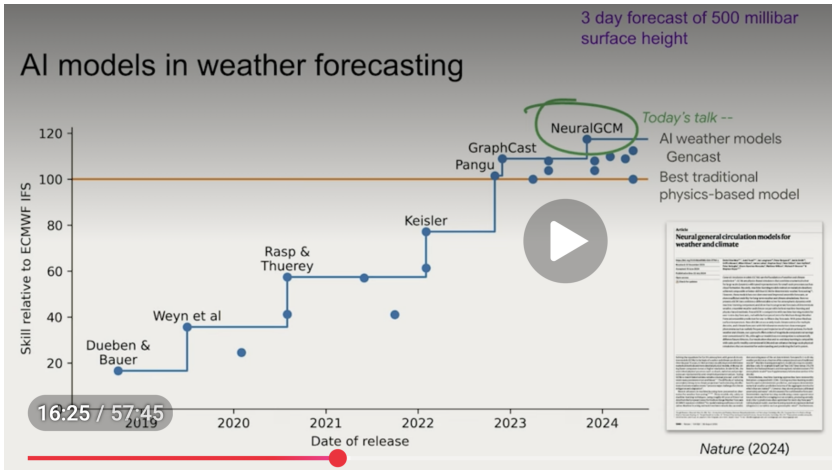
A nebulous presentation on hybrid models, transport, AI and more

B. Després (LJLL-SU+MEGAVOLT-INRIA)

Hybrid model





(Michael Brenner)

Being optimistic, hybrid modeling is also part of a possible future of AI for plasma physics.

Here I show some internal contradictions for this scenario (hybrid modeling for plasma physics).

What is hybrid modeling ?

Take your favorite **PDE-based model** and perform **data-enhancement**

$$\begin{cases} \partial_{\mathbf{t}}\mathbf{U} + \mathcal{L}(\mathbf{U}, \partial_{\mathbf{x}}\mathbf{U}, \partial_{\mathbf{xx}}\mathbf{U}, \dots) = S_\theta^{\mathrm{NN}}(\alpha), & \theta \text{ contains all NN } \textbf{weights}, \\ \mathbf{U}(\mathbf{x}, \mathbf{0}) = \beta, & \text{initial data at time } t = 0, \end{cases}$$

where one has a large **dataset of observations**

$$\mathcal{D} = \{(\alpha_i, \beta_i, \gamma_i) \mid i = 1, 2, \dots, N\} \subset \mathbb{R}^{m+n+n}.$$

The cost function evaluated at given final time $T > 0$ is something like

$$J(\theta) = \frac{1}{N} \sum_i \int_{x \in \Omega} |U(x, T : \alpha_i, \beta_i, \theta) - \gamma_i|^2 \, dx \qquad (+ \text{ many other terms}).$$

The cost function is minimized with gradient descent (training)

$$\theta'(t) = -\nabla J(\theta(t)).$$

Blue part : NN functions

Take a fully-connected **feed-forward neural network** function $f : \mathbb{R}^{a_0} \to \mathbb{R}^{a_{\ell+1}}$

$$f = f_\ell \circ S_\ell \circ f_{\ell-1} \circ S_{\ell-1} \circ \cdots \circ f_1 \circ S_1 \circ f_0$$

where $f_i(x_i) = W_i x_i + b_i \in \mathbb{R}^{a_{i+1}}$ $(x_i \in \mathbb{R}^{a_i})$ is linear
and $S_i \in C^0(\mathbb{R})$ is a non linear activation function $0 \leq S_i'(x) \leq 1$.

• Sigmoidal functions $S_I = \sigma$ are smooth activation functions.

• ReLU function $S_i = R$ with $R(x) = \max(0, x)$ is an extremely popular activation function used in production codes.

• *Max-pooling* used in **convolutional neural networks** is based on the maximum function $(a, b, \dots) \mapsto \max(a, b, \dots) \in \mathbb{R}$.

