Implicit function theorem in Physics-Informed Neural Networks to solve parameterized differential equations

Julien Marie-Anne[*]
julien.marie-anne@ens-paris-saclay.fr

Cyriaque Rousselot[*]
cyriaque.rousselot@inria.fr

Nilo Schwencke[*]
nilo.schwencke@protonmail.com

Alena Shilova[*]
alena.shilova@inria.fr

Abstract

Physics-informed neural networks (PINNs) have shown promising results in solving partial differential equations (PDEs). Nevertheless, for some challenging PDEs, standard PINNs can fail to converge. We propose a novel curriculum learning strategy that addresses this limitation. Our method leverages an extension of the implicit function theorem to guide the training process along the solution manifold of the parameterized differential equation, starting from an easy-to-solve problem and progressively moving towards a hard-to-solve one. We establish a theoretical link between our approach and natural gradient descent, giving rise to a new effective curriculum learning algorithm allowing us to solve difficult PDEs such as Eikonal and Hamilton Jacobi-Bellman equations.

1 Introduction

Physics-informed neural networks (PINNs) (Raissi, Perdikaris, and Karniadakis 2019) are a powerful tool for approximating solutions to differential equations, which are ubiquitous in science and engineering. Many real-world problems involve not a single equation, but a parameterized family of partial differential equations (PDEs). Typically, a parameter in such PDEs represents some physical property of the environment, e.g. a viscosity coefficient in the equations of fluid dynamics. At the same time, the difficulty of finding a solution of a PDE with a PINN often depends on these parameters, and for certain parameter values, standard PINNs fail to converge (Krishnapriyan et al. 2021). To address this challenge, we propose a novel curriculum learning strategy. Our method leverages an extension of the implicit function theorem to navigate the solution space of parameterized PDEs represented by a graph that connects a parameter of a PDE with its corresponding solution. By starting with an "easy" problem, we progressively move towards the "hard" target problem, using the structure of the solution manifold to guide the training process.

Therefore, the main contributions of this work are twofold. (i) We extend the implicit function theorem for parameterized differential equations, which provides a principled way to perform curriculum learning by moving along the "solution curve". (ii) We provide a theoretical connection between our method and natural gradient descent (Amari 1998), leading to an efficient implementation of a curriculum learning algorithm for solving parameterized differential equations.

2 Optimization of PDEs via Physics-Informed Neural Networks

Let $\Omega \subset \mathbb{R}^d$ be a domain. We consider a differential equation $D[u] = f$ in Ω, where $D : \mathcal{H} \to L^2(\Omega)$ is a differential operator on a Hilbert space $\mathcal{H}$ of real-valued functions on Ω. For ease of presentation,

[*]Inria-Saclay, LISN, Paris-Saclay University, 91190, Gif-sur-Yvette, France.

1st Workshop on Differentiable Systems and Scientific Machine Learning @ EurIPS 2025.

we omit boundary conditions here, but our method extends to them (see appendix G). PINNs approximate the solution u with a neural network u_θ by minimizing the empirical loss

$$\hat{\ell}(\theta) = \frac{1}{2S_D} \sum_{i=1}^{S_D} \left(D[u_\theta](x_i^D) - f(x_i^D) \right)^2, \tag{1}$$

where $\{x_i^D\}_{i=1}^{S_D}$ are points sampled in Ω.

Optimization in Function Space Standard optimizers like Adam can struggle with the non-convex loss landscape of PINNs (De Ryck et al. 2023). While L-BFGS is a common alternative (Raissi, Perdikaris, and Karniadakis 2019), recent work leverages the function space geometry through natural gradient descent (NGD) (Schwencke and Furtlehner 2024; Müller and Zeinhofer 2024). NGD considers the functional loss $\mathcal{L}(v) = \frac{1}{2}\|v - f\|^2_{L^2(\Omega)}$, whose L^2 gradient is $\nabla\mathcal{L} = v - f$. The NGD update projects this ideal gradient onto the tangent space of the neural network's output manifold, then maps it back to the parameter space. The update rule is

$$\theta_{k+1} = \theta_k - \eta_k d_\theta D[u(\cdot)]^\dagger_{|\theta_k} \left[\Pi_{\mathcal{T}^{(D)}_{\theta_k}} \left(\nabla\mathcal{L}(D[u_{|\theta_k}]) \right) \right], \tag{2}$$

where $d_\theta D[u(\cdot)]^\dagger$ is the pseudoinverse of the differential of the operator composed with the network, and $\Pi_{\mathcal{T}^{(D)}_{\theta_k}}$ is the projection onto the tangent space (see Appendix B). Directly computing this update is often intractable. The ANaGRAM method (Schwencke and Furtlehner 2024) provides a scalable empirical approximation, which we build upon.

3 Curriculum Learning for Parameterized PDEs

Parameterized Differential Equations We lift the classical PDE $D[u] = f$ in Ω to a parameterized family by letting the operator depend on a scalar $\alpha \in \mathcal{A} \subset \mathbb{R}$:

$$D[\alpha, u] = f \quad \text{in } \Omega. \tag{3}$$

Fixing α recovers the classical setting, while varying α induces a solution map $\alpha \mapsto u_\alpha$ under standard well-posedness assumptions. Our goal is to reach a hard target $\boldsymbol{\alpha_1}$ starting from an easy $\boldsymbol{\alpha_0}$ by following the solution manifold $\{(\alpha, u) : D[\alpha, u] = f\}$ (see Figure 1); boundary operators can be incorporated analogously (Appendix G).

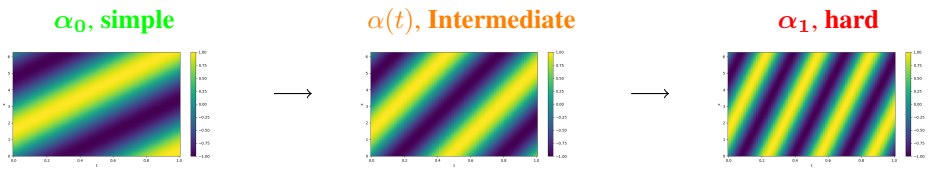


Figure 1: Three solutions of the convection differential equation (see Appendix A) for three different values of α. Training a PINN directly on $\boldsymbol{\alpha_1}$ leads to disappointing results (Krishnapriyan et al. 2021). In the framework of curriculum learning, we start at $\boldsymbol{\alpha_0}$ and progressively increase α to attain our goal $\boldsymbol{\alpha_1}$.

Curriculum learning Curriculum learning (Bengio et al. 2009) consists in training a model on a sequence of tasks whose difficulty increases progressively. Krishnapriyan et al. 2021 demonstrate that in some parametrized families, standard PINNs succeed only within favorable parameter regimes, while their performance deteriorates in more challenging settings. By incorporating curriculum learning into the PINN framework, they achieve accurate approximations even for challenging parameters. The proposed strategy starts by training the PINN to solve the parameterized differential equation at an initial, easily solvable parameter α_0. Then, for each new alpha, until the target parameter $\boldsymbol{\alpha_1}$ is reached, we train the PINN to learn a new solution by starting from the previously trained weights, thereby guiding the model through increasingly difficult regimes. This procedure is formalized in Algorithm 3.

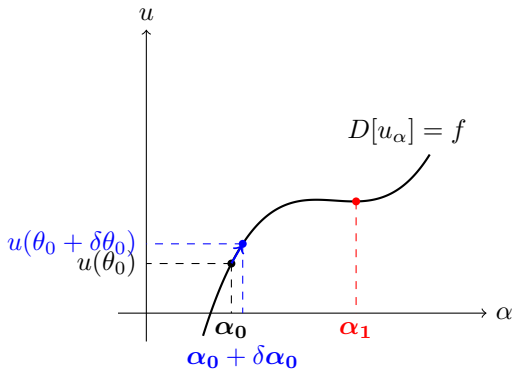


Figure 2: Illustration of implicit function theorem. The graph corresponds to the solution u_α for the corresponding differential equation with parameter α. We start at a point $(\boldsymbol{\alpha_0}, \mathbf{u_0})$, and the goal is to calculate $\mathbf{u_1}$ at point $\boldsymbol{\alpha_1}$ using implicit function theorem, by moving along the solution curve.

4 Our method

We propose Implicit Curriculum Learning, a curriculum learning strategy for PINNs that leverages the implicit function theorem. Similarly to classical curriculum learning, we incrementally find the solution for a target parameter $\boldsymbol{\alpha_1}$ by starting from a known solution $\mathbf{u_0}$ corresponding to a parameter $\boldsymbol{\alpha_0}$. The difference is the implicit construction of a continuous path of solutions that follows the evolution from $(\boldsymbol{\alpha_0}, \mathbf{u_0})$ to $(\boldsymbol{\alpha_1}, \mathbf{u_1})$ ensuring the smoothest transition.

