Thoughts on Mathematics, Plasma Physics and Machine Learning

B. Després (LJLL-SU+MEGAVOLT-INRIA)

Two ancient aristocratic disciplines

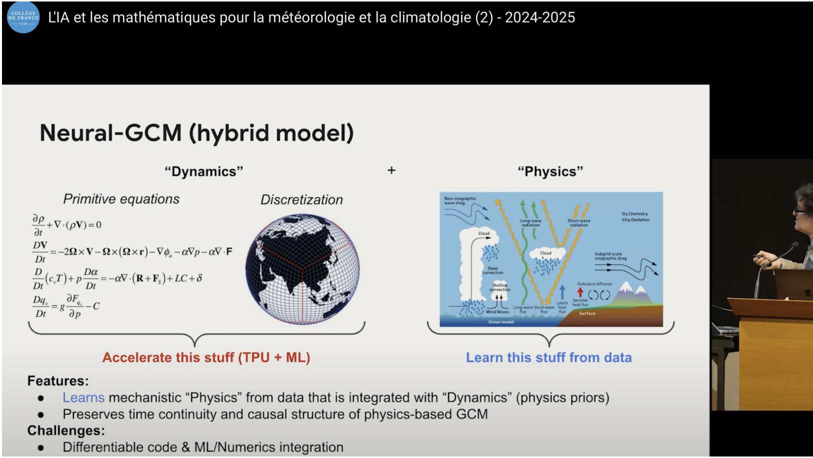
- **Math (wiki)** Earliest mathematical texts available are from :
Mesopotamia and Egypt-Plimpton 322 (Babylonian 2000-1900 BC),
the Rhind Mathematical Papyrus (Egyptian 1800 BC),
and the Moscow Mathematical Papyrus (Egyptian 1890 BC).

- **Plasma physics (wiki)**
Plasma was first identified in laboratory by Sir William Crookes. Crookes presented a lecture on what he called "radiant matter" to the British Association for the Advancement of Science, in Sheffield, on Friday, 22 August 1879.
Systematic studies of plasma began with the research of Irving Langmuir and his colleagues in the 1920s. Langmuir also introduced the term "plasma" as a description of ionized gas in 1928.

A new comer : Machine Learning (NN, IA, ...)

- DeepBlue beats Kasparov at chess in 1997.
- Classification, MNIST, NLP and more discussed in review paper : Deep learning, LeCun, Bengio, Hinton, 2015 = 95 965 GS citations ! !
- AlphaGo beats Ke Jie at Go in 2017.
- Chatbots explosion (ChatGPT, Deepseek, Mistral AI, ...)
- Nobel prize in physics 2024 : Hopfield, Hinton.
- Nobel prize in chemistry 2024 : David Baker (Seattle), Demis Hassabis and John Jumper (Google DeepMind).

Recent progress

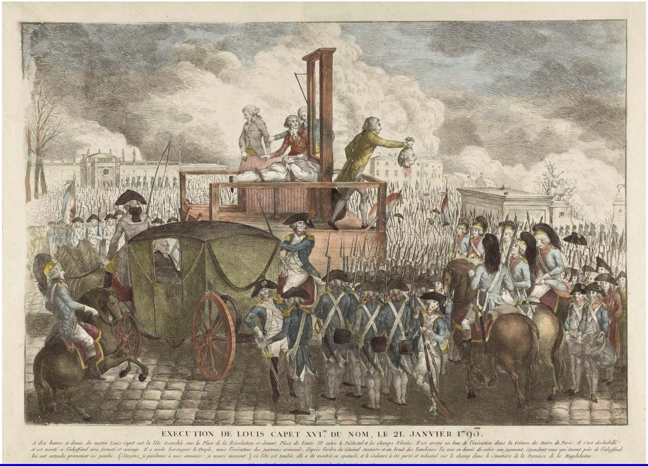


Michael Brenner (Harvard+Google)

Nowadays many theoretical works on turbulence modeling with ML/AI/NN **absolutely everywhere** : WPI, Turin, Paris, US, . . .,

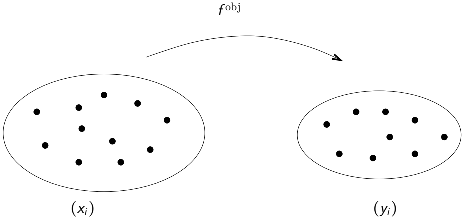
The usual aristocratic reaction

La réaction du roi est connue à l'annonce de la prise de la Bastille.
« Mais c'est une révolte ? », demande-t-il.
« Non, Sire, c'est une révolution », lui répond le duc de la Rochefoucauld-Liancourt.



