Machine Learning

2024 (ML-2024)

Lecture 14. Physics-informed neural networks

by Alexei Valerievich Kornaev, Dr. habil. in Eng. Sc.,
Researcher at the RC for AI, Assoc. Prof. of the Robotics and CV Master's Program, Innopolis University
Researcher at the RC for AI, National RC for Oncology n.a. NN Blohin
Professor at the Dept. of Mechatronics, Mechanics, and Robotics, Orel State University

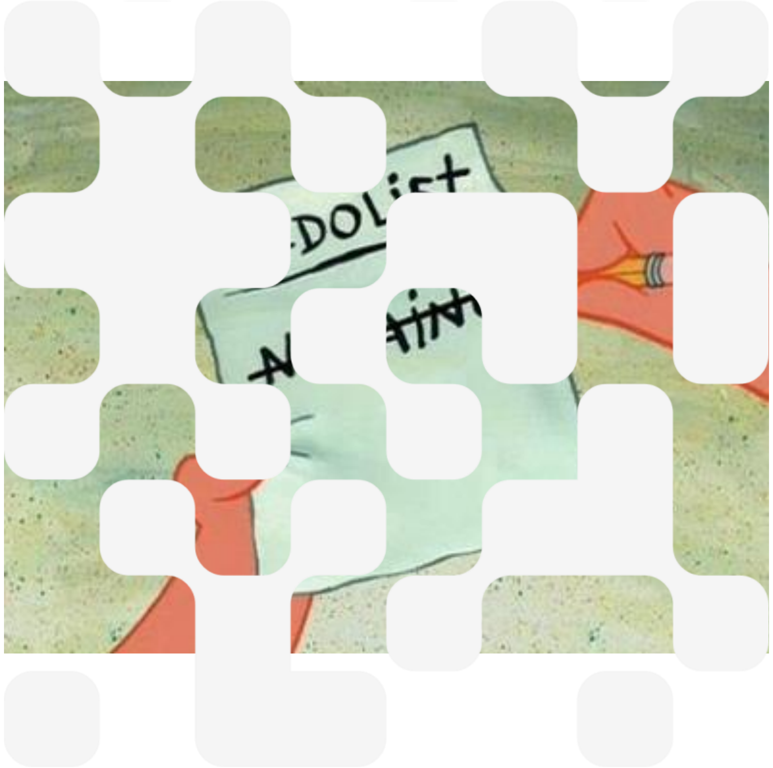
Agenda

I. INTRODUCTION TO PHYSICS-INFORMED NEURAL NETWORKS (PINNs)

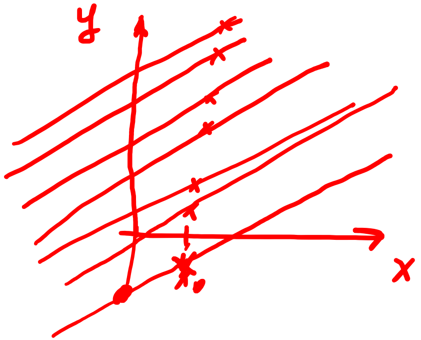
II. MODELS WITH ORDINARY DIFFERENTIAL EQUATIONS (ODEs)

III. MODELS WITH PARTIAL DIFFERENTIAL EQUATIONS (PDEs)

IV. PROSPECTS IN THE FIELD OF PINNS



Recap: what does it mean to solve a differential equation?



$$\frac{dy}{dx} = 5$$

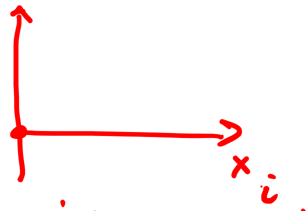
$$\int dy = \int 5\, dx$$

$$\underline{y = 5x + C}$$

$y = 5x + C$ → init. cond
→ bound. cond

$x = 0$, $y(0) = 0$

$$\frac{dy}{dx} \approx \frac{y^{i+1} - y^{i}}{\underbrace{x^{i+1} - x^{i}}_{h}}$$



$i = 0$: $y^{i+1} = 5h + y^{i}$

y^1

$i = 1$, y^2

$\vdots$

Function

$x \leftrightarrow y$

$y = 5x$

Functional

$y(x) \leftrightarrow J$

$J[y(x)] = J$

$$J = \int_{x_0}^{x_1} y(x)\, dx$$

Operator

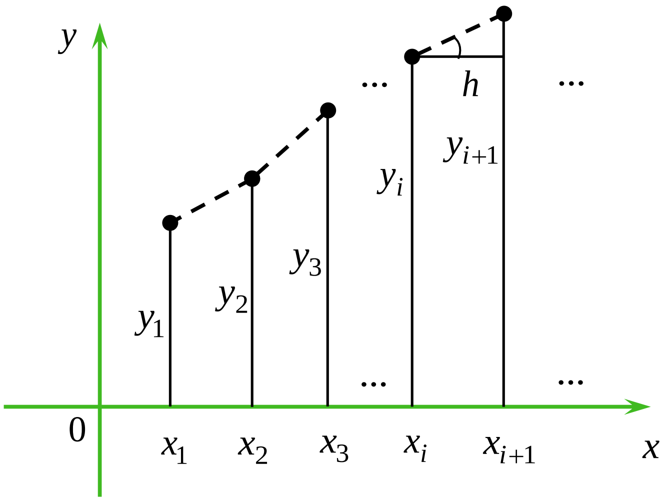
$y(x) = z(x)$

$$\frac{dy}{dx}$$

$$\int y(x)\, dx$$

Recap: what does it mean to solve a differential equation numerically?

Метод Эйлера



$$\frac{y_{i+1} - y_i}{h} = f(x_i, y_i);$$

$$\boxed{y_{i+1} = y_i + f(x_i, y_i)h;}$$

1) $y_1 = y_a$ $\quad$ (*начальное условие*);

2) $y_2 = y_1 + f(x_1, y_1)h;$

3) $y_3 = y_2 + f(x_2, y_2)h;$

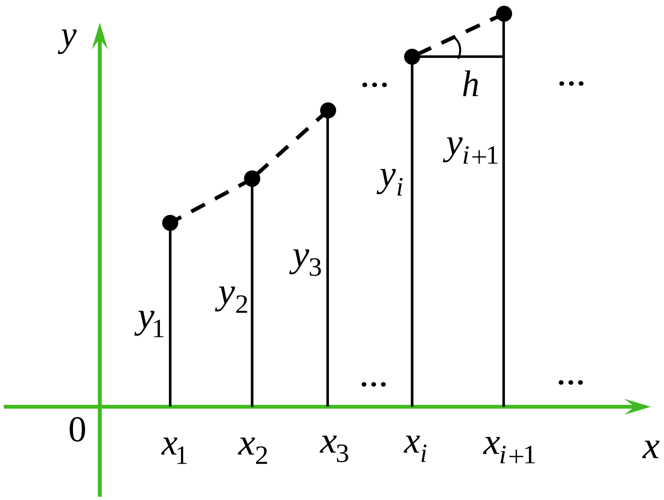
...

n) $y_n = y_{n-1} + f(x_{n-1}, y_{n-1})h.$

Recap: what does it mean to solve a differential equation numerically?

