

# Introduction à l'IA pour les sciences

## ANF MLM4S 2024

Alexis Lechervy  
<alexis.lechervy@unicaen.fr>

12 décembre 2024

# Sommaire

1 Introduction

2 Les PINNs

3 Les librairies

# Présentation de la journée

## Présentation du formateur



- Maitre de conférence à l'Université de Caen
- Membre du laboratoire GREYC

Alexis Lechervy

## Déroulement de la journée

- 8h30 - 11h Présentation de l'IA pour les Sciences au travers des PINNs
- 15h15 - 19h30 TP de mise en pratique

# Intérêts des Réseaux de Neurones pour les Sciences

- **Modélisation de systèmes complexes** : Capture des relations non linéaires et modélisation de systèmes difficiles à formuler analytiquement.
- **Prédiction et Simulation** : Prédictions sur le comportement futur et simulation de phénomènes physiques complexes.
- **Analyse de données massives** : Identification de motifs et anomalies, analyse de grandes quantités de données expérimentales.
- **Interprétation des données expérimentales** : Identification de signaux faibles et événements rares, analyse de données complexes.
- **Optimisation** : Optimisation des paramètres de modèles physiques et amélioration des conceptions de systèmes.
- **Découverte de nouvelles lois physiques** : Révélation de relations sous-jacentes et affinement des théories existantes.

# Problème physique illustratif

## Diffusion de la chaleur

Nous allons étudier la diffusion de la chaleur, dans une plaque de métal 1D.

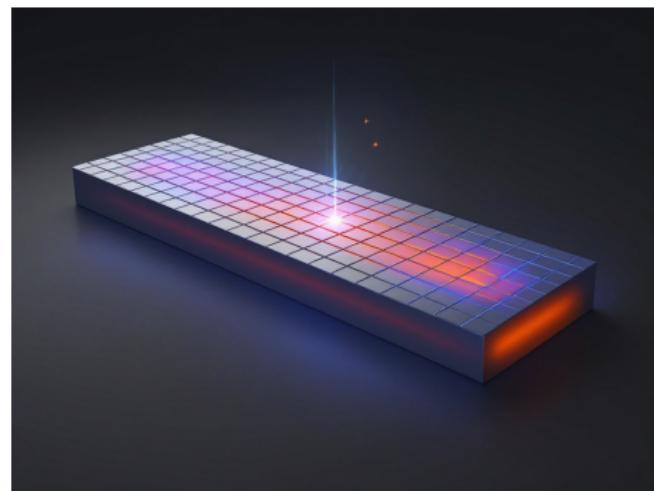
- Modélisation physique :

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

avec  $\alpha$  la diffusivité thermique.

- Condition aux bords (on considère des conditions de Dirichlet homogènes) :

$$T(0, t) = T(L, t) = 0 \quad \forall t > 0$$



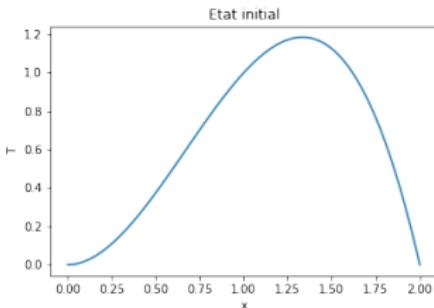
# Simulation de la diffusion

## Paramètre de notre simulation

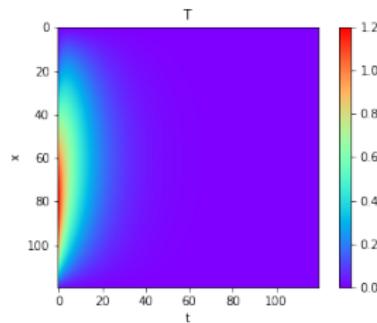
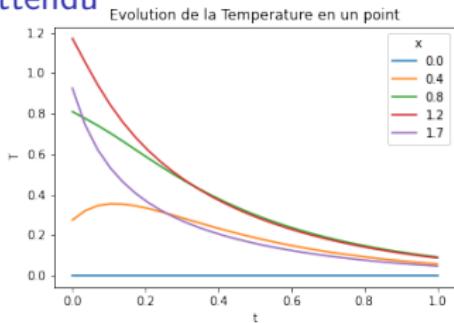
- Une barre de 2cm :  $x \in [0, 2]$
- État initial :

$$T_0 = x^2(2 - x)$$

- Une diffusion thermique  $\alpha = 2$



## Rendu attendu



# Méthode de résolution classique

## La méthode par *Séparation des variables*

- Supposons que la solution  $T(x, t)$  peut être écrite sous la forme

$$T(x, t) = X(x)T(t)$$

- En substituant dans l'équation de la chaleur, nous obtenons :

$$X(x) \frac{dT}{dt} = \alpha T(t) \frac{d^2X}{dx^2}$$

- En séparant les variables, nous obtenons :

$$\frac{1}{\alpha T} \frac{dT}{dt} = \frac{1}{X} \frac{d^2X}{dx^2} = -\lambda$$

où  $\lambda$  est une constante de séparation.

