Machine learning and cosmology

Josué Weiss*
Faculty of Physics and Engineering, University of Strasbourg

This seminar introduces Physics-Informed Neural Networks (PINNs) as a machine-learning framework for solving physical problems governed by differential equations. After presenting the basic principles of deep learning, neural networks and loss minimization, we explain how PINNs use automatic differentiation to impose physical laws during the training. The method is first illustrated on simple systems, such as Newton's law of cooling, to highlight the main differences between a basic neural network and Physics-Informed Neural Networks. Then, the harmonic oscillator is studied in more detail to better understand how PINNs work. We then introduce the cosmological context and apply the PINN approach to the Friedmann equations, showing how the scale factor and cosmological parameters can be learned within the same framework. This provides a simple example of how machine learning can complement standard numerical methods in cosmology.

CONTENTS

* josue.weiss314@gmail.com

INTRODUCTION

Many physical systems are described by differential equations. In classical mechanics, electromagnetism, fluid dynamics or cosmology, the evolution of a system is often obtained by solving such equations from given initial or boundary conditions. Traditional numerical methods, such as finite-difference schemes or Runge–Kutta methods, provide powerful and well-established tools for this purpose. However, recent developments in machine learning have opened new possibilities for combining numerical modelling, observational data and physical constraints within a single computational framework.

The aim of this seminar is to introduce one such approach: Physics-Informed Neural Networks PINNs. A standard neural network is a parametrized function trained to reproduce data by minimizing a loss function. A PINN extends this idea by adding the physical equation directly into the loss. Instead of learning only from data, the network is also penalized when its prediction does not satisfy the differential equation governing the system. This makes PINNs particularly interesting for direct problems, inverse problems and parameter estimation in physics.

The first part of the seminar introduces the basic concepts of deep learning. We explain what an artificial neuron is, how several neurons form a neural network, and how such a network learns by adjusting its weights and biases. We then compare a classical data-driven neural network with a physics-informed neural network. A central technical ingredient is automatic differentiation, which allows one to compute derivatives of the network output with respect to its input. These derivatives can then be inserted into a physical residual, for example in Newton's law of cooling or in the equation of the harmonic oscillator.

The second part applies these ideas to simple physical examples. The harmonic oscillator is used as a pedagogical test case because its equation,

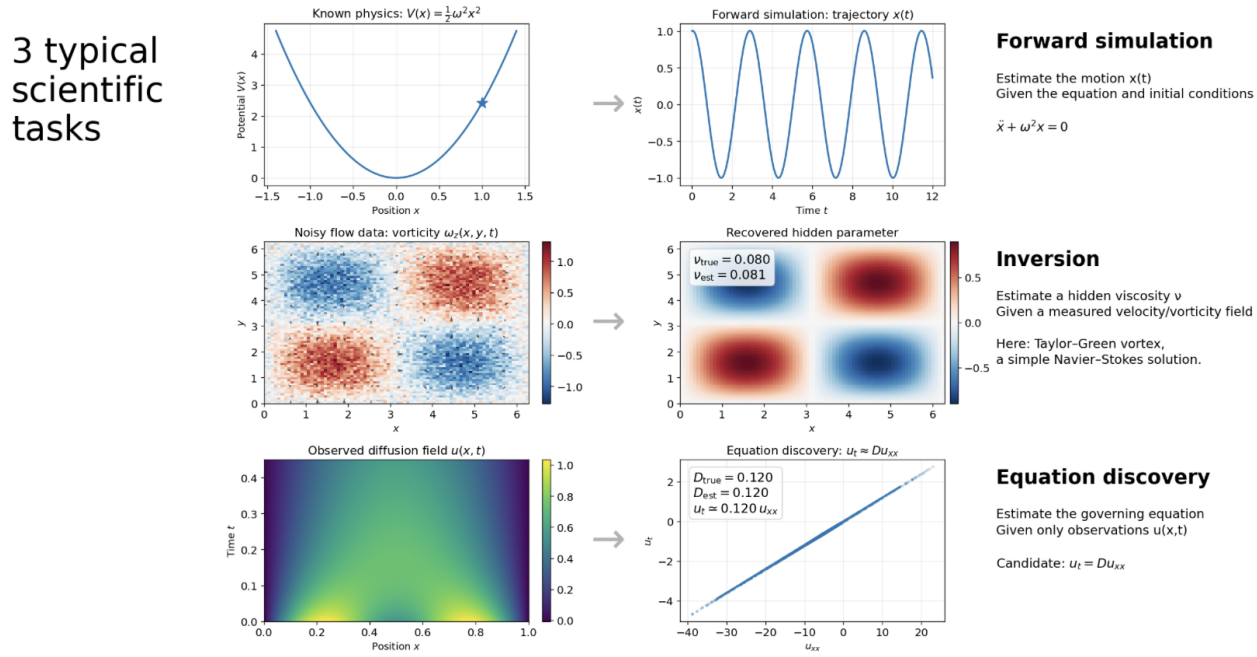
$$x''(t) + \omega^2 x(t) = 0, \tag{1}$$

is simple, well understood, and can be solved both analytically and numerically. Whereas a PINN learns a continuous function constrained by the physical equation over the whole time interval.

The final part of the seminar connects this framework to cosmology. After introducing the basic role of the metric in general relativity and the Friedmann–Lemaître–Robertson–Walker metric, we consider the Friedmann equations, which describe the evolution of the scale factor of the Universe. These equations are central in modern cosmology because they relate the expansion of the Universe to its matter, radiation, curvature and dark-energy content. In this context, PINNs can be used not only to approximate the evolution of the scale factor, but also to infer cosmological parameters from data. This provides a simple example of how machine learning can complement traditional numerical methods in cosmology.

I. FROM NEURAL NETWORKS TO PHYSICS-INFORMED NEURAL NETWORKS

The aim of this first part is to introduce the basic ideas behind neural networks and to explain how they can be adapted to physical problems. In computational physics, three typical tasks often appear. The first one is the direct problem: the physical law is known, and one computes the evolution of the system from given initial conditions. The second one is the inverse problem: the observations are known, and one tries to infer hidden parameters of the model. The third one is equation discovery: from measured or simulated data, one tries to identify the differential equation governing the system. These three tasks are central in many areas of physics, including cosmology.



A. Artificial neurons and neural networks

A neural network is a parametrized function. Its parameters are adjusted during training in order to reproduce data or satisfy constraints. The elementary building block is the artificial neuron. Given input variables $x_1, \ldots, x_n$, the neuron first computes a weighted sum, adds a bias, and then applies a nonlinear activation function:

$$z = \sum_{i=1}^{n} w_i x_i + b, \qquad a = \sigma(z). \tag{2}$$

The coefficients w_i are called weights, b is the bias, and σ is the activation function. In vector notation, this can be written more compactly as

$$a = \sigma(\mathbf{w} \cdot \mathbf{x} + b). \tag{3}$$

The activation function is a crucial ingredient because it introduces nonlinearity into the model. Without a nonlinear activation, a neural network made of several layers would simply be a composition of linear maps. But the composition of linear maps is still linear. Therefore, even a very deep network without nonlinear activation functions would not be more expressive than a single linear layer. Nonlinear activation functions allow the network to approximate much more complex functions.