Методы Рунге-Кутта

$$\frac{dy}{dx} = f(x,y), \;\; y(0) = y_0, x \in [0; X]$$



Идея: замена искомого решения несколькими членами разложения в ряд Тейлора:

$$y_{i+1} = y(x_i) + \frac{y^{/}(x_i)}{1!}h + \frac{y^{//}(x_i)}{2!}h^2 + \frac{y^{///}(x_i)}{3!}h^3 + \ldots$$

Для нахождения y_{i+1} необходимо вычислить 4 числа:

$$m_1 = f(x_i, y_i);$$

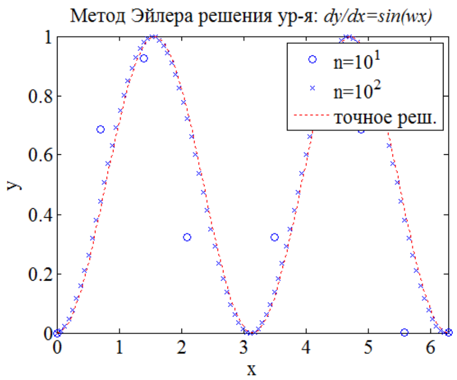
$$m_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}m_1\right);$$

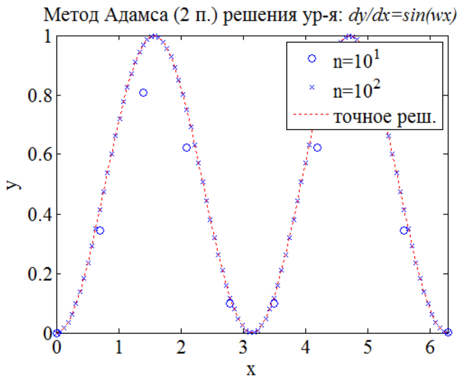
$$m_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}m_2\right);$$

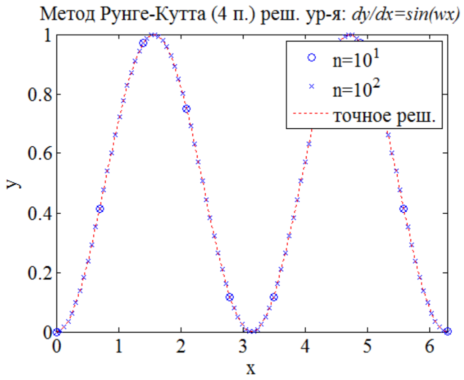
$$m_4 = f(x_i + h, y_i + hm_3);$$

после воспользоваться формулой: $\boxed{y_{i+1} = y_i + \frac{h}{6}(m_1 + 2m_2 + 2m_3 + m_4).}$

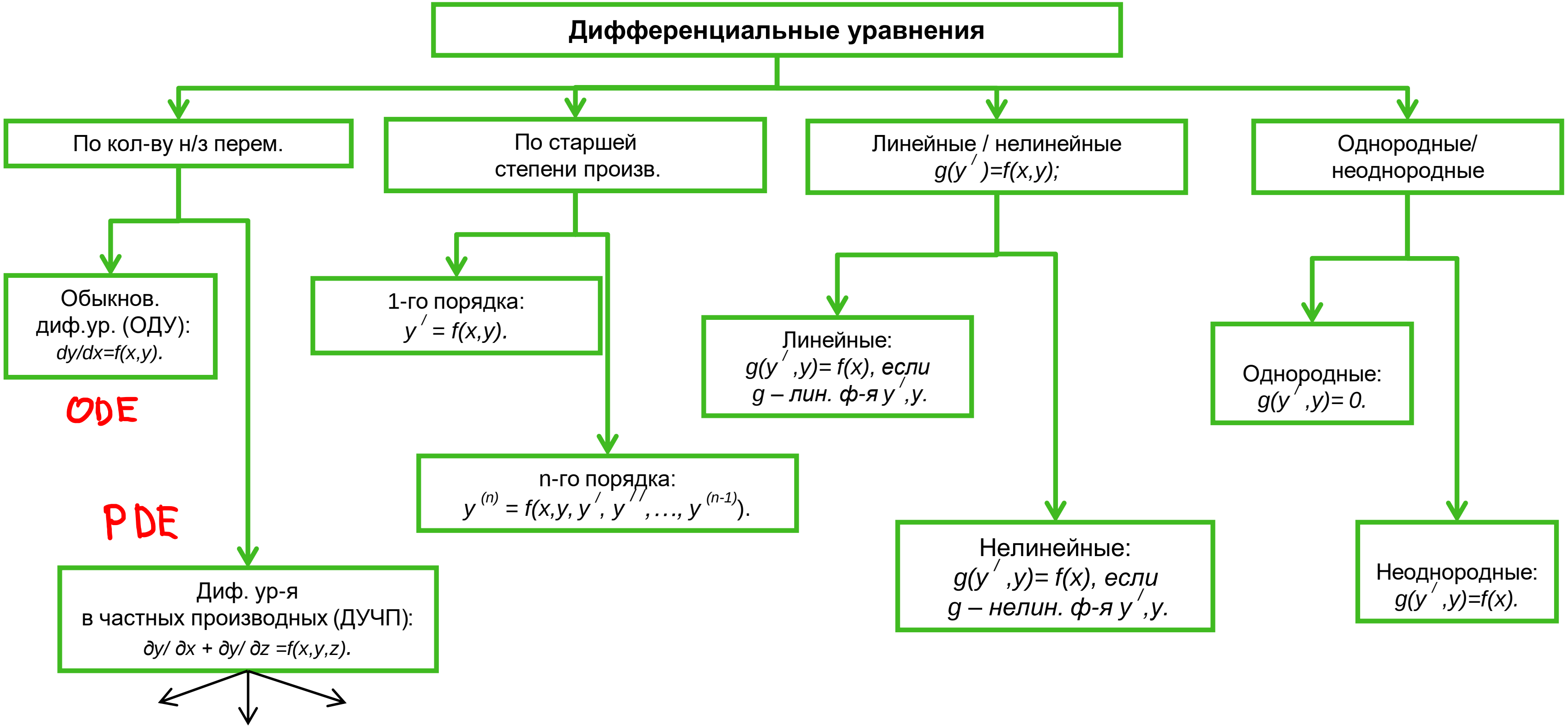
Recap: what does it mean to solve a differential equation numerically?





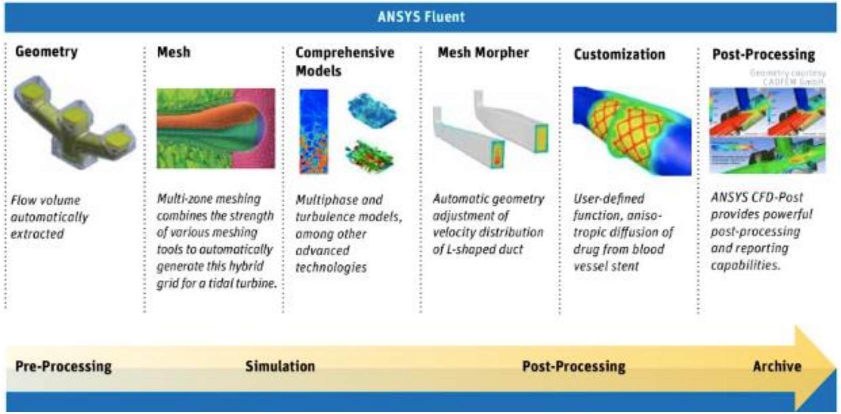


Recap: classificatioin of differential equations



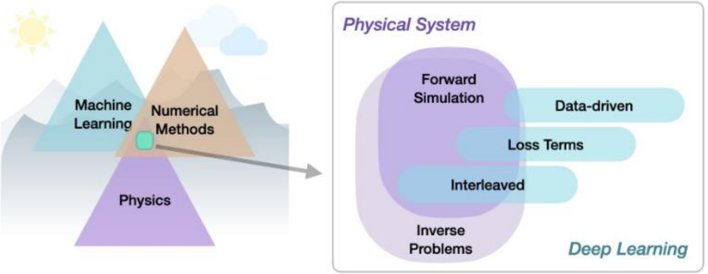
CAE-systems vs machine learning

Physics and CAE-systems



PDE

Machine learning



+ Prospects:

almost any problem can be solved;

- Limitations:

the software is expensive;
it is hard to use and verify the obtained results.