Regularity issues for ML

Training is based on explicit evaluation of the gradient of the cost

$$\theta'(t) = -\nabla J(\theta(t)), \qquad J = \text{Loss}.$$

- $f \in \text{Lip}(\mathbb{R}^{a_0} : \mathbb{R}^{a_{\ell+1}})$: Rademacher Th. $\Longrightarrow \nabla f \in L^\infty(\mathbb{R}^{a_0} : \mathcal{M}_{a_{\ell+1},a_0}(\mathbb{R}))$.
- ML is universally based on the **chain rule/backpropagation/autodiff**

$$\nabla f(x) = A_\ell(x)B_\ell(x)A_{\ell-1}(x)B_{\ell-1}(x)\dots A_1(x)B_1(x)A_0(x).$$

Non ambiguous in C^1. **Highly ambiguous** for Lipschitz functions.

Example taken from Karpathy's `microgpt.py`

```
The most atomic way to train and run inference for GPT in Python. @karpathy
```

```
def log(self): return Value(math.log(self.data), (self,), (1/self.data,))
def exp(self): return Value(math.exp(self.data), (self,), (math.exp(self.data),))
def relu(self): return Value(max(0, self.data), (self,), (float(self.data > 0),));
```

Last line means $R'(0) = 0$, where $R(x) = \max(0, x)$.

Basic contradictions

- Do as if ReLU, defined by $R(x) = \max(0, x)$, is smooth at $x = 0$.
- From $R(R(x)) = R(x)$, the chain rule $R'(x)R'(R(x)) = R'(x)$ yields

$$R'(0)^2 = R'(0) \Longrightarrow R'(0) = 0 \text{ or } 1.$$

- From $R(x) - R(-x) = x$, we deduce

$$2R'(0) = 1 \Longrightarrow R'(0) = 1/2.$$

- **Of course, no solution is satisfactory.**

Same phenomenon for max (Boustany paradox)

- For $x = (1, 2, 3, 4)$ encode the function $t \mapsto f(t) = \max_1(tx) - \max_2(tx)$.

```
def max1(x):
res = x[0]
for i in range(1, a):
if x[i] > res:
   res = x[i]
return res
```

```
def max2(x):
return torch.max(x)
```

- **Pytorch-autodiff** is (would be similar with TensorFlow, Scikilearn, ...,)

t	-1	-0.5	-0.01	0	0.01	0.5	1
$f(t)$	0	0	0	0	0	0	0
$f'(t)$	0	0	0	-1.5	0	0	0

- The paradox can be solved in the good algebra.

- Since such pathologies concerns only the **blue part**, let's us decide we will never use ReLU for modeling data in hybrid models for plasma physics.

Red part : Discrete transport schemes

- Take your favorite advection equation $\mathbf{t \geq 0},\ \mathbf{x} \in \mathbb{R}:\quad \partial_t \mathbf{u} + \mathbf{a}\partial_x \mathbf{u} = \mathbf{0}.$

- The upwind scheme is

$$\frac{\mathbf{u_j}^{n+1} - \mathbf{u_j^n}}{\mathbf{\Delta t}} + \mathbf{a}\frac{\mathbf{u_{j+\frac{1}{2}}} - \mathbf{u_{j-\frac{1}{2}}}}{\mathbf{\Delta x}} = \mathbf{0},$$

where $\mathbf{u_{j+\frac{1}{2}}} = \mathbf{G_a}(\mathbf{u_j^n}, \mathbf{u_{j+1}^n})$ with

$$\mathbf{G_a}(\alpha, \beta) = \alpha \text{ if } \mathbf{a} > \mathbf{0},\ \mathbf{G_a}(\alpha, \beta) = \beta \text{ if } \mathbf{a} \leq \mathbf{0}.$$

- The upwind scheme is the basis of many schemes (all stable under CFL).
- The time steps are composed one after the other. That is any numerical scheme can be seen as the composition of basic functions.

Principle

$\{\text{Numerical schemes}\} \subset \{\text{Neural Nets}\}$.

Autodiff and discrete transport schemes

What matters is the regularity of the flux function F

$$au_{j+\frac{1}{2}} = F(a, u_j, u_{j+1}) = aG_a(u_j, u_{j+1}) = R(a)u_j - R(-a)u_{j+1}.$$

• Unfortunately the function $a \mapsto R(a)$ is not differentiable at $a = 0$ (in ML codes, usual convention is $R'(0) = 0$).

Principle

The non-differentiable ReLU function $R(x) = \max(0, x)$ is (almost) everywhere in advection dominated numerical codes.

Even if the code does not blow up, we just don't know what's going on.