## Problème à résoudre

Pour  $X(x) : \frac{d^2X}{dx^2} + \lambda X = 0$

avec les conditions aux limites  $X(0) = 0$  et  $X(L) = 0$ .

Pour  $T(t) : \frac{dT}{dt} + \alpha \lambda T = 0$

# Résolution d'équations différentielles ordinaires

## Solution pour $X(x)$

- Les solutions de l'équation différentielle pour  $X(x)$  sont de la forme :

$$X(x) = A \cos(\sqrt{\lambda}x) + B \sin(\sqrt{\lambda}x)$$

- En appliquant les conditions aux limites  $X(0) = 0$  et  $X(L) = 0$ , nous obtenons :

$$A = 0 \quad \text{et} \quad \sin(\sqrt{\lambda}L) = 0$$

- Donc,  $\sqrt{\lambda}L = n\pi$  pour  $n = 1, 2, 3, \dots$
- Ainsi,  $\lambda_n = \left(\frac{n\pi}{L}\right)^2$  et  $X_n(x) = B_n \sin\left(\frac{n\pi x}{L}\right)$

## Solution pour $T(t)$

- Les solutions de l'équation différentielle pour  $T(t)$  sont de la forme :

$$T_n(t) = C_n e^{-\alpha \left(\frac{n\pi}{L}\right)^2 t}$$

# Solution générale

- La solution générale est une combinaison linéaire des solutions particulières :

$$u(x, t) = \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L}\right) e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}$$

- Les coefficients  $B_n$  sont déterminés en utilisant la condition initiale  $u(x, 0) = f(x)$  :

$$f(x) = \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L}\right)$$

- En utilisant la série de Fourier, nous obtenons :

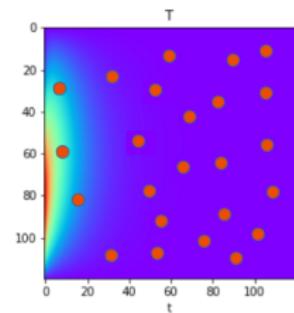
$$B_n = \frac{2}{L} \int_0^L f(x) \sin\left(\frac{n\pi x}{L}\right) dx$$

- La solution analytique de l'équation de la chaleur avec des conditions aux limites de Dirichlet homogènes est :

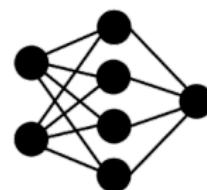
$$u(x, t) = \sum_{n=1}^{\infty} \left( \frac{2}{L} \int_0^L f(x) \sin\left(\frac{n\pi x}{L}\right) dx \right) \sin\left(\frac{n\pi x}{L}\right) e^{-\alpha\left(\frac{n\pi}{L}\right)^2 t}$$

# Simulation par apprentissage machine classique

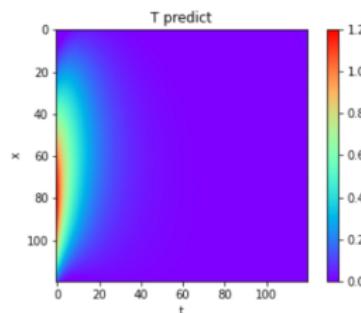
Résolution d'un problème de régression



Données  
d'apprentissage



Model



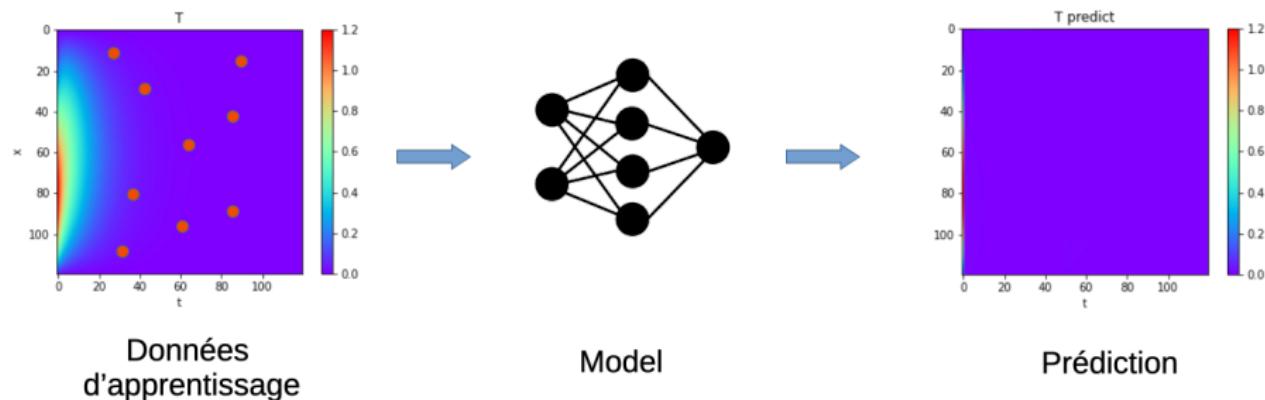
Prédiction

Chaque point de l'ensemble d'apprentissage correspond à la réalisation d'une mesure expérimentale.