Ansys Fluent

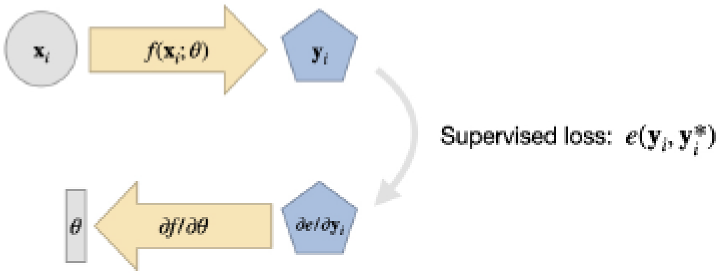
N. Thuerey et al. *Physics-based Deep Learning* . WWW, 2021

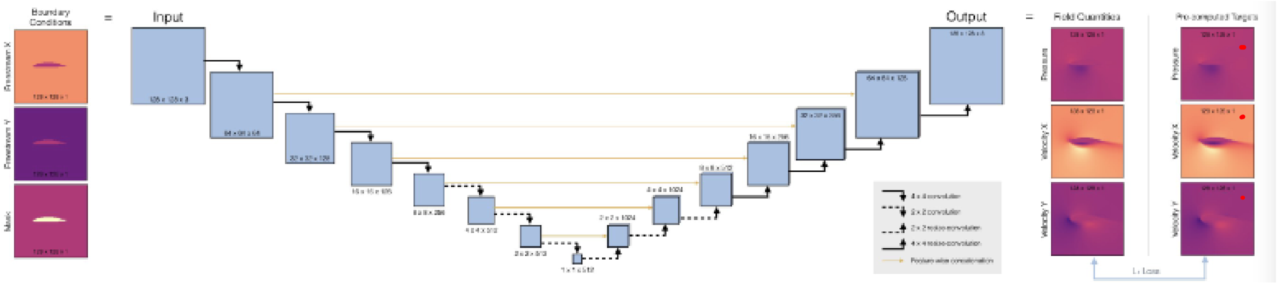
PINN

Physics-informed machine learning: supervised training (diff. eqs. participation level: 0 %)

Given m pairs of data: $\left((\boldsymbol{x}_i, y_i^*)\right)$.
In supervised learning we approximate $\boldsymbol{y}^*$ using function $f = f(\boldsymbol{x}_i,\ \boldsymbol{\theta})$ $(i = 1, \ldots, m)$ by minimizing loss:

$$\boxed{\arg\min_\theta \sum_i (f(\boldsymbol{x}_i,\ \boldsymbol{\theta}) - y_i^*)^2.}$$





p

$\vec{v}$

v_1

v_2

N. Thuerey et al. *Physics-based Deep Learning* . WWW, 2021

Physics-informed machine learning: physical loss terms (DEs participation level: up to 100 %)

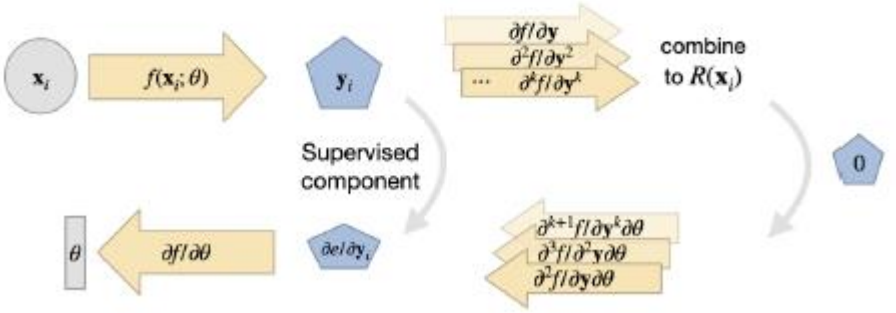
Given PDE:

$$\frac{\partial \boldsymbol{u}}{\partial t} = \boldsymbol{F}\left(\frac{\partial \boldsymbol{u}}{\partial x}, \frac{\partial^2 \boldsymbol{u}}{\partial x^2}, \dots, \frac{\partial^n \boldsymbol{u}}{\partial x^n}\right),$$

with unknown function: $\boldsymbol{u} = \boldsymbol{u}(\boldsymbol{x},\ t)$.

Residual $\boldsymbol{R}$ should be equal to zero: $\boldsymbol{R} = \frac{\partial \boldsymbol{u}}{\partial t} - \boldsymbol{F}\left(\frac{\partial \boldsymbol{u}}{\partial x}, \frac{\partial^2 \boldsymbol{u}}{\partial x^2}, \dots, \frac{\partial^n \boldsymbol{u}}{\partial x^n}\right).$

$$\boxed{\arg\min_\theta \alpha_0 \sum_i \left(f(x_i;\theta) - y_i\right)^2 + \alpha_1 R(x_i),}$$



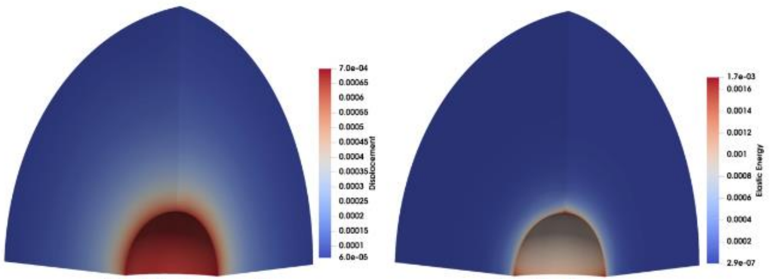
N. Thuerey et al. *Physics-based Deep Learning*. WWW, 2021

Physics-informed machine learning: physical loss (a conservation law: 100 %)

4.2. Deep Energy Method

The main idea of the method advocated in this contribution is to take advantage of the variational (energetic) structure of some BVPs. To that end, the energy of the system is used as the loss function for the DNN, as proposed by [14]. Due to its mechanical flavor, we name it the Deep Energy Method (DEM) here. One of the key ingredients is to approximate the energy of the body by a weighted sum of the energy density at integration points. Then, the following form for the loss function, $\mathcal{L}(\boldsymbol{p})$ is obtained:

$$\boxed{\mathcal{E}[\boldsymbol{u}_p] \approx \mathcal{L}(\boldsymbol{p}) = \sum_i \Psi\big(\epsilon(\boldsymbol{u}_p(\boldsymbol{x}_i))\big) w_i,} \tag{12}$$



(a) Displacement. (b) Strain energy.

```
net_uv(self,x,y,vdelta):
X = tf.concat([x,y],1)
uv = self.neural_net(X,self.weights,self.biases)
uNN = uv[:,0:1]
vNN = uv[:,1:2]
u = (1-x)*x*uNN
v = y*(y-1)*vNN
return u, v
```

Samaniego E. et al. An Energy Approach to the Solution of Partial Differential Equations in Computational Mechanics via Machine Learning Concepts, Implementation and Applications. Computer Methods in Applied Mechanics and Engineering, 2019

Good news: AI libraries (Pytorch, TF) can calculate derivatives very well

Optional Reading - Vector Calculus using `autograd`

Mathematically, if you have a vector valued function $\vec{y} = f(\vec{x})$, then the gradient of $\vec{y}$ with respect to $\vec{x}$ is a Jacobian matrix J:

$$J = \left(\frac{\partial \mathbf{y}}{\partial x_1} \quad \cdots \quad \frac{\partial \mathbf{y}}{\partial x_n} \right) = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

Generally speaking, `torch.autograd` is an engine for computing vector-Jacobian product. That is, given any vector $\vec{v}$, compute the product $J^T \cdot \vec{v}$

If $\vec{v}$ happens to be the gradient of a scalar function $l = g(\vec{y})$:

$$\vec{v} = \left(\frac{\partial l}{\partial y_1} \quad \cdots \quad \frac{\partial l}{\partial y_m} \right)^T$$

then by the chain rule, the vector-Jacobian product would be the gradient of l with respect to $\vec{x}$:

$$J^T \cdot \vec{v} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \begin{pmatrix} \frac{\partial l}{\partial y_1} \\ \vdots \\ \frac{\partial l}{\partial y_m} \end{pmatrix} = \begin{pmatrix} \frac{\partial l}{\partial x_1} \\ \vdots \\ \frac{\partial l}{\partial x_n} \end{pmatrix}$$

This characteristic of vector-Jacobian product is what we use in the above example; `external_grad` represents $\vec{v}$.