Several activation functions are commonly used. A classical example is the sigmoid function,

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{4}$$

which maps any real number to a value between 0 and 1. Another important example is the hyperbolic tangent,

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}, \tag{5}$$

which maps real numbers to the interval $[-1, 1]$. The function tanh is smooth and differentiable, which makes it particularly convenient in Physics-Informed Neural Networks, where derivatives of the neural-network output must be computed.

A very common activation function in modern deep learning is the rectified linear unit, or ReLU:

$$\mathrm{ReLU}(z) = \max(0, z). \tag{6}$$

It is simple and efficient, and it often works very well for standard machine-learning tasks. However, for problems involving differential equations, especially when second derivatives are needed, smoother activation functions such as tanh are often preferred. This is because ReLU is not differentiable at $z = 0$ and its second derivative is zero almost everywhere, which can be problematic when the physics loss involves second-order derivatives.

A neural network is obtained by stacking many such neurons into successive layers. Each layer applies an affine transformation followed by a nonlinear activation:

$$\mathbf{h}^{(l+1)} = \sigma\left(W^{(l)}\mathbf{h}^{(l)} + \mathbf{b}^{(l)}\right), \tag{7}$$

where $W^{(l)}$ is a matrix of weights, $\mathbf{b}^{(l)}$ is a vector of biases, and $\mathbf{h}^{(l)}$ denotes the output of layer l. The final output of the network is therefore obtained by composing several nonlinear transformations.

If all the parameters of the network are denoted by θ, the network defines a function

$$f_\theta : x \mapsto f_\theta(x). \tag{8}$$

For example, if the input is a time t, the network can be used to approximate a time-dependent physical quantity,

$$t \mapsto x_\theta(t). \tag{9}$$

In this sense, the network is not only a numerical object: it represents a continuous function whose shape is controlled by trainable parameters. Training the network means finding the values of these parameters that make the function useful for the problem under study.

B. Training a classical neural network

In supervised learning, the network is trained from examples. One gives a set of data points

$$\{(x_i, y_i)\}_{i=1}^{N}, \tag{10}$$

where x_i denotes the input and y_i the expected output. The neural network produces predictions $f_\theta(x_i)$, which are then compared to the true values y_i. The goal of training is to adjust the parameters θ so that the predictions become as close as possible to the data.

This comparison is made through a loss function, also called a cost function. The loss function associates a number to the current state of the network: this number measures how wrong the network is. A standard example is the mean squared error:

$$\mathcal{L}_{\text{data}}(\theta) = \frac{1}{N}\sum_{i=1}^{N}\left(f_\theta(x_i) - y_i\right)^2. \tag{11}$$

If the predictions are close to the targets, the loss is small. If the predictions are far from the targets, the loss is large. Training the neural network therefore means solving the minimization problem

$$\theta^\star = \arg\min_\theta \mathcal{L}_{\text{data}}(\theta). \tag{12}$$

The loss function can be understood as a landscape in the space of parameters. Each possible choice of weights and biases corresponds to a point in this landscape, and the value of the loss tells us the altitude of this point. Training consists in moving through this landscape in order to reach a region where the loss is low. This is done using the gradient of the loss with respect to the parameters:

$$\nabla_\theta \mathcal{L}_{\text{data}}. \tag{13}$$

The gradient indicates the direction in which the loss increases most rapidly. Therefore, to reduce the loss, the parameters are updated in the opposite direction:

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta \mathcal{L}_{\text{data}}(\theta_k), \tag{14}$$

where η is the learning rate. The learning rate controls the size of the update. If it is too small, training can be very slow. If it is too large, the optimization may become unstable and fail to converge.

In practice, the gradients are computed efficiently using backpropagation. Backpropagation is an application of the chain rule of differentiation to the computational graph of the neural network. It allows one to compute how each weight and bias contributes to the final loss, and therefore how each parameter should be modified.

There are several ways to use the data during training. In *batch gradient descent*, the loss is computed using the whole dataset at each update:

$$\mathcal{L}_{\text{batch}} = \frac{1}{N}\sum_{i=1}^{N}\left(f_\theta(x_i) - y_i\right)^2. \tag{15}$$

This gives an accurate estimate of the gradient, but it can be computationally expensive when the dataset is large.

In *stochastic gradient descent*, one uses only one data point at each update:

$$\mathcal{L}_i = \left(f_\theta(x_i) - y_i\right)^2. \tag{16}$$

This makes each update cheaper, but the direction of the update is noisy because it depends on a single example. This noise can sometimes be useful because it helps the optimization avoid getting trapped too early in poor local minima.

A compromise between these two approaches ismini-batch training. Instead of using the whole dataset or a single point, one uses a small subset $\mathcal{B}$ of the data:

$$\mathcal{L}_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}\left(f_\theta(x_i) - y_i\right)^2. \tag{17}$$

This is the most common approach in deep learning. It is computationally efficient, works well with modern hardware, and provides a good balance between stable convergence and stochastic exploration of the loss landscape.

The optimization is repeated many times. One complete pass through the training dataset is called an epoch. During training, the network progressively modifies its weights and biases so that its predictions better match the desired outputs. In classical supervised learning, the information available to the model comes mainly from the data. However, in physics, we often know more than the data: we also know equations, conservation laws, symmetries or initial conditions. Physics-Informed Neural Networks are designed to include this physical information directly in the training process.

C. Physics-informed neural networks

A physics-informed neural network, or PINN, is a neural network trained not only to fit data, but also to satisfy a physical law. In contrast with a standard data-driven neural network, which learns only from observations, a PINN incorporates prior physical knowledge directly into the training process. This idea has become a useful framework for solving direct and inverse problems involving differential equations. It is also connected to earlier work on the use of neural networks for solving ordinary and partial differential equations.

The main idea is the following. Suppose that a physical quantity $u(t)$ satisfies a differential equation of the form

$$\mathcal{N}[u](t) = 0, \tag{18}$$

where $\mathcal{N}$ is a differential operator. Instead of solving directly for $u(t)$ by a standard numerical method, one introduces a neural-network approximation

$$u(t) \simeq u_\theta(t), \tag{19}$$

where θ denotes the trainable parameters of the network. The physical law is then enforced by defining the residual

$$R_\theta(t) = \mathcal{N}[u_\theta](t). \tag{20}$$

If the neural network were an exact solution of the physical problem, this residual would vanish:

$$R_\theta(t) = 0. \tag{21}$$

In practice, the equation is not imposed everywhere, but at a finite set of points called collocation points. These points are not experimental data. They are points in the domain where the network is required to satisfy the differential equation. This leads to a physics-based loss term of the form

$$\mathcal{L}_{\text{phys}} = \frac{1}{N_c}\sum_{j=1}^{N_c}\left|R_\theta(t_j)\right|^2, \tag{22}$$

where N_c is the number of collocation points.

If observed data are also available, one introduces in addition a data loss,

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left(u_\theta(t_i) - u_i^{\text{data}} \right)^2 , \tag{23}$$

where N_d is the number of training points. If initial or boundary conditions must also be enforced, one may further define a loss term $\mathcal{L}_{\text{IC}}$ or $\mathcal{L}_{\text{BC}}$. The total loss is then typically written as

$$\mathcal{L}_{\text{tot}} = \lambda_{\text{data}} \mathcal{L}_{\text{data}} + \lambda_{\text{phys}} \mathcal{L}_{\text{phys}} + \lambda_{\text{IC}} \mathcal{L}_{\text{IC}}, \tag{24}$$

where the coefficients λ_{data}, λ_{phys} and λ_{IC} control the relative importance of each contribution.