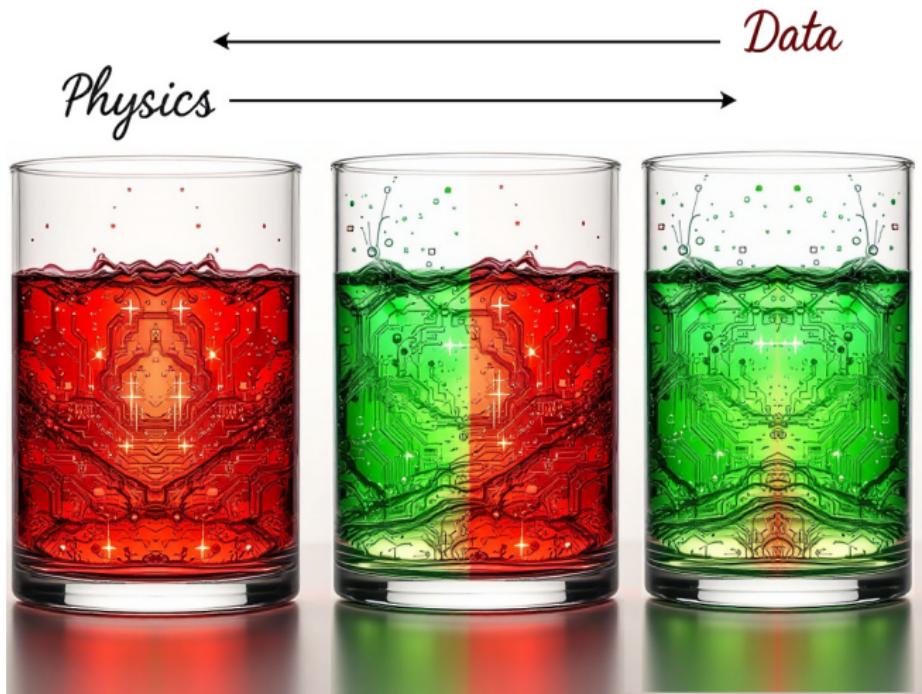
# Simulation par apprentissage machine classique

## Les limites des l'apprentissages machines

- Chaque point d'apprentissage correspond à **une mesure réelle**.
- Difficile d'avoir des grandes bases d'apprentissage.
- Les résultats ne sont pas interprétable.
- Le modèle peut trouver des solutions non physiques.



# Un apprentissage guidé par la physique



- Si peu de données, fort besoin de contraintes physiques.
- Si beaucoup de données, pas besoin de contraintes physiques.

# Comment introduire de la physique dans un réseau de neurones ?

## Type de règles physique à introduire

- Équations Différentielles Ordinaires (ODE)

Exemple :

$$\frac{dy}{dx} = y$$

- Équations Différentielles Partielles (PDE)

Exemple :

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

- Équations Différentielles Intégrales (IDE)

Exemple :

$$\frac{dy}{dx} = y(x) + \int_0^x y(t) dt$$

## Famille de méthodes

- Physics Informed neural network (PINN),
- Deep Galerkin Method,
- Feynman-Kac formula,
- Deep Ritz Method.