[Autograd tutorial]

python Copy

```python
import torch

# Define the input tensor
x = torch.tensor([1.0, 2.0, 3.0, 4.0], requires_grad=True)

# Define the weight tensor
w = torch.tensor(2.0, requires_grad=True)

# Define the bias tensor
b = torch.tensor(1.0, requires_grad=True)

# Define the model
y = w * x + b

# Define the target values
target = torch.tensor([3.0, 5.0, 7.0, 9.0])

# Compute the Mean Squared Error loss
loss = torch.mean((y - target) ** 2)

# Compute gradients
loss.backward()

# Print gradients
print("Gradient of w:", w.grad)
print("Gradient of b:", b.grad)
```

Copy

```
Gradient of w: tensor(2.5)
Gradient of b: tensor(1.)
```

[Deep Seek codes]

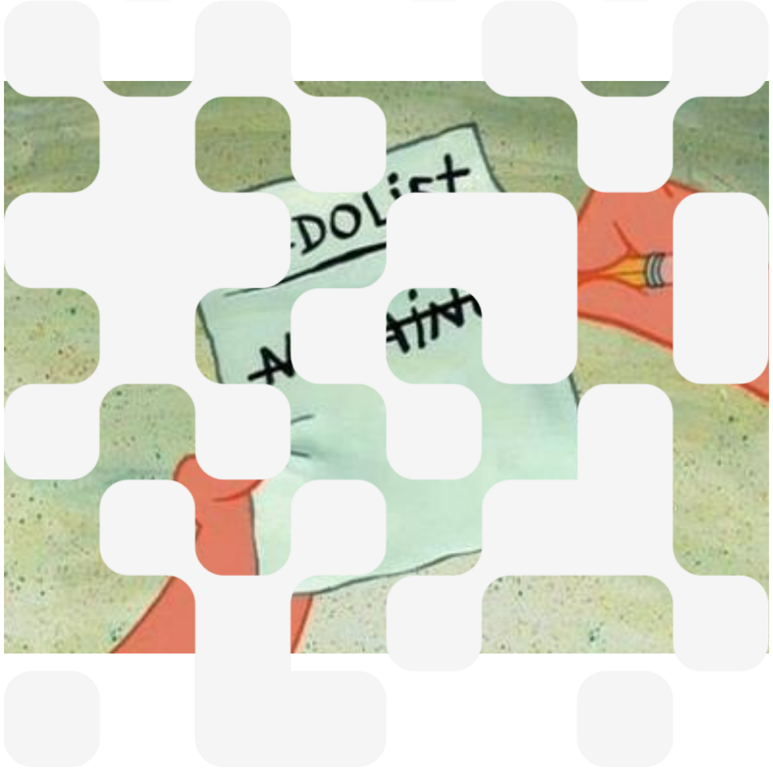
Agenda

I. INTRODUCTION TO PHYSICS-INFORMED NEURAL NETWORKS (PINNs)

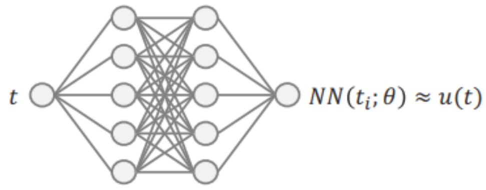
II. MODELS WITH ORDINARY DIFFERENTIAL EQUATIONS (ODEs)

III. MODELS WITH PARTIAL DIFFERENTIAL EQUATIONS (PDEs)

IV. PROSPECTS IN THE FIELD OF PINNS



ANNs for ODEs solution intuition



We are interested in modelling the displacement of the mass on a spring (green box) over time.

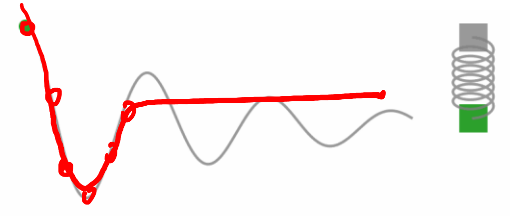
This is a canonical physics problem, where the displacement, $\boldsymbol{u(t)}$, of the oscillator as a function of time can be described by the following differential equation:

$$m\frac{d^2u}{dt^2} + \mu\frac{du}{dt} + ku = 0\,,$$

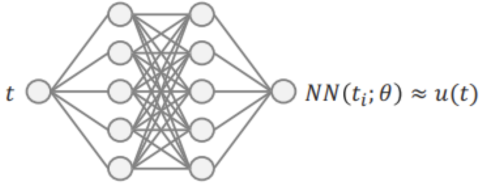
where $\boldsymbol{m}$ is the mass of the oscillator, $\boldsymbol{\mu}$ is the coefficient of friction and $\boldsymbol{k}$ is the spring constant.

$$\boxed{L(\theta) = \frac{1}{N}\sum_{i}^{N}\left(NN(t_i;\theta) - u_i\right)^2}$$

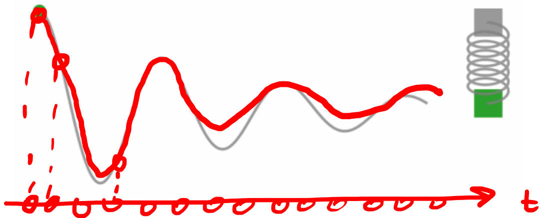
MSE



PINNs for ODEs solution intuition



$$L(\theta) = \frac{1}{N}\sum_{i}^{N}\left(NN(t_i;\theta) - u_i\right)^2 + \frac{\lambda}{M}\sum_{j}^{M}\left(\left[m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k\right]NN(t_j;\theta)\right)^2$$



[ETH Zürich DLSC: Physics-Informed Neural Networks - Applications]

From a ML perspective:

- Physics loss is an **unsupervised** regulariser, which adds prior knowledge

From a mathematical perspective:

- PINNs provide a way to solve PDEs:
  - Neural network is a mesh-free, **functional** approximation of PDE solution
  - Physics loss is used to assert solution is **consistent** with PDE
  - Supervised loss is used to assert boundary/initial conditions, to ensure solution is **unique**

PINNs training algorithm

Training loop:

1. Sample boundary/ physics training points
2. Compute network outputs
3. Compute 1st and 2nd order gradient of network output **with respect to network input** <= (recursively) apply autodiff, extending graph
4. Compute loss
5. Compute gradient of loss function **with respect to network parameters** <= apply autodiff on extended graph
6. Take gradient descent step

```
# PINN training psuedocode

#2.
t.requires_grad_(True)# tells PyTorch to start tracking graph
theta.requires_grad_(True)
u = NN(t, theta)

#3.
dudt = torch.autograd.grad(u, t, torch.ones_like(u),
                           create_graph=True)[0]
d2udt2 = torch.autograd.grad(dudt, t, torch.ones_like(u),
                             create_graph=True)[0]

#4.
physics_loss = torch.mean((m*d2udt2 + mu*dudt + k*u)**2)
loss = physics_loss + lambda_*boundary_loss

#5.
dtheta = torch.autograd.grad(loss, theta)[0]
```

[ETH Zürich DLSC: Physics-Informed Neural Networks - Applications]()

Hadns on session: please join via the link

We are interested in modelling the displacement of the mass on a spring (green box) over time.

This is a canonical physics problem, where the displacement, $u(t)$, of the oscillator as a function of time can be described by the following differential equation:

$$m\frac{d^2u}{dt^2} + \mu\frac{du}{dt} + ku = 0\,,$$

where m is the mass of the oscillator, μ is the coefficient of friction and k is the spring constant.

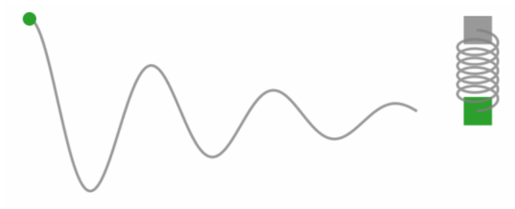
We will focus on solving the problem in the **under-damped state**, i.e. where the oscillation is slowly damped by friction (as displayed in the animation above).

Mathematically, this occurs when:

$$\delta < \omega_0\,, \quad \text{where } \delta = \frac{\mu}{2m}\,,\ \omega_0 = \sqrt{\frac{k}{m}}\,.$$

Furthermore, we consider the following initial conditions of the system:

$$u(t=0) = 1\ ,\ \frac{du}{dt}(t=0) = 0\,.$$



For this particular case, the exact solution is known and given by:

$$u(t) = e^{-\delta t}(2A\cos(\phi + \omega t))\,, \quad \text{with } \omega = \sqrt{\omega_0^2 - \delta^2}\,.$$

For a more detailed mathematical description of the harmonic oscillator, check out this blog post: https://beltoforion.de/en/harmonic_oscillator/.