A simple and instructive example is given by Newton's law of cooling. If $T(t)$ denotes the temperature of an object placed in an environment at constant temperature T_{env}, then the evolution of the system is governed by

$$\frac{dT}{dt} + \lambda \left(T - T_{\text{env}} \right) = 0, \tag{25}$$

where $\lambda > 0$ is a cooling coefficient. In a standard neural-network approach, one would simply train a network $T_\theta(t)$ to fit the available temperature measurements by minimizing

$$\mathcal{L}_{\text{data}} = \frac{1}{N} \sum_{i=1}^{N} \left(T_\theta(t_i) - T_{\text{true}}(t_i) \right)^2 . \tag{26}$$

Such a model may interpolate the data correctly, but it is not guaranteed to follow the physical law outside the training region. As a consequence, its extrapolation may become unphysical.

For a PINN, one keeps the same neural-network approximation $T_\theta(t)$, but one also imposes Newton's equation through the residual

$$R_\theta(t) = \frac{dT_\theta}{dt} + \lambda \left(T_\theta(t) - T_{\text{env}} \right) . \tag{27}$$

The corresponding physics loss is

$$\mathcal{L}_{\text{phys}} = \frac{1}{M} \sum_{j=1}^{M} \left[\frac{dT_\theta}{dt}\Big|_{t_j} + \lambda \left(T_\theta(t_j) - T_{\text{env}} \right) \right]^2 . \tag{28}$$

The total loss therefore combines the agreement with the measured data and the agreement with the governing equation:

$$\mathcal{L}_{\text{tot}} = \frac{1}{N} \sum_{i=1}^{N} \left(T_\theta(t_i) - T_{\text{true}}(t_i) \right)^2 + \frac{1}{M} \sum_{j=1}^{M} \left[\frac{dT_\theta}{dt}\Big|_{t_j} + \lambda \left(T_\theta(t_j) - T_{\text{env}} \right) \right]^2 . \tag{29}$$

This example clearly illustrates the conceptual difference between a standard neural network and a PINN. The standard neural network is only asked to reproduce the training data, whereas the PINN is asked both to fit the data and to remain compatible with the differential equation. As illustrated in Fig. 1, the PINN therefore tends to generalize better, especially when only a limited number of data points is available. The collocation points used in the physics loss are not additional measurements; rather, they are locations where the physical consistency of the solution is enforced.

This is one of the main strengths of PINNs in physics: they combine observational information and theoretical knowledge in a single framework. This makes them especially attractive for scientific problems where data may be sparse or noisy, but where the governing equations are at least partially known.

D. Automatic differentiation

A central technical point is the computation of derivatives. In a PINN, one needs derivatives of the network output with respect to its input. For instance, if the network predicts $u_\theta(t)$, one may need

$$\frac{\mathrm{d}u_\theta}{\mathrm{d}t}, \qquad \frac{\mathrm{d}^2 u_\theta}{\mathrm{d}t^2}. \tag{30}$$

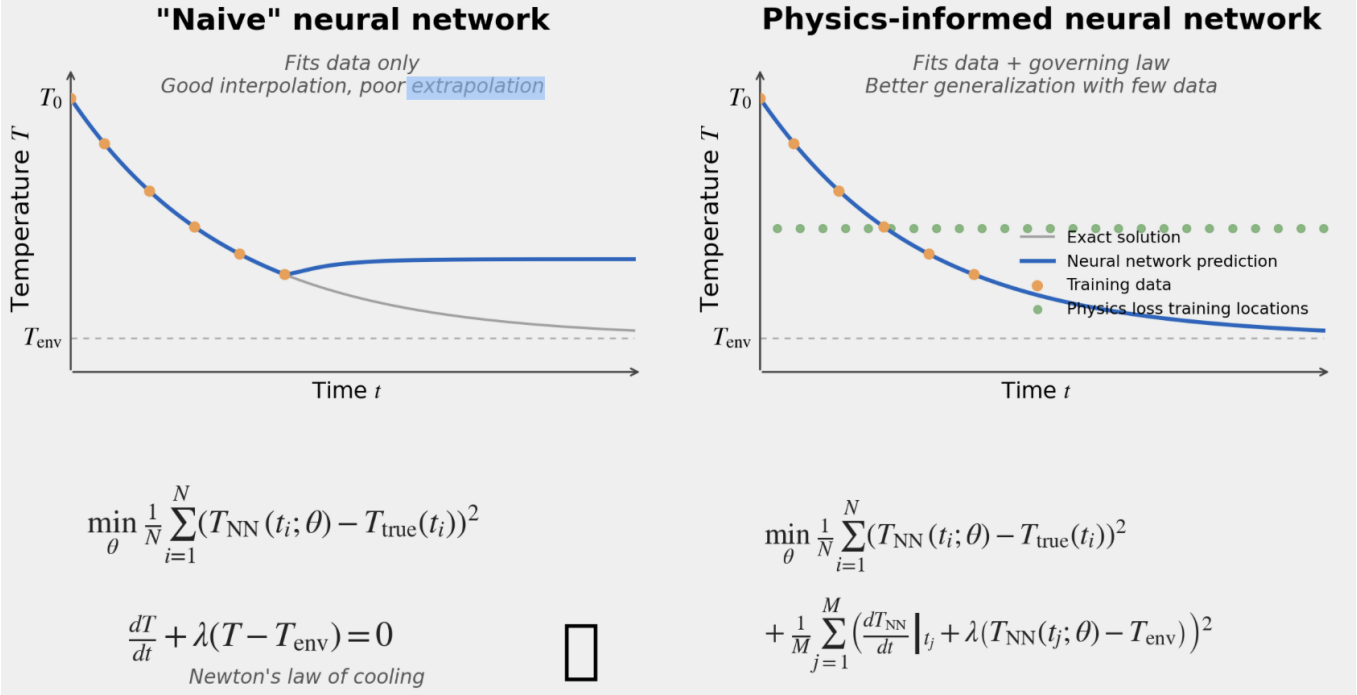


FIG. 1: Comparison between a standard data-driven neural network and a physics-informed neural network for Newton's law of cooling. A standard neural network is trained only on data and may produce poor extrapolation outside the training region. By contrast, a PINN combines a data loss with a physics loss enforcing the governing differential equation at collocation points, which generally improves physical consistency and generalization.

These derivatives are not computed by finite differences. In frameworks such as PyTorch, they are computed by automatic differentiation. The operations performed by the network are recorded in a computational graph, and derivatives are obtained by repeated application of the chain rule.

This is particularly useful in physics because the derivatives can be inserted directly into a differential equation. For example, for Newton's law of cooling,

$$\frac{\mathrm{d}T}{\mathrm{d}t} + \lambda\,(T - T_{\mathrm{env}}) = 0, \tag{31}$$

a neural network can predict $T_\theta(t)$. Automatic differentiation gives $\frac{\mathrm{d}T_\theta}{\mathrm{d}t}$, and the physical residual becomes

$$R_\theta(t) = \frac{\mathrm{d}T_\theta}{\mathrm{d}t} + \lambda\,(T_\theta(t) - T_{\mathrm{env}})\,. \tag{32}$$

Training the PINN then means forcing this residual to be as close as possible to zero.