Data set

- Take a large enough **dataset** : $\mathcal{D} = \{(x_i, y_i),\ i = 1, \dots\} \subset \mathbb{R}^m \times \mathbb{R}^n$



Postulate : dataset corresponds to unknown objective/transfer function

$$H : \mathbb{R}^m \longrightarrow \mathbb{R}^n$$

with $x_i \in \mathbb{R}^m$, $y_i = H(x_i) + \varepsilon_i \in \mathbb{R}^n$, and noise $\varepsilon_i \in \mathbb{R}^n$.

Least square representation with **composition**

- Take a linear function f with **weight** $W \in \mathcal{M}_{mn}(\mathbb{R})$ and **bias** $b \in \mathbb{R}^n$

$$\begin{aligned} f: \quad \mathbb{R}^m &\longrightarrow \mathbb{R}^n, \\ x &\longmapsto f(x) = Wx + b. \end{aligned} \tag{1}$$

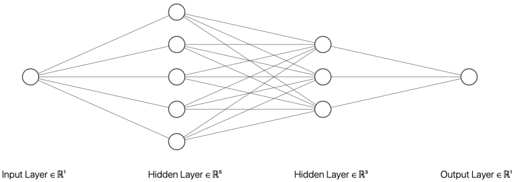
- Notations

$a_0 = m$ is the **input layer**
$a_{p+1} = n$ is the **output layer**
$(a_1, a_2, \ldots, a_p) \in \mathbb{N}^p$ are the **(dense) hidden layers** with **neurons**

- Consider

$$\begin{aligned} f_r: \quad \mathbb{R}^{a_r} &\longrightarrow \mathbb{R}^{a_{r+1}}, \\ X_r &\longmapsto f_r(X_r) = W_r X_r + b_r \end{aligned}$$

and the function $f = f_p \circ f_{p-1} \ldots f_2 \circ f_1 \circ f_0$.



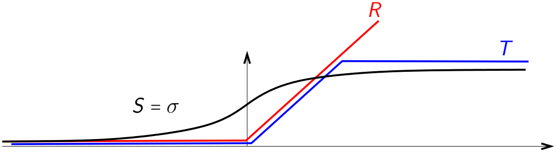
Non linearity and composition

- Non linearity is added with an **activation function**.

Sigmoid $\in C^1(\mathbb{R})$. A sigmoid σ is monotone, $0 < \sigma' < 1$, with limit value 0 at $-\infty$ and limit value 1 at $+\infty$.

ReLU $\in C^0(\mathbb{R})$. It is defined by $R(x) = \max(0, x)$.
Thresholding yields $T(x) = \min(R(x), 1)$.

Generalization component wise to activation functions $\mathbb{R}^q \to \mathbb{R}^q$.



A function f defined through a generic **feed-forward neural network** is :

$$f = f_{p+1} \circ S_{p+1} \circ f_p \circ \cdots \circ f_1 \circ S_1 \circ f_0,$$

where the activation function is either $S_r = \sigma$ or $S = R$.

Yarotsky Theorem 2017

Theorem

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

Taking $O(\varepsilon^{-d/n})$ neurons per layer, one needs $O(\log 1/\varepsilon)$ layers.

It has the flavor of magic non linear interpolation, but it is not.
It is truly linear interpolation (go to the details of the proof).

SGD, training, autodiff, ...

A generic cost function=loss function is

$$J(W) = \frac{1}{\mathrm{card}\mathcal{D}} \sum_{(x,y)\in\mathcal{D}} |f(x) - y|^2, \qquad f(x) = f_W^{\mathrm{NN}}(x).$$

Definition

SGD=stochastic gradient algorithm is an ad-hoc version of

$$W'(t) = -\nabla J(W(t)) \text{ or } W^{n+1} = W^n - \lambda \nabla J(W^n)).$$

It is used for training=minimization session on the computer with SGD.

Assume $J = v \circ u$ and $v, u \in C^1$. Then

$$\nabla J = \nabla v \circ u \ \nabla u \Longleftrightarrow \nabla J(W) = \nabla v(u(W)) \ \nabla u(W).$$

All derivatives are exactly calculated with **automatic differentiation** (Tensorflow, Pytorch, Jax, ScikiLearn, ...) based on the chain rule which is the main tool for **composition of functions**

The regularity issue/problem

ReLU activations functions (and alike) are more and more popular for some excellent reasons,
but not without contradictions.

- In practice the accuracy if often similar as for regular activation functions.
- The cost of the derivative is almost nul :
  think of the CPU cost of the calculation of $b = a \times R'(x)$.
- However it raises the issue of understanding the usual choice in softwares (Tensorflow, Pytorch, Scikitlearn, ...)

$$R'(0) = 0.$$

- Some paradoxes arise. For example

$$x = R(x) - R(-x)$$

  but it is not correct for the derivative at $x = 0$

$$1 = 0.$$

A recent result on composition of non smooth ReLU-type functions

Directly comes from **F. Murat-C. Trombetti : A chain rule formula for the composition of a vector-valued function by a piecewise smooth function. 2003.**

Theorem (Murat-Trombetti Theorem)

Consider two functions. The first one $u \in \mathrm{Lip}(\mathbb{R}^a : \mathbb{R}^b)$ is Lipschitz-continuous. The second one $v \in \mathrm{Lip}(\mathbb{R}^b : \mathbb{R}^c)$ is Lipschitz-continuous and **piecewise-C^1** with a representation with an associated gradient.