ETH Zürich DLSC: Physics-Informed Neural Networks - Applications

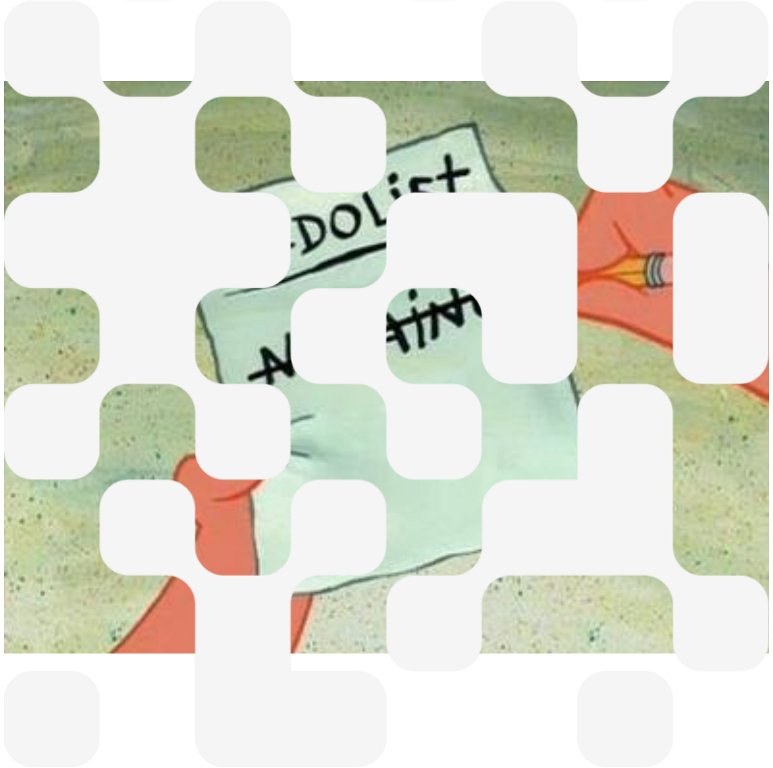
Agenda

I. INTRODUCTION TO PHYSICS-INFORMED NEURAL NETWORKS (PINNs)

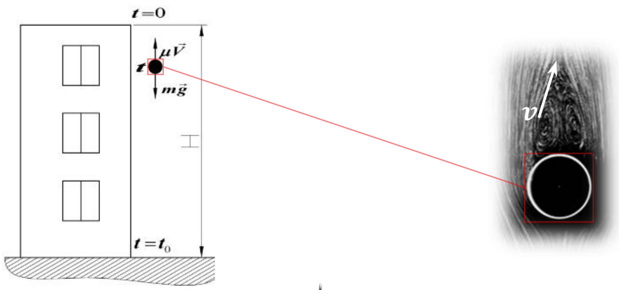
II. MODELS WITH ORDINARY DIFFERENTIAL EQUATIONS (ODEs)

III. MODELS WITH PARTIAL DIFFERENTIAL EQUATIONS (PDEs)

IV. PROSPECTS IN THE FIELD OF PINNS



Recap: Kinematics → Dynamics ← Statics



VS

Mechanics
studies *motion* of a material point

$$m\frac{d\boldsymbol{v}}{dt} = -b\boldsymbol{v} + m\boldsymbol{g},$$

declares the *mass conservation law*

$$\frac{dm}{dt} = 0.$$

Mechanics of continua
studies *motion* of the media

$$\rho\frac{d\boldsymbol{v}}{dt} = \nabla\cdot\boldsymbol{T}_\sigma + \rho\boldsymbol{f},$$

declares the mass conservation law in the form of the *continuity equation*

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho\boldsymbol{v}) = 0.$$

Nomenclature:

$\nabla = \left[\frac{\partial}{\partial x_i}\right]$ *is the Hamiltonian;*

$\nabla\cdot\boldsymbol{y} = \left[\frac{\partial y_i}{\partial x_i}\right], \nabla\cdot\boldsymbol{T}_y = \left[\frac{\partial y_{ik}}{\partial x_i}\right]$ *is the divergence;*

ρ *is the density;*

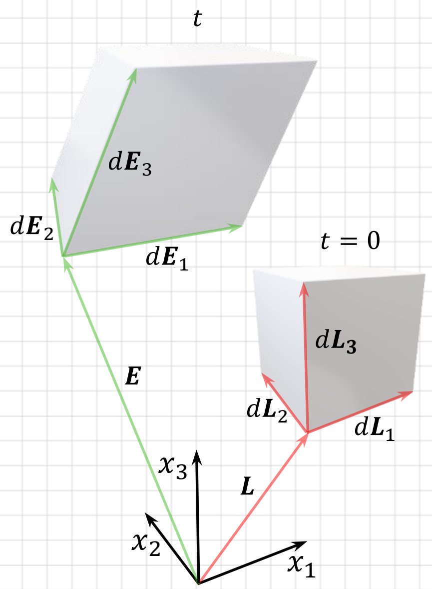
$\boldsymbol{v} = [v_i]$ *is the velocity;*

t *is time;*

$\boldsymbol{T}_\sigma = [\sigma_{ik}]$ *is the stress tensor;*

$\boldsymbol{f}$ *is the mass force (e.g. gravity).*

Recap: Kinematics → Dynamics ← Statics



Matter (mass) conservation law:

$$\frac{d\boldsymbol{L}}{dt} = 0.$$

Displacement:

$$\boldsymbol{u} = \boldsymbol{E} - \boldsymbol{L}.$$

Velocity:

$$\boldsymbol{v} = \frac{d\boldsymbol{E}}{dt}.$$

Stokes formula:

$$\boldsymbol{T}_\xi = \frac{1}{2}(\nabla\otimes\boldsymbol{v} + \boldsymbol{v}\otimes\nabla).$$

Nomenclature:

$\boldsymbol{L}, \boldsymbol{E}$ *are Lagrangian and Eulerian coord.;*

$\boldsymbol{T}_\xi = [\xi_{ik}]$ *is the strain rate tensor;*

$\boldsymbol{T}_a = D_a + S_a$ *is the decompositioin of the tensor into it's deviator and spherical part* $S_a = a_0\boldsymbol{T}_\delta$, $a_0 = a_{ii}/3$;

$\nabla y = \left[\frac{\partial y}{\partial x_i}\right]$ *is the gradient of a scalar func.*

$\nabla\otimes\boldsymbol{y} = \left[\frac{\partial y_k}{\partial x_i}\right]$ *is the gradient of a vector func.*

Recap: Kinematics → Dynamics ← Statics

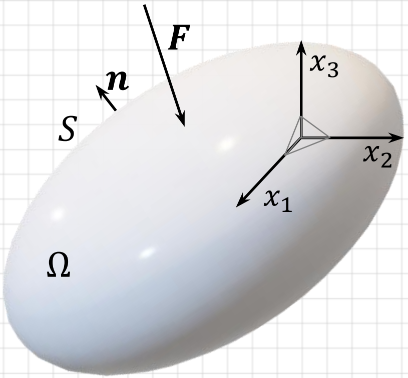
The domain Ω with surface *S* which is characterized by a unit outer normal vector *n* is under study.

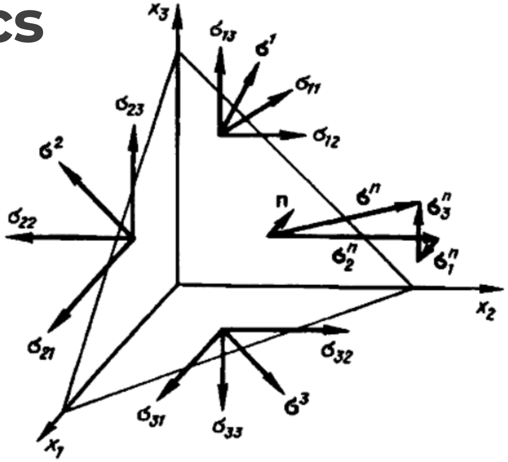
Full stress:

$$\boldsymbol{\sigma}^n = \frac{d\boldsymbol{F}}{dS}.$$

Cauchy formula:

$$\boldsymbol{\sigma}^n = \boldsymbol{n} \cdot \boldsymbol{T}_\sigma.$$





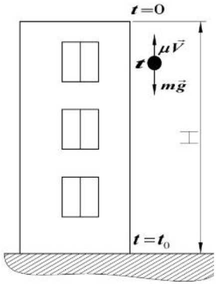
Nomenclature:

$\boldsymbol{P}$ *is the total outer force;*

$\boldsymbol{T}_\sigma = [\sigma_{ik}]$ *is the stress tensor;*

$\boldsymbol{T}_a = D_a + S_a$ *is the decompositioin of the tensor into it's deviator and spherical part* $S_a = a_0 \boldsymbol{T}_\delta$, $a_0 = a_{ii}/3$.