E. Example: the harmonic oscillator

As a simple example, consider the harmonic oscillator:

$$x''(t) + \omega^2 x(t) = 0, \tag{33}$$

with initial conditions

$$x(0) = x_0, \qquad \dot{x}(0) = v_0. \tag{34}$$

This equation is a useful benchmark because its behaviour is well understood and its exact solution is known. A classical numerical approach consists in rewriting the second-order equation as a first-order system:

$$\dot{x} = v, \tag{35a}$$

$$\dot{v} = -\omega^2 x. \tag{35b}$$

This system can then be solved with a standard method such as the fourth-order Runge–Kutta method. This gives a numerical trajectory that can be compared with the analytical solution.

In the PINN approach, the neural network directly represents the function $x_\theta(t)$. The first and second derivatives are computed by automatic differentiation:

$$\dot{x}_\theta(t) = \frac{\mathrm{d}x_\theta}{\mathrm{d}t}, \qquad \ddot{x}_\theta(t) = \frac{\mathrm{d}^2 x_\theta}{\mathrm{d}t^2}. \tag{36}$$

The physical residual is then

$$R_\theta(t) = \ddot{x}_\theta(t) + \omega^2 x_\theta(t). \tag{37}$$

The corresponding physical loss is

$$\mathcal{L}_{\mathrm{phys}} = \frac{1}{N_c} \sum_{j=1}^{N_c} \left(\ddot{x}_\theta(t_j) + \omega^2 x_\theta(t_j) \right)^2. \tag{38}$$

The initial conditions are imposed with an additional loss:

$$\mathcal{L}_{\mathrm{IC}} = \left(x_\theta(0) - x_0 \right)^2 + \left(\dot{x}_\theta(0) - v_0 \right)^2. \tag{39}$$

The total loss is therefore

$$\mathcal{L}_{\mathrm{tot}} = \mathcal{L}_{\mathrm{phys}} + \lambda_{\mathrm{IC}} \mathcal{L}_{\mathrm{IC}}. \tag{40}$$

This example illustrates the main difference between a classical numerical method and a PINN. A Runge–Kutta method propagates the solution step by step in time. A PINN instead learns a continuous function $x_\theta(t)$ over the whole time interval, while being penalized when this function does not satisfy the differential equation. The collocation points are therefore not data points: they are points where the physical law is enforced.

1. Practical behaviour and effect of the training duration

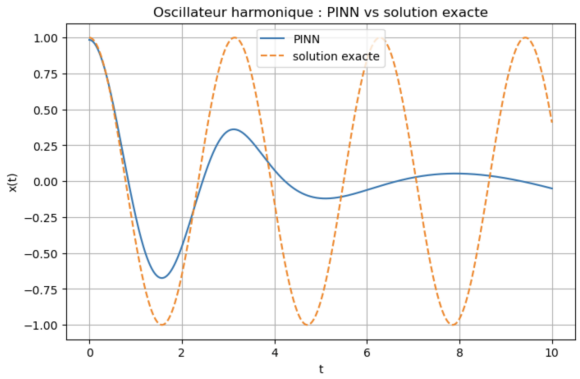
In practice, the quality of the PINN approximation strongly depends on the training duration. For the harmonic oscillator, this point is particularly important because the solution is oscillatory, which makes the learning task more difficult than for a monotonic function. If the training is stopped too early, the network may satisfy the initial condition reasonably well and capture the first part of the motion, while still failing to reproduce the periodic structure over the whole time interval.

This behaviour is illustrated in Fig. 2. After 10 000 epochs, the PINN prediction already follows the general trend at short times, but it does not reproduce the exact oscillatory behaviour correctly over the full interval. In particular, the predicted amplitude decreases too rapidly, leading to an artificial damping that is absent from the true solution. This shows that the network has not yet fully learned the physical constraint imposed by the differential equation.

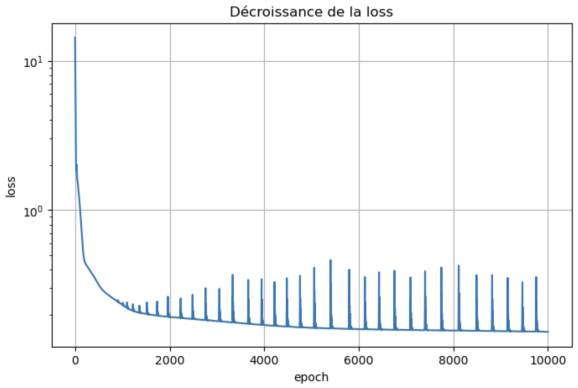
By contrast, after 60 000 epochs, the agreement between the PINN prediction and the exact solution becomes very good. The network is then able to reproduce both the amplitude and the phase of the oscillation over the whole domain. This comparison shows that, even for a simple benchmark such as the harmonic oscillator, PINN training may require a significant number of epochs before converging to a satisfactory solution.

The evolution of the loss during training is also informative. As shown in Fig. 2, the total loss decreases overall in both cases, which indicates that the optimization makes progress. However, the decrease is not perfectly monotonic: one observes repeated spikes in the loss curve. Such fluctuations are common in neural-network optimization and do not necessarily indicate a failure of the method. They may arise from the optimizer dynamics, the balance between the different contributions to the loss, or the difficulty of fitting an oscillatory solution. What matters is the global trend: after 10 000 epochs the loss remains too large for the solution to be accurate, whereas after 60 000 epochs it reaches a much lower level, consistent with the excellent agreement between the PINN and the analytical solution.

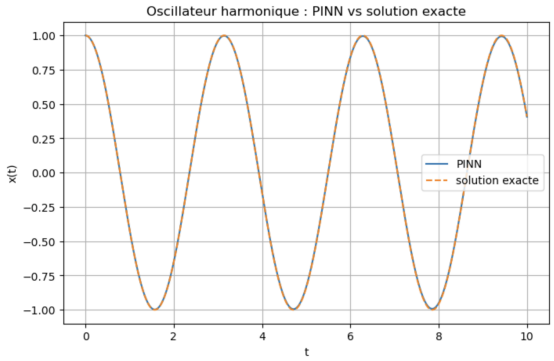
This example highlights an important practical point: for PINNs, convergence must be assessed not only from the loss value, but also from the reconstructed physical solution itself. A decreasing loss is necessary, but the final validation must always be made by comparing the learned solution with the expected physical behaviour.



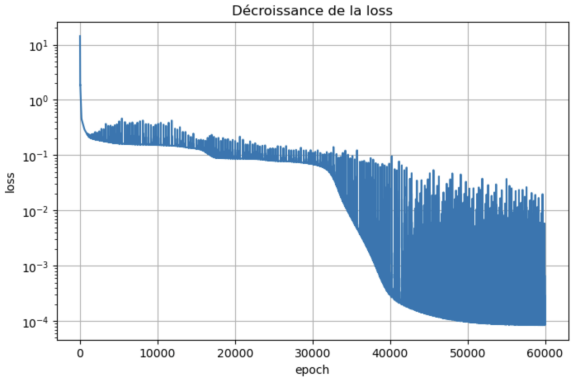
(a) PINN prediction after 10 000 epochs.



(b) Loss history after 10 000 epochs.