Let $\boldsymbol{\alpha_0}, \boldsymbol{\alpha_1} \in \mathcal{A}$ be the starting and terminal parameters, respectively. We assume that we have a function $\mathbf{u_0}$ that is a valid solution to the differential equation (3) for $\alpha = \boldsymbol{\alpha_0}$. The central question we address is: *can we build a sequence of* α_k *and* $u_k \in \mathcal{H}$ *with* $K+1$ *elements, where* $\alpha_0 = \boldsymbol{\alpha_0}$ *and* $\alpha_K = \boldsymbol{\alpha_1}$ *so that for any* $k > 1$, u_{k-1} *can be used for finding* u_k*?*

The implicit function theorem provides a way to navigate the solution manifold with respect to the parameter α, as illustrated in Figure 2. To formalize this, we introduce a time-like variable $t \in [0, 1]$ and define continuous paths for the parameter and the solution:

$$\alpha : [0,1] \to \mathcal{A}, \quad t \mapsto \alpha(t), \qquad u : [0,1] \to \mathcal{H}, \quad t \mapsto u(t). \tag{4}$$

These paths are constructed such that $\alpha(0) = \boldsymbol{\alpha_0}$, $\alpha(1) = \boldsymbol{\alpha_1}$, and $u(0) = \mathbf{u_0}$. The crucial constraint is that for all $t \in [0, 1]$, $u(t)$ must remain a valid solution to the differential equation for the corresponding parameter $\alpha(t)$, i.e., $D[\alpha(t), u(t)] = f$. This effectively creates a path in the solution space that connects the initial solution $\mathbf{u_0}$ to the target solution $\mathbf{u_1}$. We state the following theorem.

Theorem 1 (Implicit function theorem in parameter space)**.** *Let* $\alpha : [0,1] \to \mathcal{A}$*, a function that verifies* $\alpha(0) = \boldsymbol{\alpha_0}$*, and* $\alpha(1) = \boldsymbol{\alpha_1}$*, with* $\mathcal{A} \subset \mathbb{R}$*. Let* $\theta : [0,1] \to \mathbb{R}^P$ *the parameters of the parametric model such that* $u : \begin{cases} [0,1] & \to \mathcal{H} \\ t & \mapsto u_{|\theta(t)} \end{cases}$ *is a solution of the problem 3 with* α*. Suppose* $u(0) = \mathbf{u_0}$*, we have*

$$\dot{\theta}(t) = -d_\theta \left(D[\alpha(t), u(\cdot)] \right)^\dagger_{|\theta(t)} \left(\Pi^\perp_{\mathcal{T}^{(D)}_{\theta(t)}} \left(d_\alpha D \left[\dot{\alpha}(t), u(t) \right]_{|\alpha(t)} \right) \right), \tag{5}$$

which is optimal in the least squares sense for the minimal norm, where $\mathcal{T}^{(D)}_\theta$ *is the tangent space on the parametric model* $D \circ u(\cdot)$ *evaluated in* θ *(see appendix B).*

Theorem 1 provides the foundation for a numerical method to trace the solution path. By discretizing the time interval $[0, 1]$, we can employ an Euler method to approximate the evolution of the neural network parameters $\theta(t)$. Given an initial parameter set θ_0 for which u_{θ_0} is a solution for the PDE with parameter $\boldsymbol{\alpha_0}$, we can iteratively update the parameters to follow the solution curve. This leads to the following *implicit function theorem update* rule:

$$\theta_{k+1} = \theta_k + \eta_k \dot{\theta}_k = \theta_k - \eta_k d_\theta \left(D[\alpha(k), u(\cdot)] \right)^\dagger_{|\theta(k)} \left(\Pi^\perp_{\mathcal{T}^{(D)}_{\theta(k)}} \left(d_\alpha D[\dot{\alpha}(k), u(k)]_{|\alpha(k)} \right) \right), \tag{6}$$

where k corresponds to a discretization and η_k is the step size at iteration k.

Remark 1. This update rule bears a strong resemblance to the natural gradient descent (NGD) method (2). The key distinction lies in the term being projected. While NGD projects the gradient of the loss function, our update projects the derivative of the differential operator with respect to the parameter α. This projection guides the parameter update in a direction that stays on the solution curve as α changes infinitesimally.

5 Experimental results

In this section, we consider three parameterized differential equations. Details about the experimental settings can be found in Appendix D. Results are summarized in Table 1.

Eikonal equation We consider eikonal equation described in Appendix A.0.5, with viscosity parameter $\boldsymbol{\varepsilon_1} = 0$. This equation has an infinite number of generalized solutions. However, we are particularly interested in the *viscosity solution*. Standard PINN training typically does not converge to the viscosity solution, meanwhile curriculum learning allows one to guide the model towards this viscosity solution. Furthermore, typical bad optimization behaviors leading to a large variance for Adam are described in the Appendix.

Burgers' equation We consider Burgers' equation described in Appendix A.0.4 with the target parameter $\boldsymbol{\nu_1} = 0.001$. This viscosity parameter is lower than the usual $0.01/\pi$ considered in the literature (Raissi, Perdikaris, and Karniadakis 2019, Urbán, Stefanou, and Pons 2025), which complicates the approximation of the solution in the nearly-discontinuous regions.

Hamilton-Jacobi-Bellman equation We consider a particular case of Hamilton-Jacobi-Bellman equation, coming from optimal control field, as described in Appendix A.0.6. Like for the eikonal equation, we are interested in the notion of viscosity solution that we target by parameterizing the differential equation (by adding a $\varepsilon\Delta u$ regularization term, see Appendix E.3).

Table 1: Comparison of methods on different PDEs

Equation	Method	Metric
Hamilton–Jacobi–Bellman (Relative Error)	Adam	5.56e-01 $\pm$ 1.99e-01
	Implicit Curriculum Learning	**3.44e-01 $\pm$ 2.05e-01**
Eikonal (Relative Error)	Adam	8.02e-01 $\pm$ 9.78e-01
	Implicit Curriculum Learning	**2.05e-02 $\pm$ 1.09e-02**
Burgers (Evaluation Loss)	Adam	1.32e-02
	Implicit Curriculum Learning	**7.21e-03**

6 Discussion

Building on the findings of Krishnapriyan et al. 2021, we confirmed that curriculum learning can substantially boost the performance of PINNs when approximating families of parameterized differential equations. To this end, we introduced a novel curriculum learning strategy grounded in the implicit function theorem, which naturally connects to the geometry of natural gradient descent.

Our empirical evaluation demonstrates that this approach yields promising results. Unlike conventional curriculum learning schemes, the implicit function theorem based method critically relies on maintaining a *correct* solution for the current parameter value α_t at every stage t. Consequently, the choice of parameter increments (α_t to α_{t+1}) is crucial. In this work, we consider a few heuristical schedules of α_t, however an adaptive schedule could further enhance performance and is a compelling direction for future research.

Moreover, our experiments on Hamilton–Jacobi-Bellman equations revealed that interleaving standard optimization steps (e.g., Adam) with the implicit updates improves the estimate and also ensures that the approximation at time t is *correct*. Determining, in an adaptive manner, both when and how many conventional training iterations to apply before each implicit update offers another promising avenue for refining the method.

References

Amari, Shun-Ichi (1998). "Natural gradient works efficiently in learning". In: *Neural computation* 10.2, pp. 251–276.

Bardi, Martino, Italo Capuzzo Dolcetta, et al. (1997). *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Vol. 12. Springer.

Bengio, Yoshua et al. (2009). "Curriculum learning". In: *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48.

Boissaux, Marc and Jang Schiltz (2010). "An optimal control approach to portfolio optimisation with conditioning information". In: *LSF Research Working Papers Series 10-09*.

Bressan, Alberto (2011). "Viscosity solutions of Hamilton-Jacobi equations and optimal control problems". In: *Lecture notes*.

Crandall, Michael G and Pierre-Louis Lions (1983). "Viscosity solutions of Hamilton-Jacobi equations". In: *Transactions of the American mathematical society* 277.1, pp. 1–42.

De Ryck, Tim et al. (2023). "An operator preconditioning perspective on training in physics-informed machine learning". In: *arXiv preprint arXiv:2310.05801*.

Engl, Heinz W and Ronny Ramlau (2015). "Regularization of inverse problems". In: *Encyclopedia of applied and computational mathematics*. Springer, pp. 1233–1241.