Recap: Kinematics → Dynamics ← Statics



Mechanics
studies *motion* of a material point

Mechanics of continua
studies *motion* of the media

Taking the Newton hypothesis into account $\boldsymbol{D}_\sigma = 2\mu \boldsymbol{D}_\xi$,
and the tensor decomposition rule, supposing that the fluid is incompressible $\rho = \mathrm{const}$,
the motion law transforms into the almost kinematic form – the Navier-Stokes equation

$$\rho \frac{d\boldsymbol{v}}{dt} = \nabla \cdot \boldsymbol{T}_\sigma + \rho \boldsymbol{f},$$

→

$$\rho \frac{d\boldsymbol{v}}{dt} = \nabla \sigma_0 + \nabla \cdot \big(\mu(\nabla \otimes \boldsymbol{v} + \boldsymbol{v} \otimes \nabla)\big) + \rho \boldsymbol{f},$$

declares the *mass conservation law*

$$m = const.$$

declares the mass conservation law in the form of the *continuity equation*

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{v}) = 0.$$

→

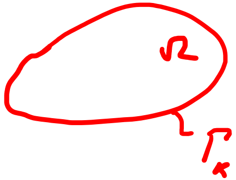
$$\nabla \cdot \boldsymbol{v} = 0.$$

PINNs for PDEs solution intuition

Given a PDE and its boundary/initial conditions

Ex. NS →

$$\mathcal{D}[u(x)] = f(x), \qquad x \in \Omega \subset \mathbb{R}^d$$
$$\mathcal{B}_k[u(x)] = g_k(x), \qquad x \in \Gamma_k \subset \partial\Omega$$



Where $\mathcal{D}$ is some differential operator, $\mathcal{B}_k$ are a set of boundary operators, and $u(x)$ is the solution to the PDE

PINNs train a neural network to **approximate** the solution to the PDE $NN(x;\theta) \approx u(x)$ using the following loss function:

$$L(\theta) = L_b(\theta) + L_p(\theta)$$

$$L_b(\theta) = \sum_k \frac{\lambda_k}{N_{bk}} \sum_j^{N_{bk}} \left\| \mathcal{B}_k[NN(x_{kj};\theta)] - g_k(x_{kj}) \right\|^2$$ **Boundary loss**

$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left\| \mathcal{D}[NN(x_i;\theta)] - f(x_i) \right\|^2$$ **Physics loss**

[ETH Zürich DLSC: Physics-Informed Neural Networks - Applications]

PINNs for PDEs solution intuition

Given a PDE and its boundary/initial conditions

$$\mathcal{D}[u(x)] = f(x), \qquad x \in \Omega \subset \mathbb{R}^d$$
$$\mathcal{B}_k[u(x)] = g_k(x), \qquad x \in \Gamma_k \subset \partial\Omega$$

Where $\mathcal{D}$ is some differential operator, $\mathcal{B}_k$ are a set of boundary operators, and $u(x)$ is the solution to the PDE

PINNs train a neural network to **approximate** the solution to the PDE $NN(x;\theta) \approx u(x)$ using the following loss function:

$$L(\theta) = L_b(\theta) + L_p(\theta)$$

$$L_b(\theta) = \sum_k \frac{\lambda_k}{N_{bk}} \sum_j^{N_{bk}} \left\| \mathcal{B}_k[NN(x_{kj};\theta)] - g_k(x_{kj}) \right\|^2$$ **Boundary loss**

$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left\| \mathcal{D}[NN(x_i;\theta)] - f(x_i) \right\|^2$$ **Physics loss**

For example, the 1+1D viscous Burgers' equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0$$

$$u(x,0) = -\sin(\pi x)$$
$$u(-1,t) = u(+1,t) = 0$$

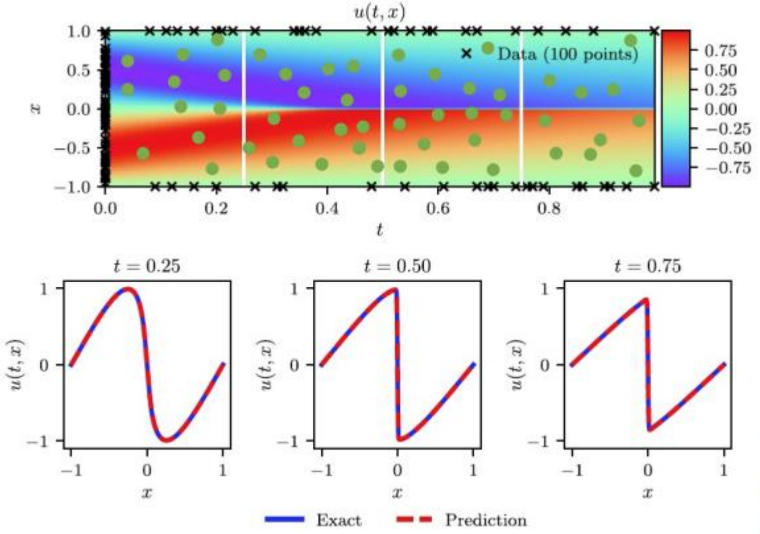
$$NN(x,t;\theta) \approx u(x,t)$$

$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_j^{N_{b1}} \left(NN(x_j,0;\theta) + \sin(\pi x_j)\right)^2 + \frac{\lambda_2}{N_{b2}} \sum_k^{N_{b2}} \left(NN(-1,t_k;\theta) - 0\right)^2 + \frac{\lambda_3}{N_{b3}} \sum_l^{N_{b3}} \left(NN(+1,t_l;\theta) - 0\right)^2$$

$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left(\left(\frac{\partial NN}{\partial t} + NN\frac{\partial NN}{\partial x} - \nu\frac{\partial^2 NN}{\partial x^2} \right)(x_i, t_i;\theta) \right)^2$$

[ETH Zürich DLSC: Physics-Informed Neural Networks - Applications]

PINNs for PDEs solution intuition



Raissi et al, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, JCP (2018)

$$L_b(\theta) = \frac{\lambda_1}{N_{b1}} \sum_j^{N_{b1}} \left(NN(x_j, 0; \theta) + \sin(\pi x_j)\right)^2$$
$$+ \frac{\lambda_2}{N_{b2}} \sum_k^{N_{b2}} \left(NN(-1, t_k; \theta) - 0\right)^2$$
$$+ \frac{\lambda_3}{N_{b3}} \sum_l^{N_{b3}} \left(NN(+1, t_l; \theta) - 0\right)^2$$
$$L_p(\theta) = \frac{1}{N_p} \sum_i^{N_p} \left(\left(\frac{\partial NN}{\partial t} + NN \frac{\partial NN}{\partial x} - \nu \frac{\partial^2 NN}{\partial x^2} \right) (x_i, t_i; \theta) \right)^2$$

$\nu = 0.01/\pi$
$N_p = 10{,}000$ (Latin hypercube sampling)
$N_{b1} + N_{b2} + N_{b3} = 100$
Fully connected network with 9 layers, 20 hidden units (3021 free parameters)
Tanh activation function
L-BFGS optimiser

[ETH Zürich DLSC: Physics-Informed Neural Networks - Applications]