(c) PINN prediction after 60 000 epochs.



(d) Loss history after 60 000 epochs.

FIG. 2: Effect of the training duration on the PINN solution for the harmonic oscillator. After 10 000 epochs, the network captures the beginning of the dynamics but still fails to reproduce the full oscillatory behaviour, showing an artificial damping. The associated loss decreases but remains relatively large. After 60 000 epochs, the agreement with the exact solution becomes excellent, while the loss reaches a much lower level. The spikes visible in the loss curves reflect the non-monotonic nature of neural-network optimization and do not prevent convergence.

F. Interest and limitations of PINNs

PINNs are attractive because they provide a flexible way to combine data and physical equations in the same framework. They can be used for direct simulations, inverse problems, and sometimes for discovering or constraining unknown parameters in equations. They are also useful pedagogically, because they make the role of the differential equation explicit in the loss function.

However, PINNs are not a universal replacement for classical numerical methods. Their training can be unstable, sensitive to hyperparameters, and computationally expensive. The balance between data loss, physical loss and initial or boundary conditions may also be difficult to tune. For oscillatory, multi-scale or wave-like problems, the training can become especially challenging. Therefore, PINNs should be understood as a complementary tool: they are powerful when one wants to combine physical modelling, sparse data and parameter inference, but classical solvers remain essential for many precise numerical simulations.

II. FROM GENERAL RELATIVITY TO COSMOLOGY

A. Gravity in Einstein's picture

In Newtonian physics, gravity is described as a force acting at a distance between massive bodies. In general relativity, Einstein proposed a different interpretation: gravity is not a force in the usual sense, but a manifestation of the curvature of spacetime. Massive bodies and energy distributions deform spacetime, and free particles move according to this geometry.

This idea can be summarized by the famous statement: matter tells spacetime how to curve, and spacetime tells matter how to move. More precisely, the motion of a freely falling particle is described by a geodesic, that is, by the natural generalization of a straight line to curved spacetime. If $x^\mu(\tau)$ denotes the particle trajectory, its motion satisfies the geodesic equation

$$\frac{d^2x^\mu}{d\tau^2} + \Gamma^\mu_{\alpha\beta}\frac{dx^\alpha}{d\tau}\frac{dx^\beta}{d\tau} = 0, \tag{41}$$

where τ is an affine parameter along the trajectory and $\Gamma^\mu_{\alpha\beta}$ are the Christoffel symbols. These coefficients encode how the geometry modifies the notion of straight motion.

The fundamental object of the theory is the metric tensor $g_{\mu\nu}$, which defines spacetime intervals through

$$ds^2 = g_{\mu\nu}dx^\mu dx^\nu. \tag{42}$$

The metric determines distances, durations, angles and causal structure. Once the metric is known, the geometric structure of spacetime is fixed.

B. From the metric to curvature

The geometric content of general relativity is built step by step from the metric. Starting from $g_{\mu\nu}$, one defines the Christoffel symbols, then the Riemann curvature tensor, then the Ricci tensor, and finally the Ricci scalar. The logical chain is illustrated in Fig. 3.

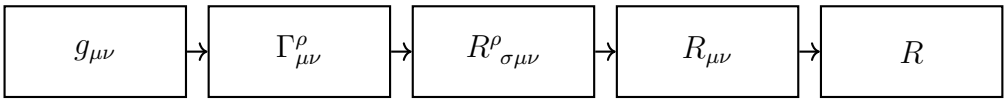


FIG. 3: Geometric hierarchy in general relativity: from the metric to the spacetime curvature scalars.

The Christoffel symbols are constructed from the metric according to

$$\Gamma^\rho_{\mu\nu} = \frac{1}{2}g^{\rho\sigma}\left(\partial_\mu g_{\nu\sigma} + \partial_\nu g_{\mu\sigma} - \partial_\sigma g_{\mu\nu}\right). \tag{43}$$

They are not tensors themselves, but they are essential because they define covariant derivatives and geodesic motion.

The Riemann tensor measures the intrinsic curvature of spacetime:

$$R^\rho{}_{\sigma\mu\nu} = \partial_\mu\Gamma^\rho_{\nu\sigma} - \partial_\nu\Gamma^\rho_{\mu\sigma} + \Gamma^\rho_{\mu\lambda}\Gamma^\lambda_{\nu\sigma} - \Gamma^\rho_{\nu\lambda}\Gamma^\lambda_{\mu\sigma}. \tag{44}$$

From it, one obtains the Ricci tensor by contraction,

$$R_{\mu\nu} = R^\rho{}_{\mu\rho\nu}, \tag{45}$$

and the Ricci scalar

$$R = g^{\mu\nu}R_{\mu\nu}. \tag{46}$$

C. Einstein field equations

The dynamics of spacetime is governed by Einstein's field equations. Throughout this seminar, we use units in which the speed of light is set to unity, $c = 1$:

$$G_{\mu\nu} + \Lambda g_{\mu\nu} = 8\pi G\, T_{\mu\nu}, \tag{47}$$

where

$$G_{\mu\nu} = R_{\mu\nu} - \frac{1}{2} R g_{\mu\nu} \tag{48}$$

is the Einstein tensor, Λ is the cosmological constant, G is Newton's gravitational constant, and $T_{\mu\nu}$ is the energy-momentum tensor.

These equations relate geometry to matter and energy. The left-hand side describes the curvature of spacetime, while the right-hand side describes its physical content. In cosmology, this equation is applied to a highly symmetric metric describing the large-scale Universe.

D. The FLRW metric

The Friedmann–Lemaître–Robertson–Walker (FLRW) metric describes the large-scale geometry of the Universe. It is based on the cosmological principle, which states that the Universe is homogeneous and isotropic on sufficiently large scales.

- **Homogeneous:** the Universe looks the same from all points in space.
- **Isotropic:** the Universe looks the same in all spatial directions.

These assumptions imply that spatial slices at fixed cosmic time are maximally symmetric 3-spaces of constant curvature. The spatial geometry may therefore be closed, flat or open, and its overall scale evolves in time through the scale factor $a(t)$.

In spherical coordinates (t, r, θ, ϕ), the FLRW line element can be written as

$$ds^2 = dt^2 - a^2(t) \left[\frac{dr^2}{1 - kr^2} + r^2 d\theta^2 + r^2 \sin^2\theta\, d\phi^2 \right], \tag{49}$$

where

$$k = \begin{cases} +1 & \text{closed Universe (positive spatial curvature),} \\ 0 & \text{flat Universe,} \\ -1 & \text{open Universe (negative spatial curvature).} \end{cases} \tag{50}$$

Equivalently, in matrix form and with the signature $(+,-,-,-)$, one may write

$$g_{\mu\nu} = \begin{pmatrix} +1 & 0 & 0 & 0 \\ 0 & -\dfrac{a^2(t)}{1-kr^2} & 0 & 0 \\ 0 & 0 & -\big(a(t)r\big)^2 & 0 \\ 0 & 0 & 0 & -\big(a(t)r\sin\theta\big)^2 \end{pmatrix}. \tag{51}$$

The scale factor $a(t)$ contains the information about the expansion of the Universe. Even if the spatial sections are flat, *i.e.* when $k = 0$, a time-dependent scale factor generally implies a non-zero spacetime curvature. In other words, flat spatial slices do not mean flat spacetime.