Fleming, Wendell H and H Mete Soner (2006). *Controlled Markov processes and viscosity solutions*. Springer.

Fleming, Wendell H and Panagiotis E Souganidis (1986). "Asymptotic series and the method of vanishing viscosity". In: *Indiana University Mathematics Journal* 35.2, pp. 425–447.

Guo, Xian, Wei Zhu, and Yongchun Fang (2018). "Guided motion planning for snake-like robots based on geometry mechanics and HJB equation". In: *IEEE Transactions on Industrial Electronics* 66.9, pp. 7120–7130.

Krishnapriyan, Aditi et al. (2021). "Characterizing possible failure modes in physics-informed neural networks". In: *Advances in neural information processing systems* 34, pp. 26548–26560.

Müller, Johannes and Marius Zeinhofer (2023). "Achieving high accuracy with PINNs via energy natural gradient descent". In: *International Conference on Machine Learning*. PMLR, pp. 25471–25485.

– (2024). "Position: Optimization in SciML should employ the function space geometry". In: *arXiv preprint arXiv:2402.07318*.

Munos, Rémi (2000). "A study of reinforcement learning in the continuous case by the means of viscosity solutions". In: *Machine Learning* 40.3, pp. 265–299.

Panetta, John Carl and K Renee Fister (2000). "Optimal control applied to cell-cycle-specific cancer chemotherapy". In: *SIAM Journal on Applied Mathematics* 60.3, pp. 1059–1072.

Paszke, Adam et al. (2019). "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (2019). "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational physics* 378, pp. 686–707.

Rudin, W. (1991). *Functional Analysis*. International series in pure and applied mathematics. McGraw-Hill. ISBN: 9780070542365. URL: `https://books.google.fr/books?id=Sh_vAAAAMAAJ`.

Schwencke, Nilo and Cyril Furtlehner (2024). "ANAGRAM: a natural gradient relative to adapted model for efficient PINNS learning". In: *arXiv preprint arXiv:2412.10782*.

Shilova, Alena et al. (2024). "Learning HJB Viscosity Solutions with PINNs for Continuous-Time Reinforcement Learning". PhD thesis. Inria Lille-Nord Europe, CRIStAL-Centre de Recherche en Informatique, Signal ...

Urbán, Jorge F, Petros Stefanou, and José A Pons (2025). "Unveiling the optimization process of physics informed neural networks: How accurate and competitive can PINNs be?" In: *Journal of Computational Physics* 523, p. 113656.

A Parameterized differential equations

In this section, we present several differential equations that can be effectively addressed using the framework of curriculum learning. This list is not exhaustive. Rather, it serves to illustrate and justify the applicability of curriculum learning–based methods for approximating the solution of differential equations.

A.0.1 Heat equation

One of the simplest differential equation we consider is the heat equation. It is a linear differential equation defined by:

$$\begin{cases} \partial_t u - \alpha \partial_{xx} u &= 0 & \text{in } \Omega = [0,1] \times [0,1]. \\ u(0,x) &= \sin(\pi x) & \text{for } x \in [0,1]. \end{cases} \tag{7}$$

where α is a parameter representing the thermal diffusivity. With high thermal diffusivity, the heat moves rapidly, while it is the opposite when the value is small.

A.0.2 1D Convection equation

We formalize the one-dimensional convection problem as a hyperbolic PDE which represents the transport of a quantity:

$$\begin{cases} \partial_t u + \beta \partial_x u &= 0, \quad x \in \Omega, t \in [0,T], \\ u(x,0) &= g(x), \quad x \in \Omega, \end{cases} \tag{8}$$

where β is a parameter of velocity, $\Omega \subset \mathbb{R}$. We can show that such equation has the following analytical solution:

$$u^*(x,t) = F^{-1}\left(F(g(x))e^{-i\beta k t}\right), \tag{9}$$

where F is the Fourier transform, $i^2 = -1$ and k is the frequency in the Fourier domain. For more reference on the solution and the equation, one can see Krishnapriyan et al. 2021.

In practice, we choose $\Omega = [0, 2\pi)$, and $T = 1$, and we will use the boundary and periodic equations:

$$u(x,0) = \sin(x), \tag{10}$$

$$u(0,t) = u(2\pi,t). \tag{11}$$

Figure 1 shows the solutions of the 1D convection equation for different values of β. As mentioned previously, the solution is harder to approximate with PINN when the velocity β becomes large.

A.0.3 Reaction-Diffusion

We introduce the reaction-diffusion differential equation, which often appears in chemistry, biology and physics. In the context of chemistry, it models the chemical reaction between substances across time and the spacial diffusion of reactants. In 1D, we define it as follows:

$$\begin{cases} \partial_t u - \nu \partial_{xx} u - \rho u(1-u) &= 0, \quad x \in \Omega, t \in [0,T], \\ u(x,0) &= g(x), \quad x \in \Omega, \end{cases} \tag{12}$$

with $\Omega \subset \mathbb{R}$, and $\nu, \rho \in \mathbb{R}$ two parameters. For further details about the reaction-diffusion and its solution, the reader is referred to Krishnapriyan et al. 2021.

In practice, we set $\Omega = [0, 2\pi)$, and $T = 1$, and we use the boundary and periodic equations:

$$u(x,0) = \sin(x), \tag{13}$$

$$u(0,t) = u(2\pi,t). \tag{14}$$

A.0.4 Burgers' equation

Burgers' equation comes originates from the fluid dynamic field and is a convection-diffusion equation that can be defined with boundary conditions as:

$$\begin{cases} \partial_t u + u \partial_x u - \nu \partial_{xx} u &= 0 \quad \text{in } [0,1] \times [-1,1] \\ u &= 0 \quad \text{in } [0,1] \times \{-1,1\} \\ u(0,x) = -\sin(\pi x) &= 0 \quad \text{for } x \in [-1,1]. \end{cases} \tag{15}$$

In particular, the viscosity coefficient ν in Burgers' equation defines the difficulty of solving the differential equation. The lower ν, the stiffer the derivative in the middle, the harder the approximation. This equation was also considered in the founding paper of Physics-Informed Neural Network in Raissi, Perdikaris, and Karniadakis 2019. Using the algorithm L-BFGS, it was solved with viscosity coefficient $\nu = 0.01/\pi \approx 0.0032$. Schwencke and Furtlehner 2024 also consider Burgers' equation with $\nu = 0.001$, however overfitting occurs, and the solution is shifted near the shock, which shows the difficulty of learning the solution directly with very low coefficient term. Burgers' equation is frequently used as a benchmark for PINN algorithms in the literature, that is the reason why we included it. We observe that curriculum learning allows us to approximate the solution of Burgers' equation in an accurate way, even for viscosity coefficient equals to 0.001.

A reference solution for the Burgers equation when $\nu = 0.01/\pi$ is presented in Figure 3.

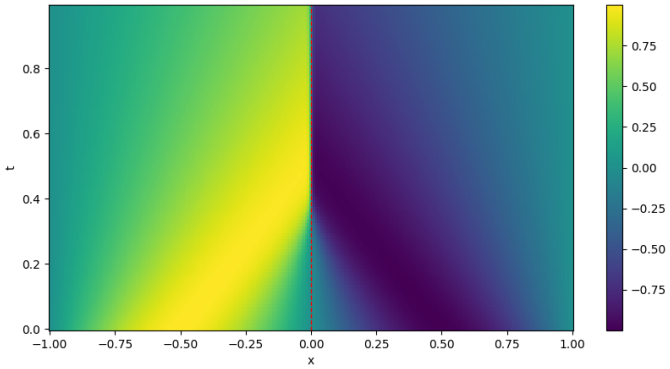


Figure 3: Reference solution for $\nu = 0.01/\pi$, obtained with EDTRK4 algorithm.

A.0.5 Eikonal equation

The eikonal equation is encountered in problems of wave propagations like in optics, describing the propagation of light, and can be written this way:

$$\begin{cases} |u'| & = 1 \quad \text{in } [-1, 1] \\ u(-1) = u(1) & = 0 \quad \text{on } \{\text{-1, 1}\}. \end{cases} \tag{16}$$

In particular, we consider a variant that we call the square eikonal equation

$$\begin{cases} |u'|^2 & = 1 \quad \text{in } [-1, 1] \\ u(-1) = u(1) & = 0 \quad \text{on } \{\text{-1, 1}\}. \end{cases} \tag{17}$$

It is easy to see that the two problems in (16) and (17) have exactly the same solutions. One can notice that currently, this problem is non-parameterized. However, it is often useful to consider a parameterized version of eikonal to obtain a special solution of eikonal equation, which is the viscosity solution. To understand why, first, we notice that the eikonal equation does not have smooth $C^1((0, 1))$ solution. Indeed, eikonal differential equation constraints the solution to have a linear function with slopes +1 or -1. However, the boundary conditions impose the function to be 0 at points $x = -1$ and $x = 1$. It is impossible to find a linear function with two points at 0 when the slope is $+1$ or -1 (a linear function has only one root). In the class of piece-wise smooth solutions, it is easy to see that eikonal equation has infinitely many solutions. Piecewise linear functions with slope +1 or -1 that verify the boundary conditions are such solutions, and three of them are shown in the Figure 4.