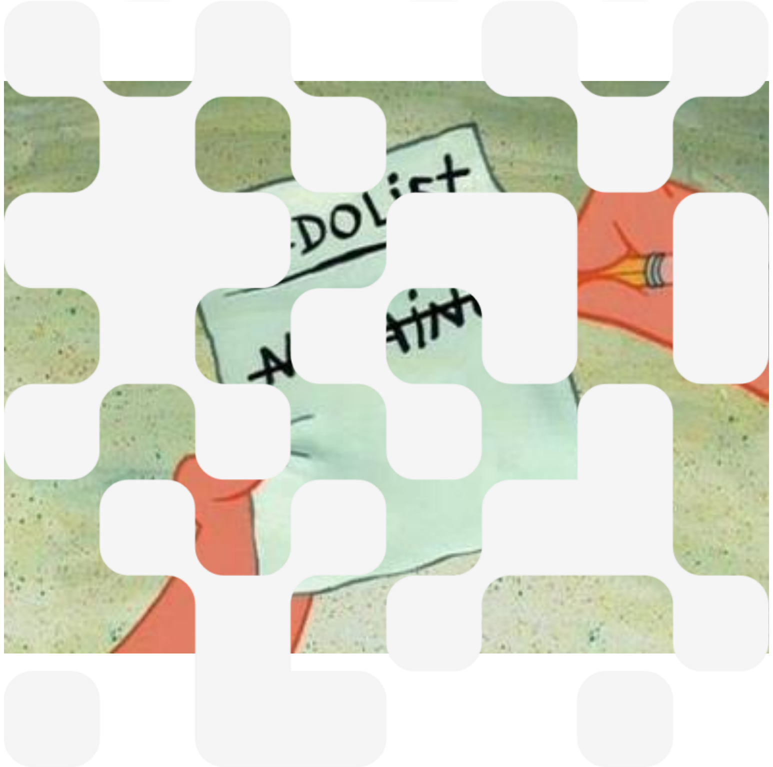
Agenda

I. INTRODUCTION TO PHYSICS-INFORMED NEURAL NETWORKS (PINNs)

II. MODELS WITH ORDINARY DIFFERENTIAL EQUATIONS (ODEs)

III. MODELS WITH PARTIAL DIFFERENTIAL EQUATIONS (PDEs)

IV. PROSPECTS IN THE FIELD OF PINNS



PINNs for equation discovery in a hadns on session: link

How do we learn an **entire** differential operator $\mathcal{D}$?

Build a **library** of n operators, such as:

$$\phi = (1, \partial_x, \partial_t, \partial_{xx}, \partial_{tt}, \partial_{xt})^T$$

Then assume the differential operator can be represented as

$$\mathcal{D} = \Lambda\phi$$

Where Λ is a (sparse) matrix of shape (d_u, n)

E.g. for 1D damped harmonic oscillator:

$$\mathcal{D} = \begin{pmatrix} k & \mu & m & 0 \end{pmatrix} \begin{pmatrix} 1 \\ d_t \\ d_{tt} \\ d_{ttt} \end{pmatrix}$$

$$= m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k$$

ETH Zürich DLSC: Physics-Informed Neural Networks - Applications

PINNs for equation discovery in a hadns on session: link

How do we learn an **entire** differential operator $\mathcal{D}$?

Build a **library** of n operators, such as:

$$\phi = (1, \partial_x, \partial_t, \partial_{xx}, \partial_{tt}, \partial_{xt})^T$$

Then assume the differential operator can be represented as

$$\mathcal{D} = \Lambda\phi$$

Where Λ is a (sparse) matrix of shape (d_u, n)

E.g. for 1D damped harmonic oscillator:

$$\mathcal{D} = (k \quad \mu \quad m \quad 0)\begin{pmatrix} 1 \\ d_t \\ d_{tt} \\ d_{ttt} \end{pmatrix}$$

$$= m\frac{d^2}{dt^2} + \mu\frac{d}{dt} + k$$

PINNs for **equation discovery**:

$$L(\theta, \Lambda) = L_p(\theta, \Lambda) + L_d(\theta)$$

$$L_p(\theta, \Lambda) = \frac{1}{N_p}\sum_{i}^{N_p} \|\Lambda\phi[NN(x_i;\theta)]\|^2 + \|\Lambda\|^2 \quad \text{Physics loss}$$

$$L_d(\theta) = \frac{\lambda}{N_d}\sum_{l}^{N_d} \|NN(x_l;\theta) - u_l\|^2 \quad \text{Data loss}$$

Where Λ are treated as **learnable** parameters and $\{x_l, u_l\}$ are a set of (potentially noisy) observational data

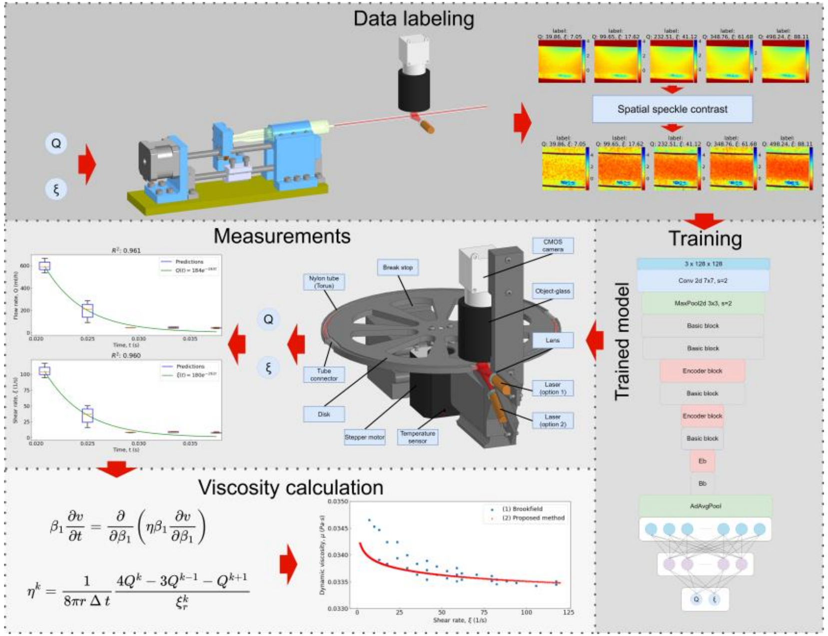
Typically, some regularization / prior on Λ (e.g. sparsity) is needed, as this optimisation problem can be very **ill-posed**

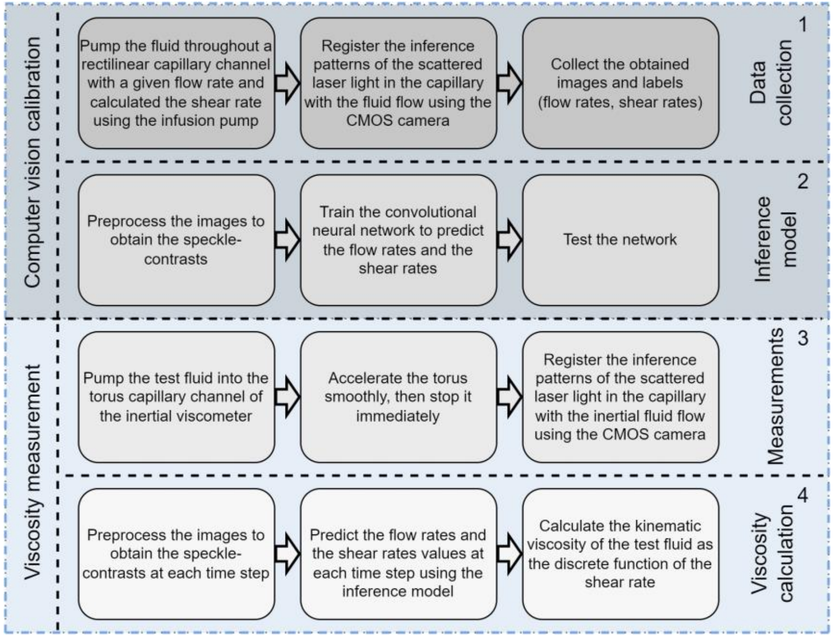
ETH Zürich DLSC: Physics-Informed Neural Networks - Applications

PINNs for equation discovery: viscosity measurement

Viscosity measurement method using inertial viscometer with a computer vision system

ξ^k, Q^k are the share rate on the fluid flow and the flow rate at each time t^k, respectively are obtained by test rig with CV system





[A method to measure non-Newtonian fluids viscosity using inertial viscometer with a computer vision system, 2023]

PINNs with energy (power) loss

Save energy (power)

$$I[? \ldots] = \int f(?, \ldots) \rightarrow min.$$

Save time

(brachistochrone task)

$$T = I[y(x)] = \int_a^b f\left(x, y(x), \frac{dy(x)}{dx}\right) \rightarrow min.$$

Save lives (Dido task)

$$S = I[y(x)] \rightarrow max, L = const.$$

PINNs with energy (power) loss

ML for physical problems as Physics-based learning

Physics-based learning [1]

Supervised learning

Data-driven
Fitting problem

$$\text{MSE} = \sum (u(\boldsymbol{\theta}, \boldsymbol{X}) - u_{true})^2 \rightarrow min$$