# Sommaire

1 Introduction

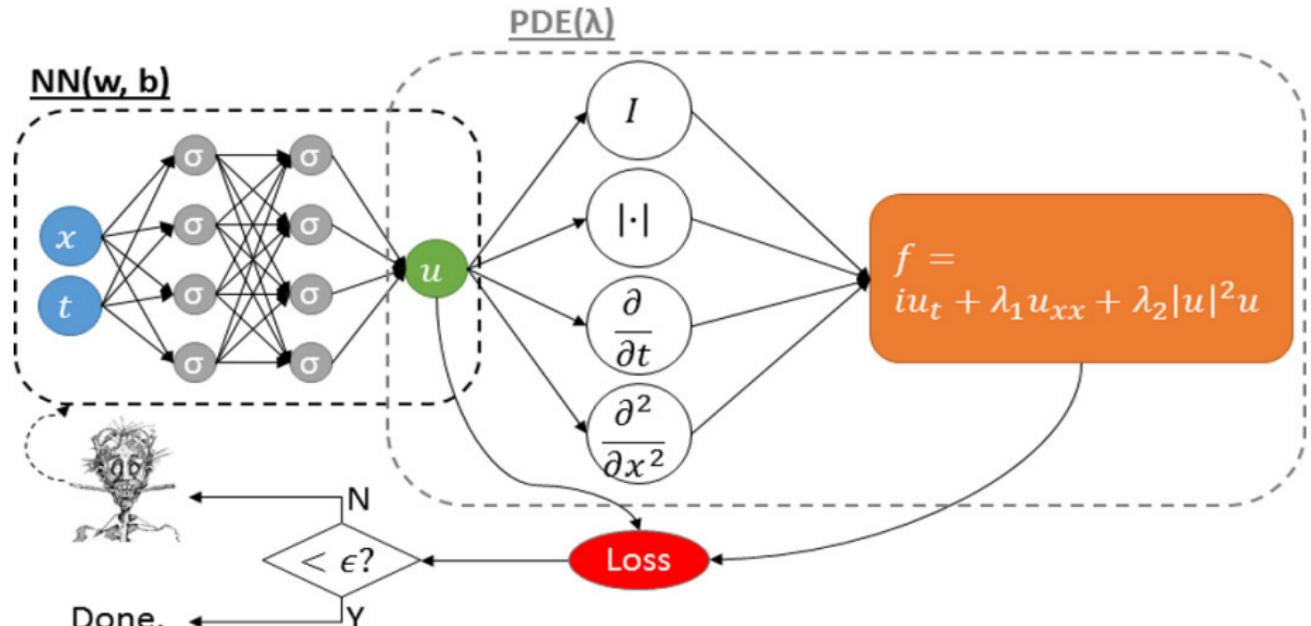
2 Les PINNs

- PINNs pour la résolution de problème direct
- PINNs pour la résolution de problème inverse

3 Les bibliothèques

# Introduction au Physics Informed neural network (PINNs)

## Principe des 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. Journal of Computational Physics 378, 686-707, 2019

# Sommaire

1 Introduction

2 Les PINNs

- PINNs pour la résolution de problème direct
- PINNs pour la résolution de problème inverse

3 Les bibliothèques

# Configuration d'un PINN pour la simulation physique

## Contraintes physiques considérées

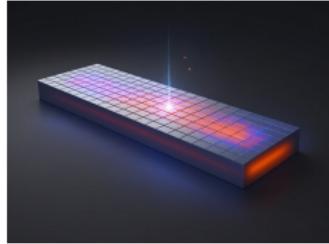
- Équations aux dérivées partielles pour l'intérieur du domaine :

$$\mathcal{L}[u](x) = f(x) \text{ sur } U \subset \mathbb{R}^d$$

- Équations aux dérivées partielles pour les bords :

$$\mathcal{B}[u](x) = g(x) \text{ sur } \partial U$$

## Sur notre exemple



- Pour l'intérieur du domaine :  $\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$
- Pour les bords et les conditions initiales :

$$T(0, t) = T(L, t) = 0 \quad \forall t > 0$$

$$T(x, 0) = x^2(2 - x)$$

# Configuration d'un PINN pour la simulation physique

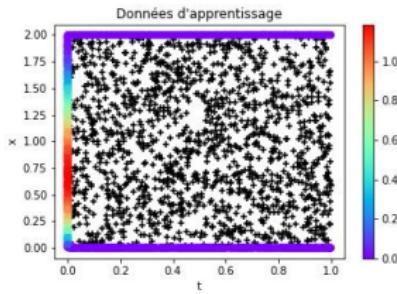
## Objectif de l'apprentissage

Trouver un réseau neuronal  $h_\theta(x)$  qui approxime la solution  $T$  de la PDE..

L'idée des PINNs : introduire la physique à l'aide des fonctions de coûts

$$\text{Loss}_{\mathbf{m}}^{\text{PINN}}(\theta) = \frac{1}{m_r} \sum_{i=1}^{m_r} \left( \underbrace{\mathcal{L}[h_\theta](\mathbf{x}_r^i)}_{\text{physics}} - \underbrace{f(\mathbf{x}_r^i)}_{\text{data}} \right)^2 + \frac{1}{m_b} \sum_{i=1}^{m_b} \left( \underbrace{\mathcal{B}[h_\theta](\mathbf{x}_b^i)}_{\text{physics}} - \underbrace{g(\mathbf{x}_b^i)}_{\text{data}} \right)^2$$

## Sur notre exemple



- Fonction de coût pour les points de collocation :  
 $\text{MSE} \left( \frac{\partial h_\theta}{\partial t} - \alpha \frac{\partial^2 h_\theta}{\partial^2 x}; 0 \right)$
- Fonction de coût pour les points aux bords :  
 $\text{MSE} (h_\theta(x_{\text{bord}}, t_{\text{bord}}); 0)$
- Fonction de coût pour les conditions initiales :  
 $\text{MSE} (h_\theta(x_0, 0); x_0^2(2 - x_0))$

# Comment coder une équation aux dérivées partielles en PyTorch ?

Coder l'équation  $\frac{\partial h_\theta(x)}{\partial x}$

```
x.requires_grad_(True)
h = self.forward(x)
h_x = torch.autograd.grad(
    u,
    x,
    torch.ones_like(u),
    retain_graph=True,
    create_graph=True
)[0]
```

# Coder un PINN en PyTorch

## Les étapes

- Créer une base de données avec les points de collocations et les points au bords.
- Définir une architecture de réseau de neurones.
- Définir les fonctions de coûts en introduisant la physique à l'aide des PDE.
- Réaliser un apprentissage par descente de gradients.