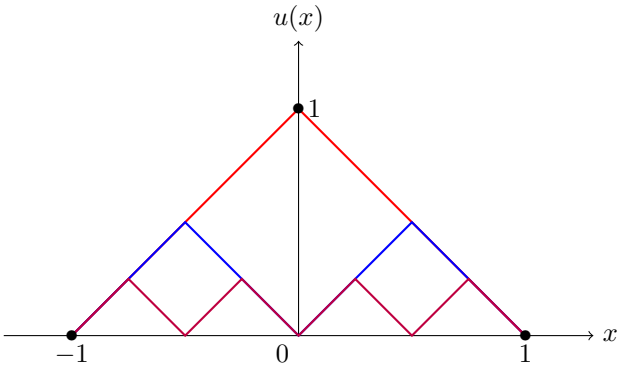


Figure 4: Several solutions of eikonal equation, represented with different colors.

In practice, finding a solution directly using PINN does not work: the convergence is almost impossible, and even if it converges, we do not really know what solution the neural network finally approximates between all the possible ones. The solution in red: $u^*(x) := 1 - |x|$ is the simplest of all, with only one sharp corner. In particular, we can show that this solution is also a viscosity solution (see section A.0.6). Leveraging the theory of viscosity (Fleming and Souganidis 1986), we can define the differential equation:

$$\begin{cases} |u'|^2 - \varepsilon u'' & = 1 \quad \text{in } [-1,1] \\ u(-1) = u(1) & = 0 \quad \text{on } \{\text{-1, 1}\} \end{cases}, \tag{18}$$

where $\varepsilon > 0$ is a real parameter. This equation admits a smooth solution:

$$u_\varepsilon(x) = \varepsilon \left[\log\left(\cosh\left(\frac{1}{\varepsilon}\right)\right) - \log\left(\cosh\left(\frac{x}{\varepsilon}\right)\right) \right]. \tag{19}$$

Also, for all $x \in [-1,1]$, $\lim\limits_{\varepsilon \to 0^+} u_\varepsilon(x) = u^*(x) := 1 - |x|$. Equation 18 can be seen as a smooth approximation of eikonal, and in particular, using PINN it is possible to find a valid solution when viscosity term ε has high value. Thus, eikonal equation can fall in curriculum learning setting where we start from a high value ε where it is easier to approximate the solution, and then we can make ε decrease to 0 until we reach the solution u^*. We can see in Figure 5 the different solutions when we move ε, along with the target solution u^*.

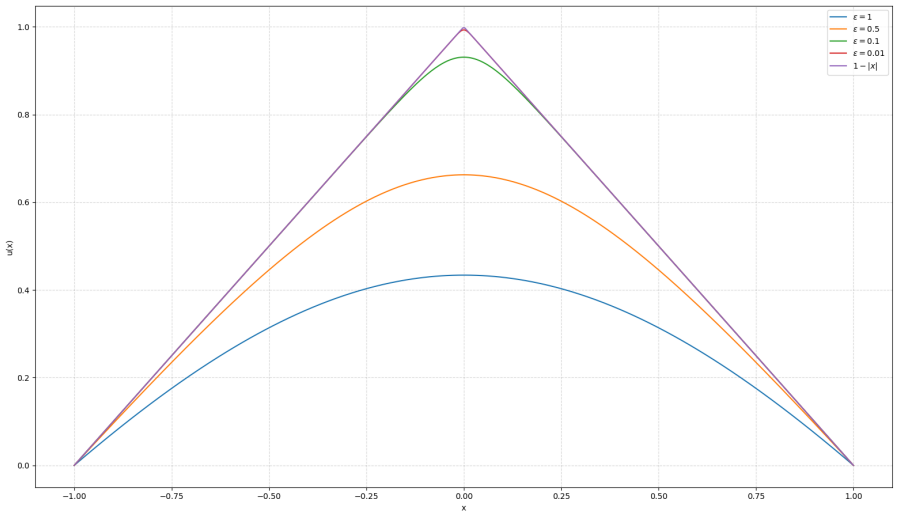


Figure 5: Solutions to smooth eikonal (18) with different value of ε.

A.0.6 Hamilton-Jacobi-Bellman equation

In this section, we define the Hamilton-Jacobi-Bellman equation from optimal control field. Optimal control is a field that consists in studying a dynamical system, by finding a control that optimizes an objective function. This section only gives a summary of this field, for more information, one can consult Fleming and Soner 2006, Bardi, Dolcetta, et al. 1997. First, we start by considering a continuous time set $\mathbb{R}_+$ which indicates the evolution of our system according to the time. Let $x : \mathbb{R}_+ \longrightarrow \mathcal{X}$ which gives the state along the time, where $\mathcal{X} \subset \mathbb{R}^N$ represents the state space. The *state dynamic* is governed by the following equation:

$$\begin{cases} y'(t) = f(y(t), \psi(t)) \quad , \quad t > 0 \\ y(0) = x \end{cases} \tag{20}$$

where f is fixed and known and is called *the dynamic*, and $\psi : \mathbb{R}_+ \longrightarrow \mathcal{A}$ is the control function in a function space Ψ, with $\mathcal{A} \subset \mathbb{R}^A$ the action space. The control ψ is a function that encodes our strategy to manipulate the state $y(t)$ through (20), in such way to maximize

$$J(x, \psi) := \int_0^\tau \gamma^t r(y(t), \psi(t))dt + \gamma^\tau R(y(\tau)), \tag{21}$$

where $r : \mathcal{X} \times \mathcal{A} \longrightarrow \mathbb{R}$ is a known function such that $r(x, a)$ represents the reward obtained in state y after doing action a, and $y(t)$ is the state after time t, when following the control ψ along all the trajectory. In the formula, we also consider a boundary term $\gamma^\tau R(y(\tau))$ where τ is an exit time representing the time where we leave the interior of the state space (in particular it can be $\tau = +\infty$). After reaching the boundary of the state space, we receive a boundary reward $R(y_x(\tau))$. The term γ^t with $\gamma \in (0, 1)$ is here to ensure that the integral is finite, and is called discount factor. The interpretation is that the reward is less important for high value of t. We finally introduce the value function as:

$$V(x) := \sup_{\psi \in \Psi} J(x, \psi). \tag{22}$$

The value function is thus the maximum cumulative reward we can obtain. If the sup is reached by an ψ^*, we call such function an optimal control.

Remark 2. Usually, optimal control is described in such way that we minimize J instead of maximizing it like in (22), by replacing the reward r by a cost l. We choose here to go in the other direction to have a context of "reinforcement learning", where we want to maximize a reward. We can pass from one formulation to the other just by choosing the cost $l(x, a) = -r(x, a)$.

By assuming we know the dynamic f and the reward r, the optimal control questions that we consider in this section are

1. What are the different strategies to calculate this value function.
2. How to deduce an optimal control from a value function.

Example 1 (Simple example from Munos 2000)**.** We consider a one-dimensional state space $[0, 1]$. The control $\psi(t)$ at time t can take only two values, $-1, +1$, and the state dynamics are given by $f(y, a) = a$ for $a \in \{-1, +1\}$. The reward is defined as $r(y, a) = 0$ for $y \in (0, 1)$, while at the boundaries we have $R(0) = R_0$ and $R(1) = R_1$, with $R_0, R_1 > 0$.

This means that, starting from an initial position $y(0) = x$, the goal is to choose actions $+1$ or -1 in such a way to reach one of the boundaries as quickly as possible to obtain a positive reward. From the dynamics, choosing $+1$ moves the state towards $y = 1$, while choosing -1 moves it towards $y = 0$.

It is quite clear that the optimal control consists of consistently choosing either $+1$ or -1, since switching actions only increases the time to reach a boundary. In other words, the value function is

$$V(x) = \max_{a \in \{-1,+1\}} \gamma^{\tau} R(y_x(\tau)), \tag{23}$$

where τ is the exit time to reach the boundary starting from x.