- Thiry-Li-Mémin-Roullet, G. A unified formulation of quasi-geostrophic and shallow water equations 2024.

Even worse for high order schemes

Consider the MinMod TVD scheme

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + a \frac{u_{j+\frac{1}{2}}^n - u_{j-\frac{1}{2}}^n}{\Delta x} = 0,$$

where

$$\begin{cases} a \geq 0 & u_{j+\frac{1}{2}} = u_j + \frac{1}{2}(1-\nu)\text{minmod}(u_{j+1} - u_j, u_j - u_{j-1}), \\ a < 0 & u_{j+\frac{1}{2}} = u_{j+1} + \frac{1}{2}(1-\nu)\text{minmod}(u_{j+1} - u_j, u_j - u_{j-1}). \end{cases}$$

Check that $\text{minmod}(x, y) = R(-\max(-x, -y)) - R(-\max(x, y))$.

Lemma

The 2nd order Lax Wendroff flux belongs to the regularity class C^1_{pw}.

Presumably, all high order conservative fluxes
(TVD, TVB, BV, WENO, MUSCL, ...)
belong to the same class.

Conclusion : The mathematics of autodiff for hybrid transport models generates new interesting questions. (we will soon investigate with Petar Samardzic (new PhD) and Louis Thiry (co-supervisor)).

Summary for this part

To develop hybrid models in plasmas physics, what is needed is

- Recode to insure compatibility with automatic differentiation (use JAX, ...).

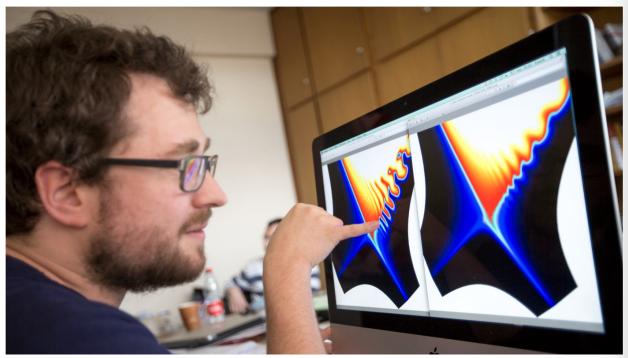
  In particular for the transport part.
- At the same time, ensure numerical integration is stable.

This was what Michael Brenner focused on in his presentation on meteorology modeling.

Presumably this holds true for plasma physics as well.

Scimba

• This part is entirely based on the work of
Emmanuel Franck, Victor Michel-Dansac and the INRIA/MACARON team



Numerical schemes for hyperbolic equations enhanced by Scientific Machine Learning

Victor Michel Dansac (Inria)

• Scimba is an entirely new library based on PYTORCH for solving PDES, for education and research
`https://www.scimba.org`
They do Claude coding, so development runs at super fast pace.

• It uses some ideas coming from PINNS, but enhanced with
a modeler, JAX, natural gradients and other optimization techniques.

Principles

• Take your favorite model=PDE + **Boundary Conditions** (or IC), written in symbolic form as

$$\begin{cases} \mathcal{L}u(x) = f(x) & x \in \Omega, \\ u(x) = g(x) & x \in \Gamma = \partial\Omega \end{cases}$$

• Rewrite as $u = \text{argmin} J(\theta)$ where

$$J(\theta) = \int_{x\in\Omega} \left|\mathcal{L}v_\theta^{\text{NN}}(x) - f(x)\right|^2 dx + \lambda \int_{x\in\Gamma} |v_\theta^{\text{NN}}(x) - g(x)|^2 d\sigma \quad (\lambda > 0).$$

• What they did is :
- implement a modeler to design domains/shapes (Ω)
- define functions with NN (layers and neurons)
- use only smooth activation function (no ReLU stuff)
- use Monte-Carlo to approximate integrals (on any Ω and Γ)
- use autodiff to assemble $\mathcal{L}v_\theta^{\text{NN}}(x_i)$ at MC points x_i
- use autodiff to minimize J and find quasi-optimal values $\theta_\star$

DeepRitz (a variant)

Weinan+Bing, The Deep Ritz method : A deep learning-based numerical algorithm for solving variational problems, 2017.