E. Friedmann equations

When Einstein's equations are applied to the FLRW metric, assuming that the matter content of the Universe behaves as a perfect fluid, one obtains the Friedmann equations. The first Friedmann equation is

$$H^2 \equiv \left(\frac{\dot{a}}{a}\right)^2 = \frac{8\pi G}{3}\rho - \frac{k}{a^2} + \frac{\Lambda}{3}, \tag{52}$$

where $H(t) = \dot{a}/a$ is the Hubble parameter and ρ is the total energy density.

The second Friedmann equation is

$$\frac{\ddot{a}}{a} = -\frac{4\pi G}{3}(\rho + 3p) + \frac{\Lambda}{3}, \tag{53}$$

where p is the pressure of the cosmic fluid.

These equations describe how the expansion rate of the Universe depends on its matter and energy content. They form the basis of modern cosmology.

F. Cosmological parameters

A convenient way to write the Friedmann equations is to introduce the present-day critical density

$$\rho_c = \frac{3H_0^2}{8\pi G}, \tag{54}$$

and the density parameters

$$\Omega_i = \frac{\rho_i}{\rho_c}. \tag{55}$$

One also defines the curvature parameter

$$\Omega_k = -\frac{k}{a_0^2 H_0^2}, \tag{56}$$

where H_0 is the present value of the Hubble parameter and a_0 is the present value of the scale factor, often normalized to $a_0 = 1$.

In terms of these dimensionless quantities, the first Friedmann equation becomes

$$\left(\frac{H}{H_0}\right)^2 = \frac{\Omega_r}{a^4} + \frac{\Omega_m}{a^3} + \frac{\Omega_k}{a^2} + \Omega_\Lambda, \tag{57}$$

where Ω_r, Ω_m and Ω_Λ respectively denote the radiation, matter and dark-energy contributions.

In a spatially flat Universe, $\Omega_k = 0$, and one has

$$\Omega_r + \Omega_m + \Omega_\Lambda = 1. \tag{58}$$

G. A simple reference model

In the simplified model considered later in this seminar, we neglect radiation and assume a flat Universe. The first Friedmann equation then reduces to

$$\left(\frac{H}{H_0}\right)^2 = \frac{\Omega_m}{a^3} + \Omega_\Lambda, \tag{59}$$

with

$$\Omega_m + \Omega_\Lambda = 1. \tag{60}$$

This form is particularly convenient for a first PINN implementation, since it already captures the competition between matter and dark energy while remaining analytically tractable. In this case, the scale factor admits the exact solution

$$a(t) = \left(\frac{\Omega_m}{\Omega_\Lambda}\right)^{1/3} \sinh^{2/3}\left(\frac{3}{2} H_0 \sqrt{\Omega_\Lambda}\, t\right), \tag{61}$$

which provides a useful benchmark for generating synthetic data and comparing the PINN prediction with a known solution.

III. PHYSICS-INFORMED NEURAL NETWORK FOR THE FRIEDMANN EQUATIONS

A. Goal of the cosmological PINN

We now apply the PINN framework to a simple cosmological model. The goal is to reconstruct the evolution of the scale factor $a(t)$ and, in the inverse setting, to infer the cosmological parameters entering the Friedmann equation.

For simplicity, we consider a spatially flat Universe containing only matter and a cosmological-constant contribution. Radiation and curvature are neglected. The first Friedmann equation then reads

$$\left(\frac{H}{H_0}\right)^2 = \frac{\Omega_m}{a^3} + \Omega_\Lambda, \tag{62}$$

where

$$H = \frac{\dot{a}}{a}. \tag{63}$$

Introducing the dimensionless time variable

$$\tau = H_0 t, \tag{64}$$

the equation becomes

$$\left(\frac{1}{a}\frac{da}{d\tau}\right)^2 = \frac{\Omega_m}{a^3} + \Omega_\Lambda. \tag{65}$$

This equation will be used as the physical constraint inside the PINN. The network must therefore learn a function $a_\theta(\tau)$ which both fits the available data and satisfies the Friedmann equation.

B. Synthetic data and noisy observations

In order to test the method in a controlled setting, synthetic data are generated from a known reference solution. We choose fixed true values of the cosmological parameters, for example

$$\Omega_m^{\text{true}} = 0.30, \qquad \Omega_\Lambda^{\text{true}} = 0.70. \tag{66}$$

These values are used only to generate the artificial data. The PINN does not directly know them during the inverse problem.

For a flat matter–Λ universe, the scale factor admits the analytical form

$$a(\tau) = \left(\frac{\Omega_m}{\Omega_\Lambda}\right)^{1/3} \sinh^{2/3}\left(\frac{3}{2} \sqrt{\Omega_\Lambda}\, \tau\right). \tag{67}$$

This reference solution gives a clean dataset

$$a_{\text{clean}}(\tau_i). \tag{68}$$

To imitate observational uncertainties, a random noise term is added:

$$a_{\text{noisy}}(\tau_i) = a_{\text{clean}}(\tau_i) + \epsilon_i, \tag{69}$$

where ϵ_i is a small random perturbation. These noisy values are the only data points seen by the neural network.

This distinction is important. The clean data represent the ideal theoretical solution, whereas the noisy data represent an imperfect observational dataset. The PINN must learn from these noisy observations while also being constrained by the Friedmann equation.

C. Collocation points

In addition to the data points, the method uses collocation points. These points are not observational data. They are points in the time domain where the physical equation is imposed.

We denote them by

$$\{\tau_j^c\}_{j=1}^{N_c}. \tag{70}$$

At each collocation point, the network prediction $a_\theta(\tau_j^c)$ is inserted into the Friedmann equation. The residual measures how much the neural-network solution violates the physical law.

Thus, the training set contains two different types of points:

- **data points**, where the prediction is compared with noisy observations;
- **collocation points**, where the prediction is tested against the Friedmann equation.

The first type enforces agreement with observations. The second type enforces physical consistency.

D. Neural-network representation of the scale factor

The neural network takes the dimensionless time τ as input and outputs an approximation of the scale factor:

$$\tau \longmapsto a_\theta(\tau). \tag{71}$$

Equivalently, the network represents a continuous function:

$$a(\tau) \simeq a_\theta(\tau). \tag{72}$$

Since the cosmological scale factor must be positive, the raw output of the neural network can be transformed with a positive activation function. A convenient choice is the softplus function:

$$\text{softplus}(x) = \ln(1 + e^x). \tag{73}$$

If the raw network output is denoted by $f_\theta(\tau)$, one may define

$$a_\theta(\tau) = \text{softplus}\left(f_\theta(\tau)\right) + \varepsilon, \tag{74}$$

where ε is a small positive constant added for numerical stability. This guarantees

$$a_\theta(\tau) > 0. \tag{75}$$

This is useful because the Friedmann equation contains powers such as a^{-3} and the ratio a'/a, which would become ill-defined if the network predicted zero or negative values.

E. Trainable cosmological parameters

In a standard direct problem, the cosmological parameters are fixed and the network learns only the scale factor. In the inverse problem considered here, the parameters Ω_m and Ω_Λ are also learned during training.