If we start at x and take the constant action $+1$, the dynamics are $y'(t) = 1$, which integrates to $y(t) = x + t$. The boundary $y = 1$ is reached when $x + t = 1$, i.e., at time $\tau = 1 - x$. Similarly, if we choose the constant action -1, we reach the boundary $y = 0$ at time $\tau = x$.

Thus, the value function can be written explicitly as

$$V(x) = \max(\gamma^x R_0, \gamma^{1-x} R_1). \tag{24}$$

Apart from this simple example, optimal control has many real-world applications, for instance, in robotics for the snake gait (Guo, Zhu, and Fang 2018), or in physical systems such as a pendulum, where by applying a force we can make it stand still in the upward position. Panetta and Fister 2000 considered the use of optimal control in cancer chemotherapy, while Boissaux and Schiltz 2010 showed that it can also be applied to finance.

Thus, there are many problems that fall within the optimal control framework. Each formulation is associated with a value function, that corresponds to the maximum cumulative reward achievable with an optimal policy. This value function is particularly interesting to compute because it naturally yields the optimal control: for a given state x, the optimal action is the one that drives the system toward the state with the highest value function. In other words, the state should evolve along the gradient of the value function.

In the following, we present a result showing that the value function satisfies a first-order nonlinear differential equation known as the Hamilton–Jacobi–Bellman (HJB) equation. As discussed above, since one can define infinitely many optimal control problems with their associated value functions, it is more accurate to say that there exists a whole class of Hamilton–Jacobi–Bellman equations, each depending on the specific problem. We also discuss some difficulties that arise when trying to approximate such equations, due to the existence of infinitely many generalized solutions, and we introduce the vanishing viscosity lemma, which provides a potential way to overcome this issue.

Theorem 2 (Hamilton-Jacobi-Bellman equation)**.** *If the value function V is differentiable at x, then it verifies the HJB equation:*

$$\log(\gamma) V(y) + \sup_{\psi \in \Psi} \{\nabla V(y) \cdot f(y, u) + r(y, u)\} = 0. \tag{25}$$

Remark 3 (Interpretation of HJB equation). A proof of the HJB equation can be found in Fleming and Soner 2006. However, we are going to give an intuitive interpretation of this differential equation. Let us suppose that V is differentiable, thus by theorem 2, the HJB equation is verified by the function V. In particular, we have:

$$V(y) = \frac{1}{\log\left(\frac{1}{\gamma}\right)} \sup_{a \in \mathcal{A}} \{\nabla V(y) \cdot f(y, a) + r(y, a)\}. \tag{26}$$

A theorem called *verification theorem* shows that the optimal action to take for a given state y is the one that verifies the sup on the right hand side of the equation (see Fleming and Soner 2006). This also validates the intuition we provided earlier, because it consists in finding the action $a \in \mathcal{A}$ such that the state evolves in the direction of $\nabla V(x)$ (the steepest ascent of the value function) while balancing this with the instantaneous reward $r(y, a)$.

Several problems arise from the HJB equation. First, the value function is not necessarily differentiable everywhere. In example 1, the value function is not differentiable everywhere because of the max. Another problem is that if we consider generalized solutions, which are solutions that verify the HJB equation almost everywhere, then often there is an infinite number of them. Again, we can see this problem in the simple example 1 where the HJB equation can be written:

$$\log(\gamma)V(y) + \max(V'(y), -V'(y)) = 0, \quad \text{for } y \in (0, 1). \tag{27}$$

with the boundary condition $V(0) \geq R_0$ and $V(1) \geq R_1$. Munos 2000 showed that this differential equation admits several generalized solutions. Thus, if we apply PINNs to approximate a solution of the differential equation, it is not clear to which solution the method is going to converge. In particular, we are specifically interested in the value function, rather than other possible solutions. Our goal is therefore to find this value function so that we can target it directly when solving the HJB equation with PINNs.

To achieve this, we make use of the theory of viscosity solutions developed in Crandall and Lions 1983. For more details, see Fleming and Soner 2006. The key idea is to define a generalized solution with desirable properties such as uniqueness. We will not provide a formal definition of viscosity solutions here, as this has already been extensively developed in the literature. A gentle introduction can be found in Bressan 2011. An important result from this theory is that, for the HJB equation, under certain assumptions such as the continuity of the state dynamics f and the reward function r , the value function is the unique viscosity solution.

Thus, solving the optimal control problem amounts to targeting the unique viscosity solution of the HJB equation. At the best of our knowledge, Shilova et al. 2024 is the only work that use PINNs to target specifically this viscosity solution on the HJB equation. For this, they make use of the vanishing viscosity lemma that consists in considering the differential equation:

$$\log(\gamma)W_\varepsilon(y) + \sup_{\psi \in \Psi}\{\nabla W_\varepsilon(y) \cdot f(y, u) + r(y, u)\} = \varepsilon \Delta W_\varepsilon(y). \tag{28}$$

This differential equation is similar to the HJB equation except that we add a regularization term $\varepsilon \Delta W_\varepsilon(y)$ for a fixed $\varepsilon > 0$. Under some assumption, Fleming and Soner 2006 show that the solution of this new differential equation is unique, and that it is satisfies the vanishing viscosity lemma: if W_ε converges uniformly to a function W and $\varepsilon \Delta W_\varepsilon$ converges uniformly to 0, then the limit function W is the viscosity solution of the original HJB equation 25. Shilova et al. 2024 proposes thus to use curriculum learning by solving the equation 28 for different values of ε, as ε converges to 0 to obtain the corresponding function limit which should be the viscosity solution.

B Natural gradient descent and ANaGRAM

B.1 Natural Gradient Descent

Gradient descent has the general form:

$$\theta_{t+1} \longleftarrow \theta_t + \eta_t d_t, \tag{29}$$

where d_t is a descent direction. In general $d_t = \nabla \hat{\ell}(\theta)$, but such an update does not take into consideration the geometry of our function space (e.g. the neural networks function space). To explain the notion of natural gradient descent, we first introduce some definitions and notations (mostly taken from Schwencke and Furtlehner 2024)

Definition 1 (Parametric model)**.** Given a domain $\Omega \subset \mathbb{R}^n$ and a Hilbert space $\mathcal{H}$ containing functions defined on $\Omega \longrightarrow \mathbb{R}^m$, we define our parametric model as a differentiable functional:

$$u : \begin{cases} \mathbb{R}^P & \to \mathcal{H} \\ \theta & \mapsto (x \in \Omega \mapsto u(x; \theta)) \end{cases} \tag{30}$$

To prevent confusion, we write $u_{|\theta}(x)$ instead of $u(x; \theta)$ for all $x \in \Omega$.

Definition 2 (Differential of a parametric model)**.** Let $u : \mathbb{R}^P \longrightarrow \mathcal{H}$ be a parametric model, and $\theta \in \mathbb{R}^P$. The differential of the parametric model u with respect to θ is the differential:

$$du_{|\theta} = \begin{cases} \mathbb{R}^P & \to \mathcal{H} \\ h & \mapsto \sum_{p=1}^{P} h_p \frac{\partial u_\theta}{\partial \theta_p} \end{cases} \tag{31}$$

To simplify notations, we note $\partial_p u_\theta$ instead of $\frac{\partial u_\theta}{\partial \theta_p}$.

We can generalize gradient descent from the specific case of a parameter spaces $\mathbb{R}^P$ to any Hilbert space. To do so, let us consider a Hilbert space $\mathcal{H}$ equipped with its inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$. A (functional) loss on $\mathcal{H}$ is then any differential map $\mathcal{L} : \mathcal{H} \longrightarrow \mathbb{R}_+$. We can then consider the following minimization problem:

$$\min_{u \in \mathcal{H}} \mathcal{L}(u), \tag{32}$$

i.e. finding $u \in \mathcal{H}$ such that $\mathcal{L}(u)$ is minimal. Since $\mathcal{L}$ is differentiable, we can write for all $u, h \in \mathcal{H}$ its first order Taylor's expansion:

$$\mathcal{L}(u + h) = \mathcal{L}(u) + \mathrm{d}\mathcal{L}_u(h) + o(\|h\|_{\mathcal{H}}). \tag{33}$$

By Riesz-Fréchet representation theorem, we have that for all $u \in \mathcal{H}$, there is $\nabla \mathcal{L}_u \in \mathcal{H}$ such that for all $h \in \mathcal{H}$,

$$\mathrm{d}\mathcal{L}_u(h) = \langle \nabla \mathcal{L}_u, h \rangle_{\mathcal{H}}. \tag{34}$$

$\nabla \mathcal{L}_u$ is then precisely the *gradient* of $\mathcal{L}$ with respect to the metric of $\mathcal{H}$. Similarly to the $\mathbb{R}^P$ case, one way to minimize $\mathcal{L}$ is to perform a gradient descent. More precisely, considering an initial $\mathbf{u_0} \in \mathcal{H}$, we define the following sequence of u_{k+1} for $k \geq 0$ by:

$$u_{t+1} \leftarrow u_t - \eta_t \nabla_u \mathcal{L}(u_t). \tag{35}$$

In the following, we will investigate the particular case, where the functional loss is given by the regression of a function $f \in L^2(\Omega)$. We will see later how we can generalize this to PINNs and differential equations. The following proposition illustrates that this gradient descent is the most natural for a regression in $L^2(\Omega)$.