Data set:
- experimental data
- CFD data

Physics-informed learning

PINN
Using physical models as PDE:
$$u_t = \mathcal{F}(u, u_x, u_{xx})$$
put $\mathcal{R} = u_t - \mathcal{F}(u, u_x, u_{xx}) = 0$

$$J = MSE + \lambda\mathcal{R} \rightarrow min$$

step 1. Solve $J = MSE \rightarrow min \Rightarrow \boldsymbol{\theta}^*$
step 2. $\boldsymbol{\theta}^0 = \boldsymbol{\theta}^*$ for $J = \mathcal{R} \rightarrow min$

$$J = \mathcal{R} \rightarrow min$$

In General case:
$$J = MSE + \lambda_r\mathcal{R} + \lambda_{bc}\mathcal{R}_{bc} + \lambda_{ic}\mathcal{R}_{ic} \rightarrow min$$
Data set:
- experimental or CFD data*
- Boundary conditions
- Initial conditions

Power loss

$$J_L^* = \int_\Omega (\Pi_v + \rho\boldsymbol{F}(\boldsymbol{\nabla} \times \boldsymbol{\Psi})) \rightarrow min$$

$$J_H = \int_\Omega \mu D_\xi \cdot D_\xi d\Omega \rightarrow min$$

Data set:
- Boundary conditions;

Advantages:
- non-Newtonian fluid;
- mass forces;
- less hyperparameters.

Limitations:
- stationary or quasi-stationary;
- incompressible

PINNs with energy (power) loss Step 1: prove that the loss is true

Variational problem

$$J[v_1(x_i), v_2(x_i)] = \int_{\Omega} f\left(x_i, v_1, v_2, r_i^k, q_{ij}^k\right) d\Omega \rightarrow \min(\max),$$

where $r_i^k = v_{k_{x_i}}'$, $q_{ij}^k = v_{k_{x_i x_j}}''$

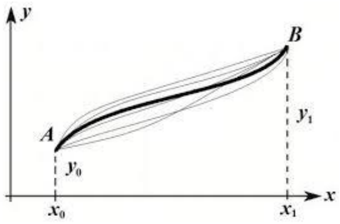
The generalized Euler's equations have the following form:

$$f_{v_k}' - \frac{\partial}{\partial x_i}\left(r_i^k\right) + \frac{\partial^2}{\partial x_i \partial x_j}\left(q_{ij}^k\right) = 0, \qquad i = 1{,}2;\ k = 1{,}2^*$$

Boundary conditions for these second order equations:

$$\begin{matrix} v_k\left(x_i^0\right) = v_i^0 & v_k'\left(x_i^0\right) = u_i^0 \\ v_k\left(x_i^1\right) = v_i^1 & v_k'\left(x_i^1\right) = u_i^1, \end{matrix} \quad i = 1{,}2; k = 1{,}2$$

1d interpretation



Non-trivial problem: to propose variational problem is equivalent to the partial differential equations (PDEs)

Task in another way: To find such functions $v_1(x_i), v_2(x_i)$ that provide an extremum to the functionality $J[v_1(x_i), v_2(x_i)]$ looks like a **ML task:**

Data → fitting algorithm Loss->min → $Y(\boldsymbol{\theta}, \boldsymbol{X})$

Fitting problem in ML

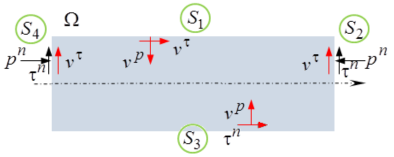
Just Boundary conditions → optimization algorithm of Functional->min(max) → $Y(\boldsymbol{\theta}, \boldsymbol{X})$

Variational problem

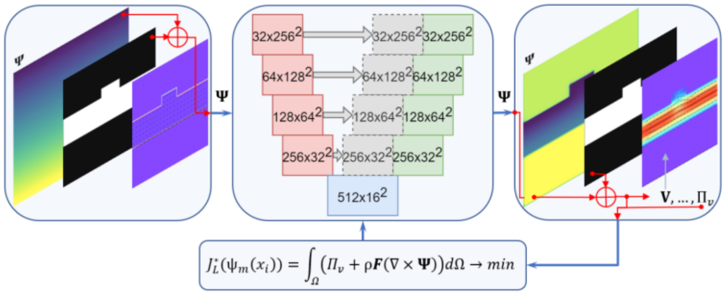
* The Einstein summation notation is used hereinafter

PINNs with energy (power) loss Step 2: minimize the loss

Newtonian 2d fluid flow. Convolutional network and image-based flow domain

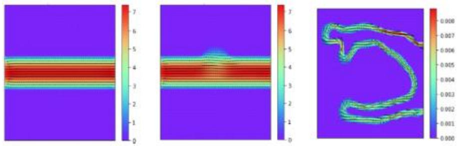


$$V = [\![v_1 \quad v_2 \quad 0]\!], \Psi = \psi(x_1, x_2),$$
$$v_1 = \frac{\partial \psi}{\partial x_2}, v_2 = -\frac{\partial \psi}{\partial x_1}, (\nabla \cdot V = 0)$$



The unknown Ψ function can be represented as a three-dimensional image:
- the position of a pixel in the image corresponds to coordinates;
- the pixel intensity corresponds to the value of the ψ function in the point.

Method	Maximum velocity, m/s		
	parallel plates	*parallel plates with notch*	*nailfold capillary*
Analytical solution	7.5	-	-
Ansys Fluent	7.42	7.83	-
UNet with loss (22)	7.35	7.52	$8.8 \cdot 10^{-3}$



(a) $L = 4 \cdot 10^{-4} m$ (b) $L = 4 \cdot 10^{-4} m$ (c) $L = 5 \cdot 10^{-4} m$

E. Kornaeva, A. Kornaev, A. Fetisov, I. Stebakov, and B. Ibragimov. Physics-based loss and machine learning approach in application to non-newtonian fluids flow modeling. 2022 IEEE Congress on Evolutionary Computation, CEC 2022 - Conference Proceedings, 2022. doi:10.1109/CEC55065.2022.9870411. A*.

Check the most cited papers on PINNs?

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations
https://www.sciencedirect.com/science/article/abs/pii/S0021999118307125

https://arxiv.org/pdf/1708.07469

https://arxiv.org/pdf/1710.00211

https://arxiv.org/pdf/1910.03193

https://arxiv.org/pdf/2006.10739

Paris Perdikaris
PhD · Professor (Associate) at University of Pennsylvania
United States

Research Interest Score	17,579
Citations	26,784
h-index	45

Citations over time

George Em Karniadakis
PhD, MIT, 1987 · Professor at Brown University
United States

Research Interest Score	54,570
Citations	91,419
h-index	135

Citations over time

Jian-Xun Wang
Ph.D. · Robert W. Huether Collegiate Associate Professor at University of Notre Dame
Notre Dame, United States

Research Interest Score	3,898
Citations	4,791
h-index	28

Citations over time

Why PINNs?

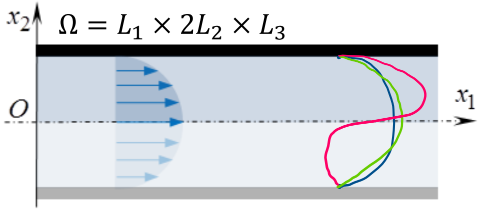
Thank you for your attention!

a.kornaev@innopolis.ru, @avkornaev

Notes

PINNs with energy (power) loss

Save mechanical power



Analytical solution:
$v_1 = c(L_2^2 - x_2^2).$

$$I[v_1(x_2)] = \int_{-L_2}^{L_2} f\left(x_2, v_1(x_2), \frac{dv_1(x_2)}{dx_2}\right) \to min, \qquad flow\ rate = const.$$

$$v_1(x_2) = \theta_0 + \theta_1 x_2 + \theta_2 x_2^2,$$
$$v_1(x_2) = \theta_0 + \theta_1 x_2 + \theta_2 x_2^2 + \theta_3 x_2^3 + \theta_4 x_2^4,$$
$$\dots$$
$$v_1(x_2) = \theta_0 + \theta_1 x_2 + \theta_2 x_2^2 + \dots + \theta_n x_2^n.$$

$$I[v_1(x_2)] = I[\theta_i] \to min.$$