Solve elliptic problems like the model equation

$$\begin{cases} -\Delta u + u = f, & x \in \Omega, \\ \partial_n u = 0, & x \in \partial\Omega, \end{cases}$$

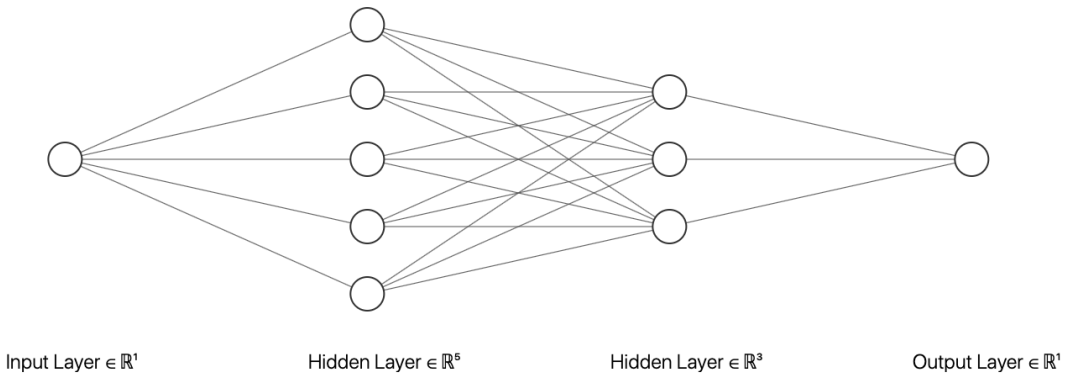
using

$$u = \underset{v \in V}{\operatorname{argmin}} \int_\Omega \left(\frac{1}{2} |\nabla v(x)|^2 + \frac{1}{2} |v(x)|^2 - f(x) v(x) \right) dx, \qquad V = H^1(\Omega).$$

- Lagaris+Likas+Fotiadis, Artificial Neural Networks for Solving Ordinary and PDEs, 1997.
- Raissi+Perdikaris+Karniadakis. Physics-informed neural networks (PINNs), 2019.
- Kharazmi+Zhang+Karniadakis, VPINNs : Variational Physics-Informed Neural Networks For Solving Partial Differential Equations, 2019,
- Berrone+Canuto+Pintore. VPINNs quadratures and test functions, 2021.

NN modeling

- Step 1 : choose a NN architecture (hyper-parameters)



from which constructs functions $x \mapsto v_\theta^{\mathrm{NN}}(x)$ where we renamed the weights $\theta = W$. It defines the space

$$V^{NN} = \left\{ v_\theta^{\mathrm{NN}} \text{ for all weights } \theta \right\}.$$

- Step 2 : minimize $u^{\mathrm{NN}} = \underset{v_\theta^{\mathrm{NN}} \in V^{\mathrm{NN}}}{\operatorname{argmin}} J(\theta)$

$$J(\theta) = \int_\Omega \left(\frac{1}{2} |\nabla v_\theta^{\mathrm{NN}}(x)|^2 + \frac{1}{2} |v_\theta^{\mathrm{NN}}(x)|^2 - f(x) v_\theta^{\mathrm{NN}}(x) \right) dx.$$

- Step 3 : replace $\int h(x)dx$ with $\Sigma h(x_i) w_i$: **Riemann, Monte Carlo, . . .**

Install

Extremely easy. Install :

```
pip3 install --upgrade pip
pip3 install --user virtualenv
virtualenv myenv2
source myenv2/bin/activate
pip install torch
pip install matplotlib
pip install jupyter
pip install scimba
pip install jax
```

Then run with either :