Proposition 1. *Let f a function in $L^2(\Omega)$, we define the loss for $u \in \mathcal{H}$ in the following way*

$$\mathcal{L}(u) := \frac{1}{2} \|u - f\|^2_{L^2(\Omega)}, \tag{36}$$

in such way that:

$$\nabla_u \mathcal{L}(u) = u - f. \tag{37}$$

Let u: $\mathbb{R}_+ \longrightarrow \mathcal{H}$, then the following ODE

$$\begin{cases} u(0) = \mathbf{u_0} \\ \frac{du}{dt}(t) = -\nabla_u \mathcal{L}(u) \end{cases} \tag{38}$$

has a unique solution

$$u(t) = \mathbf{u_0} + (1 - e^{-t})(f - \mathbf{u_0}). \tag{39}$$

Proof. Let $u(t)$ a solution of (38). In particular, we have:

$$u'(t) = -\nabla_u \mathcal{L}(u) = f - u(t). \tag{40}$$

By passing u to the left and multiplying both sides by e^t, we have:

$$e^t u'(t) + e^t u(t) = e^t f. \tag{41}$$

We can rewrite it:

$$(e^{\cdot} u(\cdot))'(t) = e^t f. \tag{42}$$

By integrating on both sides:

$$e^t u(t) = (e^t - 1) f + C, \tag{43}$$

from which we deduce:

$$u(t) = e^{-t} C + (1 - e^{-t}) f. \tag{44}$$

We also know that $u(0) = \mathbf{u_0}$, thus we get $C = \mathbf{u_0}$, we conclude:

$$u(t) = e^{-t} \mathbf{u_0} + (1 - e^{-t}) f = \mathbf{u_0} + (1 - e^{-t})(f - \mathbf{u_0}). \tag{45}$$

We verify for the converse that such $u(t)$ is a solution of the ODE (38).

□

In particular, the functional gradient descent (35) corresponds to the Euler approximation of the ODE (38), which in turn corresponds to regressing f by following in a straight line the segment $[\mathbf{u_0}, f]$. It is thus an ideal and desirable update. However, the update (35) implies that we can perform updates directly in the function space, which is not the case in practice. In a parametric model settings (e.g. a neural network), we update the function defined by the model *only* through the parameters. Doing a gradient descent in the parameter space yields the following:

$$\theta_{t+1} \leftarrow \theta_t + \eta_t w_t, \tag{46}$$

where w_t is a descent direction. If we write the Taylor's expansion of $u_{\theta_{t+1}} = u_{\theta_t + \eta_t w_t}$, we obtain:

$$u_{\theta_{t+1}} = u_{\theta_t} + \eta_t du_{\theta_t}(w_t) + o\left(\eta_t \|w_t\|\right). \tag{47}$$

One way to define the natural gradient update is to choose w_t in the parameter update in such way that the ideal functional update (35) is the closest to the Taylor expansion of $u_{\theta_{t+1}}$ defined in (47).

Thus, we can define w_t as the minimizer:

$$w_t \in \arg\min_{w \in \mathbb{R}^P} \frac{1}{2} \left\| du_{\theta_t}(w) + \nabla_u \mathcal{L}(u_{\theta_t}) \right\|^2_{L^2(\Omega)}. \tag{48}$$

The following proposition gives an expression for the descent direction w_t as defined in (48), which allow us to define the natural gradient descent.

Proposition 2 (Natural gradient direction, Müller and Zeinhofer 2024)**.** *Let*

$$\ell = \begin{cases} \mathbb{R}^P & \to \mathbb{R} \\ \theta & \mapsto \mathcal{L}(u_\theta) \end{cases} \tag{49}$$

The descent direction w_t defined in (48) *can be rewritten:*

$$w_t = -G(\theta_t)^\dagger \nabla_\theta \ell(\theta_t), \tag{50}$$

where $G : \mathbb{R}^P \to \mathbb{R}^P \times \mathbb{R}^P$ correspond to the Gram matrix for a given parameter θ. For $\theta \in \mathbb{R}^P$ it is defined by:

$$G(\theta)_{p,q} = \langle \partial_p u_\theta, \partial_q u_\theta \rangle_{L^2(\Omega)}. \tag{51}$$

Thus the natural gradient update can be formulated this way:

$$\theta_{t+1} \leftarrow \theta_t - \eta_t G(\theta_t)^\dagger \nabla_\theta \ell(\theta_t), \tag{52}$$

where $G(\theta)^\dagger$ refers to Moore-Penrose pseudoinverse, see appendix E.

Proof. Let's consider the descent direction w_t as defined in (48). We can simplify the expression by using the definition of the differential 2:

$$\frac{1}{2} \left\| du_{\theta_t}(w) + \nabla_u \mathcal{L}(u_{\theta_t}) \right\|^2_{L^2(\Omega)} = \frac{1}{2} \left\| \sum_{p=1}^P w_p \partial_p u_{\theta_t} \right\|^2_{L^2(\Omega)} + \sum_{p=1}^P w_p \left\langle \partial_p u_{\theta_t}, \nabla_u \mathcal{L}(u_{\theta_t}) \right\rangle_{L^2(\Omega)} \tag{53}$$

$$+ \frac{1}{2} \left\| \nabla_u \mathcal{L}(u_{\theta_t}) \right\|^2_{L^2(\Omega)}. \tag{54}$$

$$\tag{55}$$

The first term can be rewritten:

$$\frac{1}{2} \left\| \sum_{p=1}^P w_p \partial_p u_{\theta_t} \right\|^2_{L^2(\Omega)} = \frac{1}{2} \sum_{p=1}^P \sum_{q=1}^P w_p w_q \langle \partial_p u_{\theta_t}, \partial_q u_{\theta_t} \rangle_{L^2(\Omega)} \tag{56}$$

$$= \frac{1}{2} \sum_{p=1}^P \sum_{q=1}^P w_p G(\theta_t)_{p,q} w_q \tag{57}$$

$$= \frac{1}{2} \langle G(\theta_t) w, w \rangle_{\mathbb{R}^P}. \tag{58}$$

For the second term, we have by chain rule:

$$\partial_p \ell(\theta_t) = d_u \mathcal{L}(\partial_p u_{\theta_t}) \tag{59}$$

$$= \langle \partial_p u_{\theta_t}, \nabla_u \mathcal{L}(u_{\theta_t}) \rangle_{L^2(\Omega)}, \tag{60}$$

where the last inequality comes from the definition of the differential in L^2. Thus,

$$\sum_{p=1}^{P} w_p \left\langle \partial_p u_{\theta_t}, \nabla_u \mathcal{L}(u_{\theta_t}) \right\rangle_{L^2(\Omega)} = \langle w, \nabla_\theta \ell(\theta_t) \rangle_{\mathbb{R}^P}. \tag{61}$$

We can then rewrite the optimization problem:

$$w_t \in \arg\min_{w \in \mathbb{R}^P} \frac{1}{2} \langle G(\theta_t) w, w \rangle_{\mathbb{R}^P} + \langle w, \nabla_\theta \ell(\theta_t) \rangle_{\mathbb{R}^P} + \frac{1}{2} \left\| \nabla_u \mathcal{L}(u_{\theta_t}) \right\|^2_{L^2(\Omega)}. \tag{62}$$