To do this, one introduces trainable raw parameters p_m and p_Λ. These are optimized in the same way as the weights and biases of the neural network. However, physical constraints must be imposed:

$$\Omega_m \geq 0, \qquad \Omega_\Lambda \geq 0, \qquad \Omega_m + \Omega_\Lambda = 1. \tag{76}$$

The last condition corresponds to a flat Universe with no curvature contribution.

A simple way to enforce these constraints is to use a softmax parametrization:

$$\Omega_m = \frac{e^{p_m}}{e^{p_m} + e^{p_\Lambda}}, \qquad \Omega_\Lambda = \frac{e^{p_\Lambda}}{e^{p_m} + e^{p_\Lambda}}. \tag{77}$$

This automatically guarantees positivity and normalization:

$$\Omega_m + \Omega_\Lambda = 1. \tag{78}$$

Therefore, the model learns both the function $a_\theta(\tau)$ and the cosmological parameters Ω_m, Ω_Λ.

F. Data loss

The data loss measures the discrepancy between the neural-network prediction and the noisy observations:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_d} \sum_{i=1}^{N_d} \left[a_\theta(\tau_i) - a_{\text{noisy}}(\tau_i) \right]^2 . \tag{79}$$

This term forces the network to remain close to the available observational data.

However, if this were the only loss term, the model would behave like a standard supervised neural network. It could fit the data without necessarily satisfying the Friedmann equation. The physics loss is therefore essential.

G. Initial-condition loss

The scale factor is also constrained by an initial condition. If the initial value is denoted by

$$a(0) = a_{\text{init}}, \tag{80}$$

one defines the initial-condition loss

$$\mathcal{L}_{\text{IC}} = \left[a_\theta(0) - a_{\text{init}} \right]^2 . \tag{81}$$

This term anchors the solution at the beginning of the time interval.

In practice, initial conditions are important because a differential equation does not determine a unique solution without them. They prevent the PINN from learning a function that satisfies the equation but corresponds to the wrong physical trajectory.

H. Physics loss and Friedmann residual

The physics loss is built from the Friedmann equation. Using the neural-network prediction $a_\theta(\tau)$, automatic differentiation gives

$$\frac{da_\theta}{d\tau}. \tag{82}$$

The dimensionless Hubble parameter predicted by the network is therefore

$$E_\theta(\tau) = \frac{1}{a_\theta(\tau)} \frac{da_\theta}{d\tau}, \tag{83}$$

where $E(\tau) = H/H_0$.

The Friedmann equation predicts

$$E^2(\tau) = \frac{\Omega_m}{a^3(\tau)} + \Omega_\Lambda. \tag{84}$$

The residual is therefore defined as

$$R_\theta(\tau) = \left[\frac{1}{a_\theta(\tau)} \frac{da_\theta}{d\tau} \right]^2 - \left[\frac{\Omega_m}{a_\theta^3(\tau)} + \Omega_\Lambda \right]. \tag{85}$$

If the network prediction exactly satisfies the Friedmann equation, then

$$R_\theta(\tau) = 0. \tag{86}$$

The physics loss is then

$$\mathcal{L}_{\text{phys}} = \frac{1}{N_c} \sum_{j=1}^{N_c} \left[R_\theta(\tau_j^c) \right]^2 . \tag{87}$$

This term is the key difference between a standard neural network and a PINN. It forces the learned function to be compatible with the cosmological dynamics.

I. Total loss and training

The full training objective combines the three contributions:

$$\mathcal{L}_{\text{tot}} = \lambda_{\text{data}}\mathcal{L}_{\text{data}} + \lambda_{\text{IC}}\mathcal{L}_{\text{IC}} + \lambda_{\text{phys}}\mathcal{L}_{\text{phys}}. \tag{88}$$

The coefficients λ_{data}, λ_{IC} and λ_{phys} control the relative importance of the data, the initial condition and the physical equation.

During training, the optimizer updates two types of quantities:

- the neural-network parameters θ, which determine the function $a_\theta(\tau)$;
- the raw cosmological parameters p_m and p_Λ, from which Ω_m and Ω_Λ are obtained through the softmax.

The optimization therefore solves two problems at the same time. First, it reconstructs the time evolution of the scale factor. Second, it infers the cosmological parameters that make this evolution compatible with the Friedmann equation.

J. Numerical results

The numerical results of the inverse Friedmann problem are shown in Fig. 4. The goal is to reconstruct the scale factor $a(\tau)$ from noisy synthetic data while simultaneously learning the cosmological parameters Ω_m and Ω_Λ.

The reconstructed scale factor is in very good agreement with the reference solution. Although the training data are noisy, the PINN prediction follows the smooth physical behaviour imposed by the Friedmann equation. This illustrates one of the advantages of the physics-informed approach: the network does not only interpolate the data, but is also constrained by the governing cosmological equation.

The evolution of the different loss terms also shows the training process. The total loss decreases rapidly at the beginning and then reaches a stable low value. The data loss does not vanish completely, which is expected because the data are noisy. The initial-condition loss becomes very small, showing that the initial value of the scale factor is well enforced. The physics loss also decreases by several orders of magnitude, meaning that the learned solution becomes increasingly compatible with the Friedmann equation. The small oscillations or spikes visible in the curves are typical of neural-network optimization and do not prevent convergence.

Finally, the learned cosmological parameters converge toward the true values used to generate the synthetic data. Starting from initial values close to $\Omega_m = \Omega_\Lambda = 0.5$, the model progressively learns values close to

$$\Omega_m \simeq 0.30, \qquad \Omega_\Lambda \simeq 0.70. \tag{89}$$

This confirms that the PINN is able to solve not only a direct problem, namely reconstructing $a(\tau)$, but also an inverse problem, namely inferring the physical parameters entering the Friedmann equation.

K. Interpretation of the final result

At the end of training, the quality of the model can be assessed in three ways.

First, one compares the learned scale factor $a_\theta(\tau)$ with the clean analytical solution and with the noisy data. A good PINN should smooth out part of the observational noise while remaining close to the underlying physical solution.

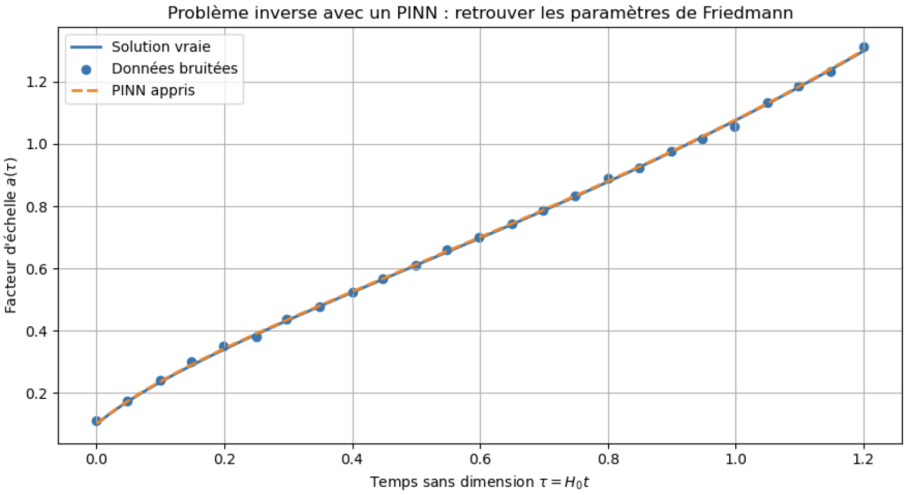
Second, one studies the evolution of the loss during training. A decreasing total loss indicates that the network progressively improves both its data fitting and its physical consistency. However, as in the harmonic oscillator example, the loss does not need to decrease perfectly monotonically. Neural-network optimization may display fluctuations or spikes.