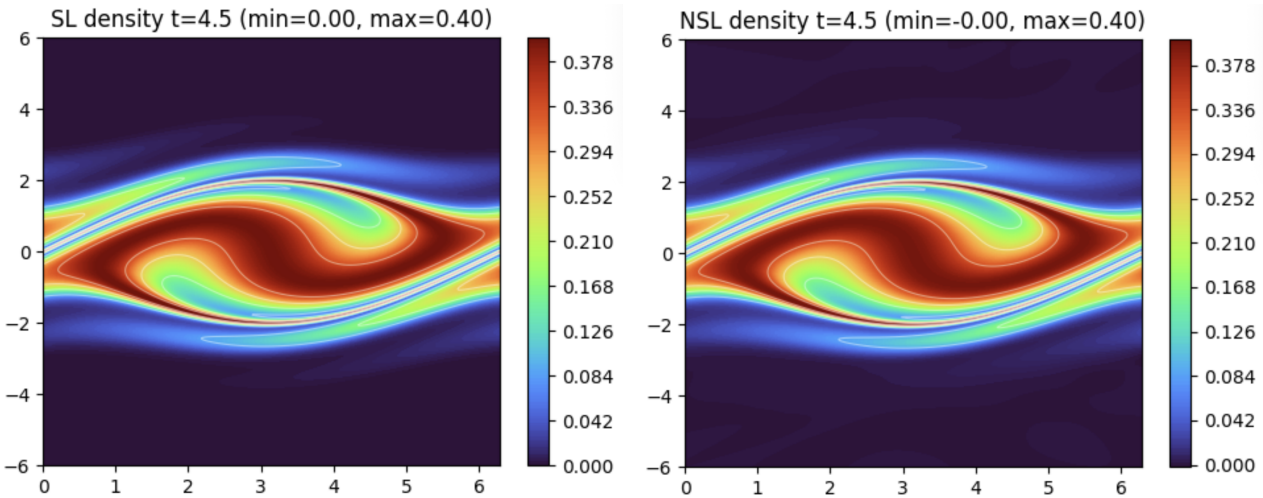
```
jupyter notebook
```

or :

```
python file.py
```

Let us run Jupyter Notebooks

`geometries.ipynb`
`gradshaf.ypnb`
`tranport.ipynb`



Efficiency in high dimensions

dimension d	classical SL				neural SL			
	n_x	Time	relative error		Time		relative error	
			e_{L^2}	e_{L^∞}	init.	iter.	e_{L^2}	e_{L^∞}
2	64	0.86 s	9.39×10^{-4}	1.15×10^{-2}	21.27 s	10.84 s	8.75×10^{-4}	1.14×10^{-2}
3	64	1.27 s	3.08×10^{-4}	1.09×10^{-2}	23.97 s	14.23 s	3.44×10^{-4}	1.10×10^{-2}
4	64	27.16 s	1.15×10^{-4}	9.42×10^{-3}	29.76 s	17.86 s	1.47×10^{-4}	9.26×10^{-3}
5	32	62.91 s	1.03×10^{-4}	1.58×10^{-2}	41.45 s	25.12 s	7.35×10^{-5}	8.02×10^{-3}
6	23	85.59 s	1.12×10^{-4}	1.75×10^{-2}	55.86 s	33.86 s	3.93×10^{-5}	4.09×10^{-3}
7	15	106.87 s	1.19×10^{-4}	3.82×10^{-2}	79.70 s	47.69 s	3.62×10^{-5}	5.34×10^{-3}
8	11	139.95 s	2.47×10^{-4}	1.08×10^{-1}	109.83 s	64.34 s	3.69×10^{-5}	3.70×10^{-3}

From :
Neural semi-Lagrangian method for high-dimensional advection-diffusion problems, 2026

`https://arxiv.org/pdf/2504.20715`

Extension to full Vlasov-Poisson at the end of the paper.

Comparison with approximation theory

Theorem (Cybenko 1989)

Let S be any sigmoid. Then finite sums of the form

$$G(\mathbf{x}) = \sum_{j=1}^{N} \alpha_j S\left(\langle \mathbf{y}_j, \mathbf{x} \rangle + \theta_j\right)$$

are dense in $C^0([0,1]^d)$ equipped with the norm of the maximum.

Theorem (Yarostsky)

There exists a ReLU-NN approximating all bounded functions belonging to $W^{n,\infty}([0,1]^d)$ with uniform accuracy ε and at most $O(\varepsilon^{-d/n} \log 1/\varepsilon)$ computational units.

Numerical results obtained with real algorithms are not well explained by such theoretical results.

Summary of that part

- SCIMBA makes Physically-Informed-Neural-Nets (just least square by the way) accessible.
- Clearly good potential and good accuracy for problems in very high dimension (thanks to MC integrator) and parametric studies.
- NN enhanced δf simulations in another work (Non-linear control variate in δf particle-in-cell methods using symplectic neural networks, Fournet, Campos-Pinto, Franck, Michel-Dansac, 2026).