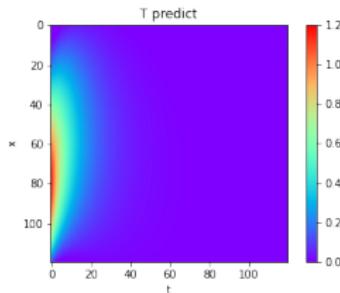


⇒ Suite dans un jupyter notebook.

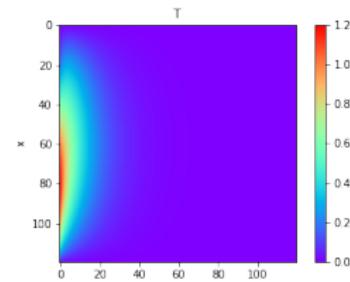
# Résultats de notre simulation

## Prédictions

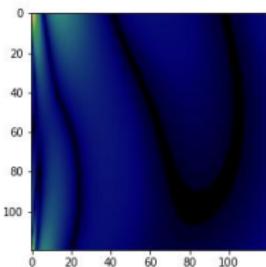
Prédictions :



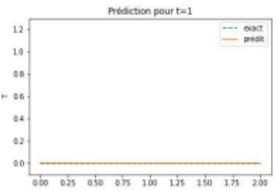
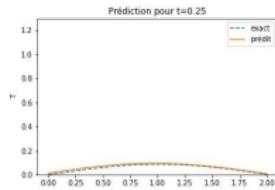
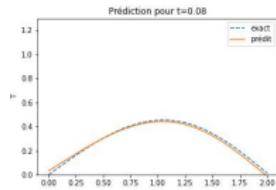
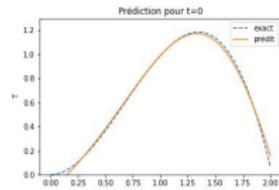
Valeurs réelles :



Erreurs :



## Prédictions à plusieurs instants



# Synthèse

## Synthèse

- On a vu comment introduire des contraintes physiques dans un réseau de neurones avec les PINNs.
- On a appliqué sur une simulation sans point de supervision.

## Avantages

- Intégration des lois physiques dans les modèles.
- Capacité à résoudre des problèmes complexes avec peu/pas de données.
- Flexibilité pour différents types d'équations différentielles.
- Facile à mettre en place.

## Inconvénients

- Complexité de l'entraînement et de l'optimisation.
- Temps de calcul pouvant être élevé / consommation de ressources.
- Nécessité de connaissances sur la physique des phénomènes analysés.
- Résultats inférieurs aux modèles classiques pour la simulation.

# Sommaire

## 1 Introduction

## 2 Les PINNs

- PINNs pour la résolution de problème direct
- PINNs pour la résolution de problème inverse

## 3 Les bibliothèques

# Problèmes inverses

## Définition

Un problème inverse consiste à déterminer les causes à partir des effets observés, contrairement à un problème direct où l'on prédit les effets à partir de causes connues.

## Exemples

- **Imagerie Médicale** : Reconstruire une image des tissus internes à partir des mesures des rayons X.
- **Sismologie** : Déterminer la structure du sous-sol à partir des enregistrements des ondes sismiques.
- **Thermodynamique** : Déterminer les propriétés thermiques d'un matériau à partir des mesures de température.

## Caractéristiques

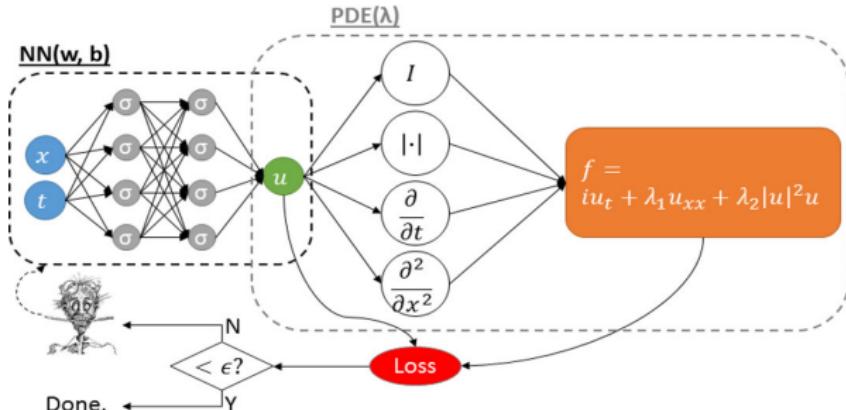
- Souvent mal posés, peuvent ne pas avoir de solution unique.
- Sensibles au bruit dans les données observées.
- Nécessitent des techniques de régularisation pour stabiliser la solution.