This problem is convex in w, and putting the derivative to zero yields

$$G(\theta_t) w_t = -\nabla_\theta \ell(\theta_t). \tag{63}$$

We can take the Moore-Penrose inverse:

$$w_t = -G(\theta_t)^\dagger \nabla_\theta \ell(\theta_t). \tag{64}$$

For such w_t to be a valid solution to (63), we need to verify that $\nabla_\theta l(\theta_t) \in \operatorname{Im} G(\theta_t)$, in such way that we can apply proposition 6. Let $T_{\theta_t} := \operatorname{Im} du_{\theta_t} = \operatorname{Span}(\partial_p u_{\theta_t} \mid 1 \leq p \leq P) \subset L^2(\Omega)$, (we call it the tangent space). We can decompose $\nabla_u \mathcal{L}(u_{\theta_t}) = W_{\theta_t} + W_{\theta_t}^\perp$ where $W_{\theta_t} \in T_{\theta_t}$ and $W_{\theta_t}^\perp \in (T_{\theta_t})^\perp$. We can decompose further $W_{\theta_t} = \sum_{q=1}^P h_q \partial_p u_{\theta_t}$ for some $h \in \mathbb{R}^P$. By using (60), we have for all $1 \leq p \leq P$:

$$\partial_p \ell(\theta_t) = \langle \partial_p u_{\theta_t}, W_{\theta_t} + W_{\theta_t}^\perp \rangle_{L^2(\Omega)} \tag{65}$$

$$= \langle \partial_p u_{\theta_t}, W_{\theta_t} \rangle_{L^2(\Omega)} \quad \text{because } W_{\theta_t}^\perp \text{ is orthogonal to } \partial_p u_{|\theta_t} \tag{66}$$

$$= \langle \partial_p u_{\theta_t}, \sum_{q=1}^{P} h_q \partial_q u_{\theta_t} \rangle_{L^2(\Omega)} \tag{67}$$

$$= \sum_{q=1}^{P} \langle \partial_p u_{\theta_t}, \partial_q u_{\theta_t} \rangle_{L^2(\Omega)} h_q \tag{68}$$

$$= \sum_{q=1}^{P} G(\theta_t)_{pq} h_q \tag{69}$$

$$= (G(\theta_t) h)_p. \tag{70}$$

We deduce that

$$\nabla_\theta l(\theta_t) = G(\theta_t) h \in \operatorname{Im} G(\theta_t), \tag{71}$$

thus by proposition 6, $w_t = -G(\theta_t)^\dagger \nabla_\theta \ell(\theta_t)$ is a valid solution to (63), and is thus the minimizer of 48 □

From proposition 2, we conclude the following natural gradient update:

$$\theta_{t+1} = \theta_t - \eta_t G(\theta_t)^\dagger \nabla_\theta \ell(\theta_t) \tag{72}$$

B.1.1 Link with the classical gradient descent

The classical gradient descent is a special case of the natural gradient descent when the derivative of the parametric model $\partial_p u_\theta$ are orthonormals, i.e. $\langle \partial_p u_\theta, \partial_q u_\theta \rangle_{L^2(\Omega)} = \delta_{pq}$ for all $\theta \in \mathbb{R}^P$. Indeed, in this case, the gram Matrix $G(\theta) = I_P$.

Example 2 (Linear parametric model)**.** Let $(\phi_p)_{p=1}^P \in L^2(\Omega)$ an orthonormal basis. Let's define the parametric model as:

$$u_\theta := \sum_{p=1}^{P} \theta_p \phi_p, \tag{73}$$

then $\partial_p u_\theta = \phi_p$, and by using the fact that the ϕ_p are orthonormal, we deduce that $G(\theta) = I_P$ for all $\theta \in \mathbb{R}^P$.

B.1.2 Tangent space

To understand what happens in the functional space, we give two additional definitions:

Definition 3 (Image set of u)**.** We define $\mathcal{M}$ as the set of functions reached with the parametric model, i.e.

$$\mathcal{M} := \operatorname{Im} u := \{u_\theta \mid \theta \in \mathbb{R}^P\} \subset L^2(\Omega). \tag{74}$$

Definition 4 (Tangent space of u at θ)**.** We define $\mathcal{T}_\theta$ as the linear subspace of $L^2(\Omega)$ reached by $du_{|\theta}$, i.e.

$$\mathcal{T}_\theta := \operatorname{Im} du_{|\theta} = \text{Span}(\partial_p u_\theta \mid 1 \leq p \leq P) \subset L^2(\Omega). \tag{75}$$

The natural gradient update defined in (72) is the one that allows us to perform an update in the parameter space such that in the function space, the corresponding update is the closest to the ideal function update (35). The following theorem states formally this assertion:

Theorem 3 (Theorem 1 of Müller and Zeinhofer 2024)**.** *Let's suppose that θ_t is defined with the natural gradient update defined in* (72)*. Then, in the function space we have:*

$$u_{\theta_{t+1}} = u_{\theta_t} - \eta_t \Pi^{\perp}_{\mathcal{T}_{\theta_t}} (\nabla_u \mathcal{L}\left((u_{\theta_t})\right) + \varepsilon_t, \tag{76}$$

where ε_t is an error

$$\varepsilon_t = o(\eta_t \|G(\theta_t)^\dagger \nabla \ell(\theta_t)\|). \tag{77}$$

Proof. From the definition of w_t in (48), we make a change of variables :

$$w_t \in \arg\min_{w \in \mathbb{R}^P} \frac{1}{2} \left\|du_{\theta_t}(w) + \nabla_u \mathcal{L}(u_{\theta_t})\right\|^2_{L^2(\Omega)} \iff du_{\theta_t}(w_t) = \arg\min_{d \in \mathcal{T}_{\theta_t}} \frac{1}{2} \left\|d - (-\nabla_u \mathcal{L}(u_{\theta_t}))\right\|^2_{L^2(\Omega)}. \tag{78}$$

Thus

$$du_{\theta_t}(w_t) = -\Pi^{\perp}_{\mathcal{T}_{\theta_t}} (\nabla_u \mathcal{L}(u_{\theta_t})). \tag{79}$$

From the Taylor's expansion in (47), and the expression of w_t in (50) we conclude the theorem. □

We can also write the natural update in the parameter space through a formula using the tangent space instead of the Gramian.

Theorem 4 (Another form of the natural update in parameter space)**.** *Let us suppose that θ_{t+1} is defined with the natural gradient update in* (72)*. We can rewrite it:*

$$\theta_{t+1} = \theta_t - \eta_t du^\dagger_{\theta_t} \left(\Pi^{\perp}_{\mathcal{T}_{\theta_t}} (\nabla_u \mathcal{L}(u_{\theta_t}))\right). \tag{80}$$

Proof. Starting from (79), by taking the pseudoinverse of du_{θ_t}, we have:

$$w_t = -du^\dagger_{\theta_t} \left(\Pi^{\perp}_{\mathcal{T}_{\theta_t}} (\nabla_u \mathcal{L}(u_{\theta_t}))\right), \tag{81}$$

which concludes the theorem. □

Remark 4 (Interpretation of theorem 4). An interpretation of the natural gradient descent can be directly understood from the theorem 4 and the update (80). We take the true ideal gradient of the loss $\nabla_u \mathcal{L}(u_{\theta_t})$ and we project it onto the tangent space of the parametric model at θ_t, because the projection is the closest point from the true gradient to the tangent space. This represents the descent direction in the parametric model space that is the closest possible to the steepest descent in the function space with the L^2 metric. Then we transform back onto the parameter space through the pseudoinverse $du^\dagger_{\theta_t}$. This defines the natural gradient descent, which allows us to perform gradient descent in the ideal function space restricted to the manifold defined by the parametric model.

B.2 ANaGRAM: an empirical gradient descent optimizer

B.2.1 Introduction

As it is shown in the previous section, natural gradient descent allows us to take into consideration the geometry of the parametric model, while classical gradient descent is a special case when the derivative of the parametric model with respect to the parameter are orthonormals. However, the computational cost of the natural gradient descent is expensive because we need to calculate the pseudoinverse of the Gram matrix $G(\theta)$ of size $P \times P$ defined in 2, which induces a time complexity of $O(P^3)$. In neural networks, P can be quite big, and thus it makes the method intractable. Another problem is that in general, we cannot calculate exactly the Gramian matrix, because we need to calculate scalar product in $L^2(\Omega)$.

Schwencke and Furtlehner 2024 show in theorem 5 that under some hypothesis on the dataset, and by supposing that the model can be decomposed as $u_\theta = L_{\theta_1} \circ C_{\theta_2}$ where $\theta = (\theta_1, \theta_2)$ and L_θ is linear in θ_1, then we can construct an algorithm completely equivalent to the natural gradient descent as presented in (72) by calculating the pseudoinverse of a matrix of shape $N \times P$ instead, where N is the number of training samples. In the case where we cannot decompose the model with a linear operator as described above, experimental results show that ANaGRAM outperforms other optimizers like Adam, LBFGS, E-NGD (Müller and Zeinhofer 2023) when using Physics-Informed Neural network on most of the differential equations.

In the previous section, we used a generic loss $\mathcal{L}(u_\theta)$, to define the natural gradient descent. In this section, to be more concrete we use the L^2 loss:

$$\mathcal{L}(u) = \frac{1}{2}\|u - f\|^2_{L^2}, \tag{82}$$

$$\ell(\theta) := \mathcal{L}(u_\theta). \tag{83}$$

We could generalize the results to other losses, but the goal will be to apply it to PINN, and in this context L^2 loss is standard.