Third, one compares the learned cosmological parameters with the true values used to generate the synthetic data:

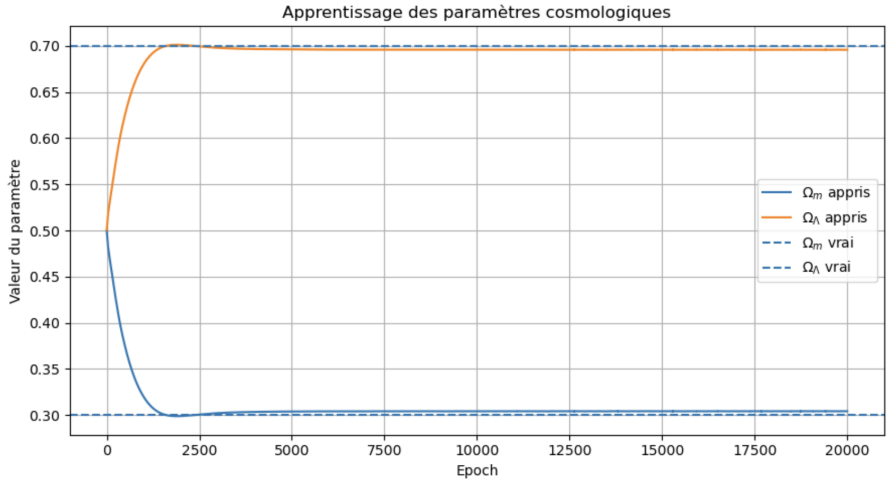
$$\Omega_m^{\text{learned}} \simeq \Omega_m^{\text{true}}, \qquad \Omega_\Lambda^{\text{learned}} \simeq \Omega_\Lambda^{\text{true}}. \tag{90}$$

If this agreement is good, the model has successfully solved an inverse cosmological problem.

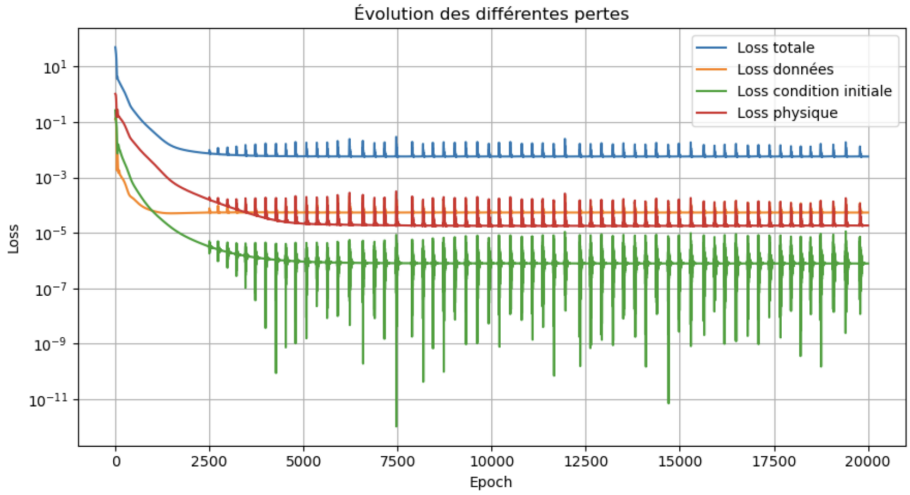
This example illustrates why PINNs are interesting for cosmology. They do not only interpolate data. They combine observations, initial conditions and theoretical equations in a single optimization problem. In more realistic applications, the same logic can be extended to richer cosmological models, including radiation, curvature, scalar fields or inflationary dynamics.



(a) Reconstruction of the scale factor $a(\tau)$. The PINN prediction follows the true solution while smoothing the noisy data.



(b) Learning of the cosmological parameters. The learned values of Ω_m and Ω_Λ converge toward the true values.



(c) Evolution of the total loss, data loss, initial-condition loss and physics loss during training.

FIG. 4: Results of the PINN applied to the inverse Friedmann problem. The network reconstructs the scale factor $a(\tau)$ from noisy data while enforcing the Friedmann equation at collocation points. The cosmological parameters Ω_m and Ω_Λ are learned together with the neural-network parameters.

CONCLUSION

In this seminar, we introduced Physics-Informed Neural Networks as a machine-learning framework for physical systems governed by differential equations. We first reviewed the basic structure of neural networks, from artificial neurons and activation functions to loss minimization and training. We then showed how the PINN approach extends classical supervised learning by adding the governing differential equation directly into the loss function.

Simple examples such as Newton's law of cooling and the harmonic oscillator illustrated the central idea of the method. In particular, the harmonic oscillator showed that a PINN does not automatically converge to the correct solution: the quality of the result depends on the training duration, the optimization process and the balance between the different loss terms. This emphasized that PINNs should not be treated as black-box solvers, but as numerical tools whose convergence must be carefully checked.

We then introduced the geometric framework of general relativity and the FLRW metric, leading to the Friedmann equations. In this cosmological setting, the PINN was used to reconstruct the scale factor $a(\tau)$ from noisy synthetic data while enforcing the Friedmann equation at collocation points. The same model was also able to infer the cosmological parameters Ω_m and Ω_Λ, showing how PINNs can address both direct and inverse problems.

The main conclusion is that PINNs provide a useful bridge between data-driven methods and theoretical physics. They are especially interesting when data are sparse or noisy but the governing equations are known. However, they are not a universal replacement for standard numerical solvers: their training can be unstable, computationally expensive and sensitive to hyperparameters. In cosmology, they should therefore be understood as a complementary tool, with possible extensions toward more realistic models including radiation, curvature, scalar fields or inflationary dynamics.

LICENSING

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. **378**, 686 (2019), doi:10.1016/j.jcp.2018.10.045.

[2] I. E. Lagaris, A. Likas, and D. I. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Networks **9**, 987 (1998), doi:10.1109/72.712178.

[3] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz, Data-driven discovery of partial differential equations, Sci. Adv. **3**, e1602614 (2017), doi:10.1126/sciadv.1602614.

[4] B. Moseley, A. Markham, and T. Nissen-Meyer, Solving the wave equation with physics-informed deep learning, arXiv preprint arXiv:2006.11894 (2020), arXiv:2006.11894.

[5] S. M. Carroll, *Spacetime and Geometry: An Introduction to General Relativity* (Addison-Wesley, 2004).

[6] S. Weinberg, *Gravitation and Cosmology: Principles and Applications of the General Theory of Relativity* (John Wiley and Sons, 1972).

[7] A. Friedmann, Über die krümmung des raumes, Zeit. Phys. **10**, 377 (1922), doi:10.1007/BF01332580.

[8] Planck Collaboration, N. Aghanim, *et al.*, Planck 2018 results. vi. cosmological parameters, Astron. Astrophys. **641**, A6 (2020), arXiv:1807.06209.

[9] PyTorch Contributors, Automatic differentiation package: torch.autograd, `https://pytorch.org/docs/stable/autograd.html`, accessed: 2026-04-30.