# Les PINNs pour les problèmes inverses

Modifications par rapport au problème direct

- Définir les inconnues du modèle physique dans la partie PDE de l'architecture.
- Minimiser un compromis entre le respect des contraintes physiques et le respect des mesures observés :

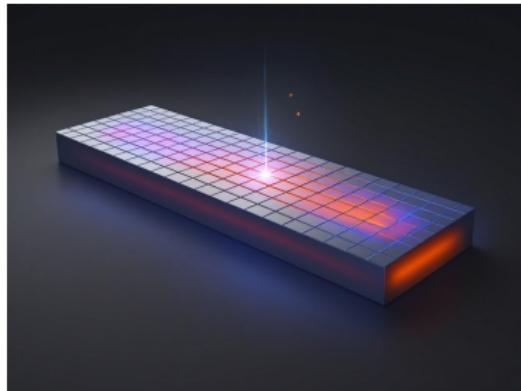
$$\text{Loss}_m(\theta) = \text{Loss}_m^{\text{PINN}}(\theta) + \frac{1}{m_e} \sum_{i=1}^{m_e} \|h_\theta(x_e^i) - y_e^i\|^2$$



# Exemple de problème inverse sur notre cas d'étude

## Problème inverse

Retrouver la valeur  $\alpha$  de la *diffusion thermique* du matériau à partir de mesurer de température sur notre plaque à différent instant.



Rappel de notre modélisation physique :

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial^2 x}$$

## Différence avec le problème direct

- La valeur de la *diffusion thermique*  $\alpha$  est inconnue et non fixé 2.
- On dispose de points de mesures (potentiellement bruités).

# Adaptation du code au problème inverse

## Modification à apporter au code précédent

- Changer le *Dataset* en ajoutant les points de mesures.
- Modifier la fonction de coût pour ajouter un terme d'attache aux données expérimentales.
- Remplacer la valeur de  $\alpha$  par un objet *Parameter* qui sera appris.



⇒ Suite dans le jupyter notebook précédent.

# Sommaire

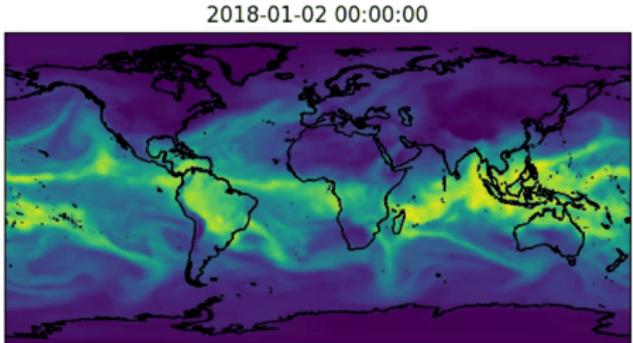
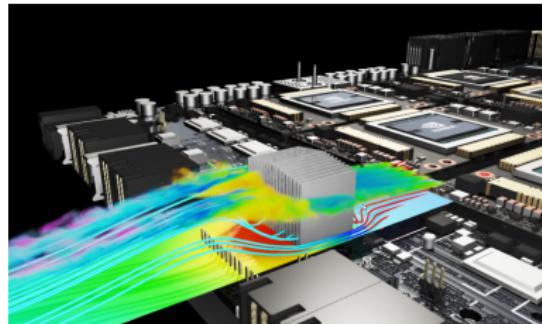
1 Introduction