Then the chain rule identity holds in $L^\infty(\mathbb{R}^a : \mathcal{M}_{c,a}(\mathbb{R}))$

$$\nabla(v \circ u) = \widetilde{\nabla} v \circ u \ \nabla u$$

where $\widetilde{\nabla} v \circ u(x) = \widetilde{\nabla} v(u(x))$ for all $x \in \mathbb{R}^a$.

Proof rewritten in the context of Machine Learning : **D., TMLR, 2025.**

As a consequence $\widetilde{\nabla} v = \nabla v$ in $L^\infty(\mathbb{R}^a : \mathcal{M}_{c,a}(\mathbb{R}))$

Proposed to us (Ruiyang Dai+D.) by V. Grandgirard and P. Donnel

Solve numerically systems of equations like (Ji-Held 2006)

$$\partial_t f_a + \mathbf{v} \cdot \nabla f_a + \frac{e_a}{m_a} (\mathbf{E} + \mathbf{v} \wedge \mathbf{B}) \cdot \partial_{\mathbf{v}} f_a = \sum_b C(f_a, f_b)$$

where the Coulomb operators are expressed with the Rosenbluth potentials

$$C(f_a, f_b) = \frac{\gamma_{ab}}{2m_a} \partial_{\mathbf{v}} \cdot \left[\partial_{\mathbf{v}} \cdot (f_a \partial_{\mathbf{v}} \partial_{\mathbf{v}} G_b) - 2 \left(1 + \frac{m_a}{m_b} \right) f_a \partial_{\mathbf{v}} H_b \right].$$

The Rosenbluth potentials are

$$H_b = \int \frac{f_b(\mathbf{v}')}{|\mathbf{v} - \mathbf{v}'|} d\mathbf{v}' \text{ and } G_b = \int f_b(\mathbf{v}') |\mathbf{v} - \mathbf{v}'| d\mathbf{v}'.$$

On numerical grounds, the equation is written in dimension 9=3+3+3 ! !

In consequence, it is just impossible to solve as is on the computer.

Objective : use learning techniques to hopefully compress and accelerate the calculations

We intended to use linearized moment methods (Donnel et al 2018) in order to **generate synthetic datas, then to learn the Fokker-Planck operator from them.**

For many reasons, it was not satisfactory.
In particular, linearized moment methods (Donnel et al 2018) are far to be a reference method.

Therefore we move to a more basic question :
is there a best ML structure for learning such problems ?

First try

Fourier discretization ($|k| \leq |N| \Leftrightarrow k = (k_1, k_2, k_3) \in \{-N, \ldots, N\}^3$)

$$\begin{cases} f_N(v) = \sum_{|m|\leq|N|} \hat{f}_k \exp(ik \cdot v), \\ \hat{f}_k = \frac{1}{(2\pi)^2} \int_{B(0,\pi)} f(t,v) \exp(-ik \cdot v) dv, \\ \frac{\partial \hat{f}_k}{\partial t} = \sum_{|m|\leq|N|} \hat{f}_{k-m} \hat{f}_m \hat{\beta}_L(k-m, m), \qquad |k| \leq |N|. \end{cases}$$

where the coefficients $\hat{\beta}_L(m,n)$ have analytical formulas (Pareschi et al 2000).
We generate synthetic data, here extremely simple,

$$f(0,v) = \frac{v^2}{\pi\sigma^2} \exp(-\frac{v^2}{\sigma^2}). \tag{2}$$

This problem has an exact solution (Lemou 1998)

$$f(t,v) = \frac{1}{2\pi S^2} \left(2S - 1 + \frac{1-S}{2S} \frac{v^2}{\sigma^2} \right) \exp(-\frac{v^2}{2S\sigma^2}), \tag{3}$$

where $S = 1 - \exp(-\sigma^2 t/8)/2$.

We use a seemingly important idea which is to **train with time series**
TensorFlow Team. Time Series Forecasting `https://www.tensorflow.org/tutorials/structured_data/time_series`

$$\text{Minimize } \frac{1}{m_s(n_t-1)} \sum_{j=1}^{m_s} \sum_{i=0}^{n_t-1} \left\| F^{\mathcal{NN}}(i,j) - F(i+1,j) \right\|_{l^2}$$

where $F_{i,j}$ is a vector of moments $|m| \leq |N|$ for the coefficient σ_j and the time $i\Delta t$.