2 Les PINNs

3 Les librairies

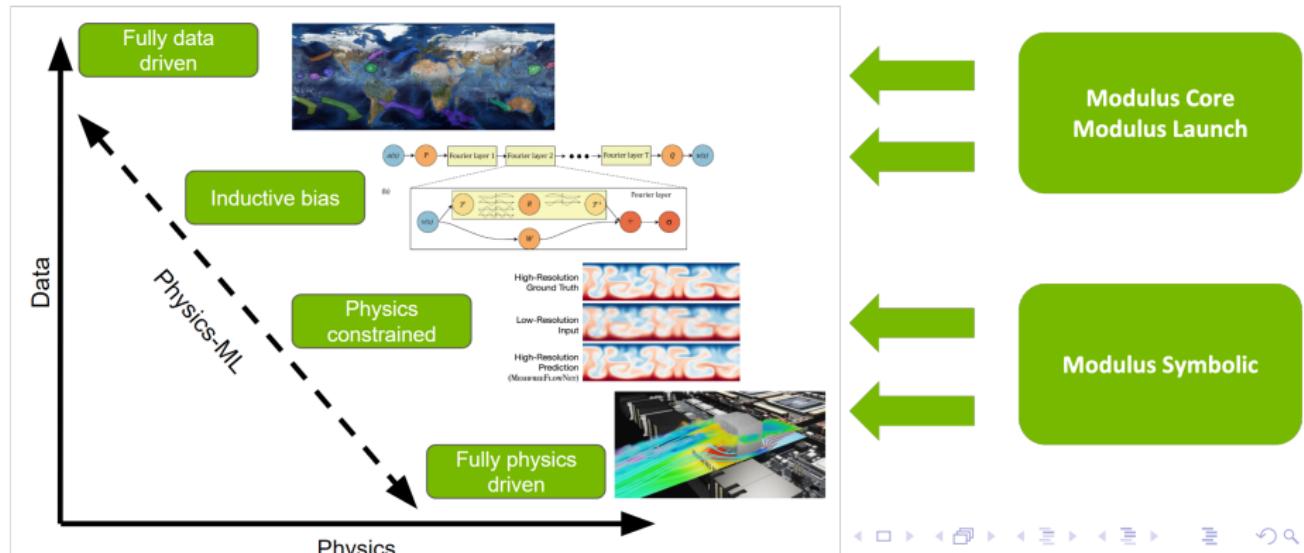
# Les librairies pour les PINNs

- DeepXDE <https://deepxde.readthedocs.io/en/latest/>
- NVIDIA Modulus <https://developer.nvidia.com/modulus>
- NeuralPDE.jl <https://github.com/SciML/NeuralPDE.jl>
- ...



# NVIDIA Modulus

- Outil permettant d'élaborer des solutions **basées sur des données** pour résoudre des problèmes d'ingénierie.
- Framework pour la **Réolution d'PDE** direct ou inverse.
- Outil d'optimisation efficace de la conception pour les problèmes multi-physiques.
- Outil pour la création de jumeaux numériques.

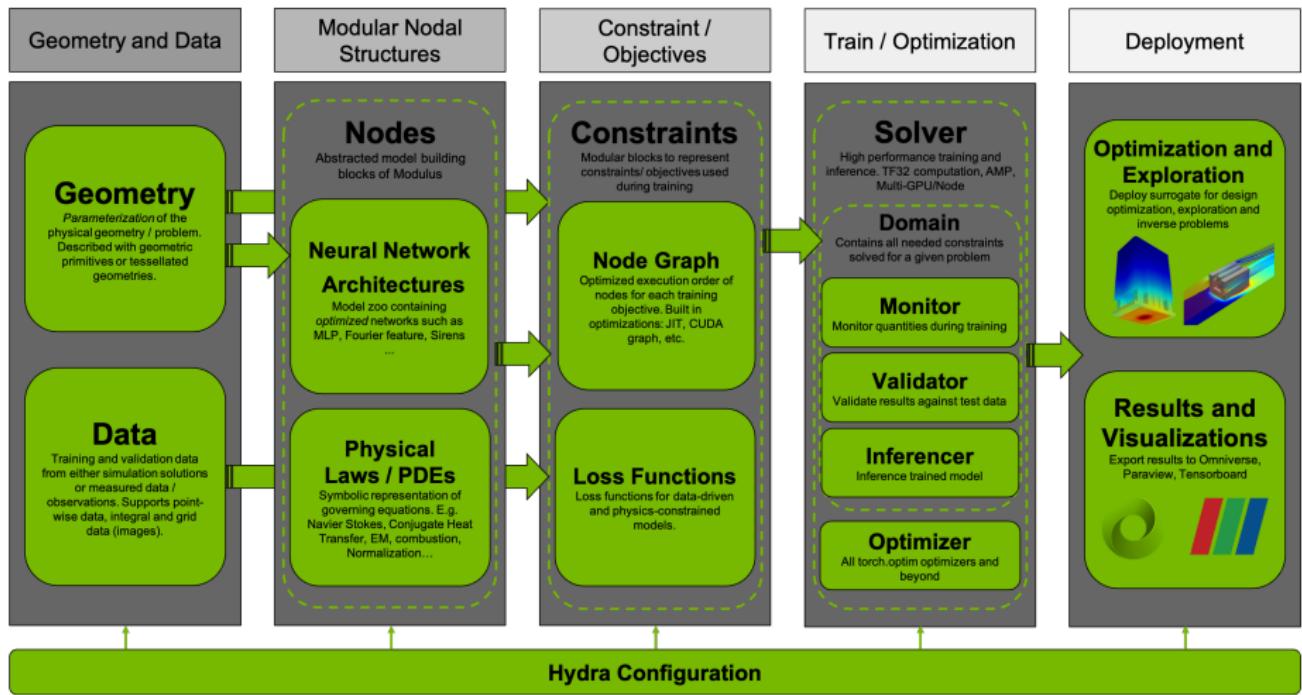


# NVIDIA Modulus Sym

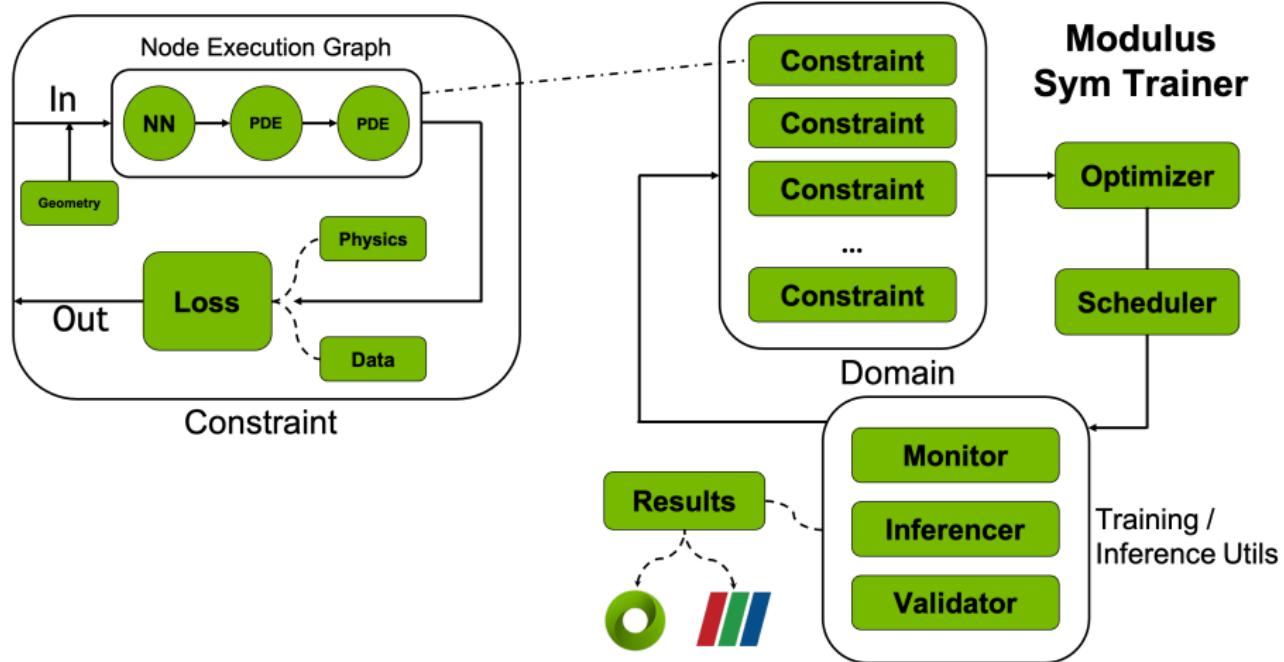
## Elements clés

- API de haut niveau pour la définition de problème physique par des experts du domaine
- Calculs automatisés des dérivées et des fonctions de coûts pour l'apprentissage des PINNs à l'aide de SymPy
- Configurable via des fichiers yaml via Hydra

# Architecture de NVIDIA Modulus Symbolic



# Apprentissage d'un modèle via Modulus



# Définition d'un réseau de neurones

```
net = FullyConnectedArch(  
    input_keys=[Key("x"), Key("y")],  
    output_keys=[Key("u")],  
    nr_layers=3,  
    layer_size=32  
)  
  
nodes = [net.make_node(name="network")]
```

# Utilisation de SymPy pour définir les équations

## Principes

Les équations physiques utilisées sont définies à l'aide de la librairie python SymPy.

## Exemple

```
# Définir les symboles
x = Symbol('x')
t = Symbol('t')
c = Symbol('c')

# Définir l'équation de la chaleur
u = Function('u')(x, y)
heat_eq = u.diff(y) - c**2 * u.diff(x, 2)
```

# Définitions des données d'apprentissage (1/2)

## Définition du domaine

```
# Definition du domaine
rec = Rectangle((0,0), (2,1))
ldc_domain = Domain()

# Extraction de points du domaine
samples_b = rec.sample_boundary(100,
    quasirandom=True
)
samples_i = rec.sample_interior(100,
    quasirandom=True
)
```

## Définitions des données d'apprentissage (2/2)

Définition de l'ensemble d'apprentissage contraint par la physique

```
left_wall = PointwiseBoundaryConstraint(  
    nodes=nodes,  
    geometry=rec,  
    outvar={"u": 0.0},  
    batch_size=100,  
    criteria=Eq(x, 0),  
)  
ldc_domain.add_constraint(left_wall, "left_wall")  
  
ldc_domain.add_constraint(  
    PointwiseInteriorConstraint(  
        nodes=nodes,  
        geometry=rec,  
        outvar={'heat_eq': 0.0},  
        batch_size=100,  
, 'interior')
```

# Définition d'une zone d'inférence

```
# add inferencer
inference = PointwiseInferencer(
    nodes=nodes,
    invar={"x": xx.reshape(-1,1),
            "y": yy.reshape(-1,1)}
    },
    output_names=["u"]
)
ldc_domain.add_inferencer(inference, "inf_data")
```

# Définition du solveur

## Définition du solveur

```
cfg = compose(config_path=". " , config_name="config")  
cfg.network_dir = 'outputs'
```

*# Définir le solveur*

```
solver = Solver(cfg=cfg , domain=ldc_domain)
```

*# Entrainer le modèle*

```
solver.solve()
```

# Configuration

## Fichier yaml de configuration

```
defaults:  
  - modulus_default  
  - scheduler: tf_exponential_lr  
  - optimizer: adam  
  - loss: sum  
  
scheduler:  
  decay_rate: 0.95  
  decay_steps: 200  
  
save_filetypes: "vtk,npz"  
  
training:  
  max_steps: 10000
```