Then we use the trained function $F^{\mathcal{NN}}$ in our numerical solver :
it fails as shown below ! !

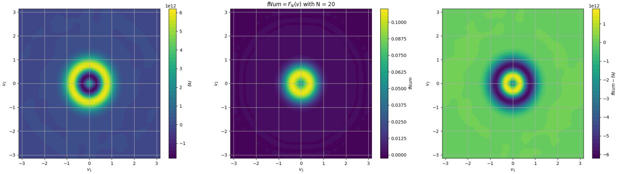


FIGURE – Left=prediction. Middle=true. Right=difference.

Second try : residual NN

$$\text{Minimize } \frac{1}{m_s(n_t-1)} \sum_{j=1}^{m_s} \sum_{i=0}^{n_t-1} \left\| F^{\mathcal{N}\mathcal{N}}(i,j) + F(i,j) - F(i+1,j) \right\|_{l^2}.$$

The map $F^{\mathcal{N}\mathcal{N}}$ serves as a residual in our iterations

$$\begin{cases} F(i+1,j) = F + F^{\mathcal{N}\mathcal{N}}(i,j), & 0 \le i < n_t, \\ \hat{f}_k^{(0)} = \dfrac{1}{(2\pi)^2} \displaystyle\int_{B(0,\pi)} f(0,v)\exp(-ik\cdot v)dv, & |k| \le |N|. \end{cases}$$

Results are much better as shown below

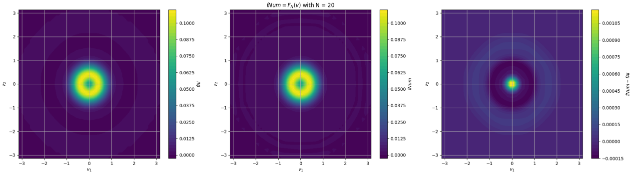


FIGURE – Left=prediction. Middle=true. Right=difference.

Proposed to us (Moreno Pintore+D.) by S. Jaouen

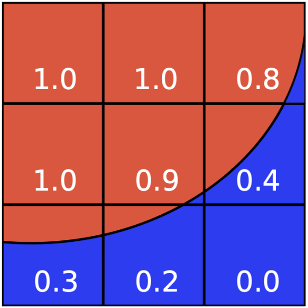
Solve transport equations the transport equation with interface datas

$$\begin{cases} \partial_t u(t,\mathbf{x}) + \mathbf{c}(\mathbf{x}) \cdot \nabla_x u(t,\mathbf{x}) = 0, & \\ u(\mathbf{x},0) = \mathbf{I}_\omega(\mathbf{x}), & \mathbf{I}_\theta \text{ is the indicatrix function of } \omega \subset \mathbb{R}^3, \\ \nabla \cdot \mathbf{c}(\mathbf{x}) = 0, & \text{for simplicity,} \end{cases}$$

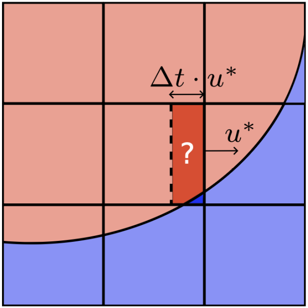
for numerical modeling of early stage of multimaterial ICF flows.

Setting of the VOFML solution

Here we train the numerical flux from 3D synthetic data on blocks of $N \times N \times N$ Finite Volume cells.



(1.a) Example of volume fractions.



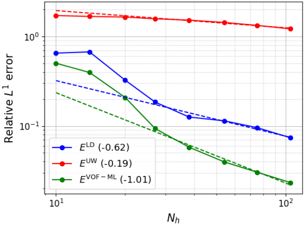
(1.b) Representation of the unknown flux.

Then we use the VOFML numerical flux in a Finite Volume solver.

Some results

$$u_1 = 2\sin(\pi x)^2 \sin(2\pi y)\sin(2\pi z)\cos(\pi t/T),$$
$$u_2 = -\sin(\pi y)^2 \sin(2\pi x)\sin(2\pi z)\cos(\pi t/T), \quad (4)$$
$$u_3 = -\sin(\pi z)^2 \sin(2\pi x)\sin(2\pi y)\cos(\pi t/T).$$

Initial condition as the indicator function of a sphere centered in $[0.35, 0.35, 0.35]$ with radius 0.15.



Theory and results in 2025 preprint
`https://hal.sorbonne-universite.fr/hal-05149322v1`

Conclusions

- ML (NN, IA) is a revolution in numerical technology,
  and so is a revolution in all applied sciences.
- The fundamental mathematical principles behind are still to be identified with certainty,
  even if almost all branches of mathematics work on it.
- Applications to transport equations show a potential in plasma physics, still to be explored and confirmed.

Allegory of the possible (but not certain) fate of scientific aristocracy confronted with ML