B.2.2 Natural Neural Tangent Kernel

We recall the natural gradient descent update in its functional form stated in the theorem 4

$$\theta_{t+1} \leftarrow \theta_t - \eta_t du^\dagger_{|\theta_t} \left(\Pi^\perp_{T_\theta}(\nabla_u \mathcal{L}(u_{\theta_t})\right). \tag{84}$$

One of the contribution of ANaGRAM is to relate the projection onto the tangent space $\Pi^\perp_{T_\theta}$ to the Natural Neural Tangent Kernel (NNTK). They prove a characterization of this projection with the following proposition:

Proposition 3 (Corollary 2 in Schwencke and Furtlehner 2024)**.** *We define the Natural Neural Tangent Kernel (NNTK), for all $\theta \in \mathbb{R}^P$*

$$NNTK_\theta(x, y) := \sum_{p=1}^{P}\sum_{q=1}^{P}(\partial_p u_{|\theta}(x))G(\theta)^\dagger_{pq}(\partial_q u_{|\theta}(y))^T \quad \textit{for all } x, y \in \Omega, \tag{85}$$

where $G(\theta)$ is the Gram matrix introduced in proposition 2. The $NNTK_\theta$ is the kernel of the projection $\Pi^\perp_{\mathcal{T}_\theta} : L^2(\Omega) \to \mathcal{T}_\theta$ onto $\mathcal{T}_\theta$. In other words:

$$\Pi^\perp_{\mathcal{T}_\theta}(f)(x) = \langle NNTK_\theta(x, \cdot), f\rangle_{L^2(\Omega)}. \tag{86}$$

First, in the proposition 3 we define the NNTK, which is the analogous of the Neural Tangent Kernel (NTK) in the case of natural gradient descent.

B.2.3 Empirical Natural Gradient Descent

The problem in practice is that we have a limited number of samples and thus cannot calculate a scalar product in L^2, except for some parametric model. We thus introduce some training points

$x_i \in \Omega$, and the corresponding targets $f(x_i)$ for all $1 \leq i \leq S$. We can write the corresponding approximation of the L^2 loss, which we call the empirical quadratic loss:

$$\hat{\ell}(\theta) := \frac{1}{2S} \sum_{i=1}^{S} \left(u_{|\theta}(x_i) - f(x_i)\right)^2. \tag{87}$$

We can show that the parametric model dynamic across time can be written in term of an empirical tangent space:

$$\hat{T}^{NNTK}_{\theta,x_i} := Span\left(NNTK_\theta(\cdot, x_i) : (x_i)_{i=1}^S\right). \tag{88}$$

In particular, considering the natural gradient update in (84), we can define the empirical natural gradient descent which instead uses the empirical tangent space that is defined by the dataset we have.

Definition 5. We define the empirical natural gradient descent with the following update in the parameter space:

$$\theta_{t+1} \leftarrow \theta_t - \eta_t du^\dagger_{|\theta_t}\left(\Pi^\perp_{\hat{T}^{NNTK}_{\theta,x_i}}(\nabla_u \mathcal{L}(u_{\theta_t})\right), \tag{89}$$

where the empirical tangent space is defined in 88.

Theorem 5 (ANaGRAM, Theorem 1 in Schwencke and Furtlehner 2024)**.** *We define the matrix $\hat{\phi}_\theta \in \mathbb{R}^{S\times P}$ such that for all $1 \leq i \leq S$ and $1 \leq p \leq P$:*

$$\hat{\phi}_{\theta_{i,p}} := \partial_p u_{|\theta}(x_i). \tag{90}$$

The empirical gradient of the loss evaluated in the dataset:

$$\widehat{\nabla\mathcal{L}}_{|u_{|\theta}} := \nabla L_{|u_{|\theta}}(x_i) = u_{|\theta}(x_i) - f(x_i), \tag{91}$$

then we can write the dynamic of the empirical natural gradient descent:

$$du^\dagger_{|\theta_t}\left(\Pi^\perp_{\hat{T}^{NNTK}_{\theta,x_i}}(\nabla_u \mathcal{L}(u_{\theta_t})\right) = \left(\hat{\phi}^\dagger_\theta + E^{metric}_\theta\right)\left(\widehat{\nabla L}_{|u_{|\theta}} + E^\perp_\theta\right), \tag{92}$$

where E^{metric}_θ and $E^\perp_\theta$ are correction terms defined in the theorem 1 in Schwencke and Furtlehner 2024.

Remark 5 (Discussion on the theorem 5). The theorem gives us an expression of the empirical natural gradient direction that we can calculate as long as we have a dataset. We recall that in the case when the parametric model is a neural network, we can compute the derivative of u_θ with respect to the parameter through auto-differentiation. The theorem makes appear E^{metric}_θ and $E^\perp_\theta$ which can be calculated through $G(\theta)^\perp$. However, in some cases, they can be ignored. To be more concrete, E^{metric}_θ will be determined by the way we choose the dataset $(x_i)_{i=1}^S$ in such way that the empirical tangent space $\hat{T}^{NNTK}_{\theta,x_i}$ approximates well the true tangent space T_θ. Proposition 1 in Schwencke and Furtlehner 2024 shows that there exists P points (x_i) such that $\hat{T}^{NNTK}_{\theta,x_i} = T_\theta$, and this case $E^{metric}_\theta = 0$. For $E^\perp_\theta$, they show that this term is equal to 0 in a very particular case when $f = 0$ and u has a last linear layer. They show that in practice, those terms can be ignored and the empirical natural gradient update becomes:

$$\theta_{t+1} = \theta_t - \hat{\phi}^\dagger_{\theta_t} \widehat{\nabla\mathcal{L}}_{|u_{|\theta_t}}. \tag{93}$$

The gradient $\widehat{\nabla\mathcal{L}}_{|u_{|\theta_t}}$ can be easily calculated by evaluating $u_{|\theta} - f$ at the sampled points. The difference with the previous update (52) is that now we need to invert a matrix of shape $P \times N$, instead of $P \times P$, which costs $O(PS\min(P, S))$. We calculate the pseudoinverse using a SVD (see E), with an appropriate chosen cutoff as we show in the next algorithm. Thus, in practice we ignore the correction terms E^{metric}_θ and $E^\perp_\theta$, which yields the ANaGRAM algorithm.

Remark 6 (Discussion on algorithm 1). As we discussed in the remark 5, the ANaGRAM algorithm used the closed form of the empirical natural gradient update given by theorem 5 by neglecting the correcting terms E^{metric}_θ and $E^\perp_\theta$. The parametric model u considered in the algorithm can be anything: it can be a Fourier model, in this case we have a simple closed-form expression to calculate $\hat{\phi}_{\theta_t}$. More commonly it can be a neural network, in this case the matrix $\hat{\phi}_{\theta_t}$ can be calculated through

Algorithm 1: ANaGRAM for regression (same as in Schwencke and Furtlehner 2024)

Data: $u : \mathbb{R}^P \to L^2(\Omega, \mu)$, a parametric model
$\theta_0 \in \mathbb{R}^P$, initialization of the parametric model
$f \in L^2(\Omega)$, target function of the quadratic regression
$(x_i) \in \Omega^S$, a batch in Ω
$\epsilon > 0$, cutoff level to compute the pseudo inverse

repeat

$\hat{\phi}_{\theta_t} \leftarrow \left(\partial_p u_{|\theta_t}(x_i)\right)_{1\le i\le S,\ 1\le p\le P}$, an $S \times P$ matrix computed *via* auto-differentiation;
$\hat{U}_{\theta_t}, \hat{\Delta}_{\theta_t}, \hat{V}^T_{\theta_t} \leftarrow \text{SVD}(\hat{\phi}_{\theta_t})$;
$\hat{\Delta}_{\theta_t} \leftarrow \left(\hat{\Delta}_{\theta_t,p} \text{ if } \hat{\Delta}_{\theta_t,p} > \epsilon \text{ else } 0\right)_{1\le p\le P}$;
$\widehat{\nabla\mathcal{L}} \leftarrow \left(u_{|\theta_t}(x_i) - f(x_i)\right)_{1\le i\le S}$;
$d_{\theta_t} \leftarrow \hat{V}_{\theta_t} \hat{\Delta}^{\dagger}_{\theta_t} \hat{U}^{\top}_{\theta_t} \widehat{\nabla\mathcal{L}}$;
$\eta_t \leftarrow \arg\min_{\eta\in\mathbb{R}^+} \sum_{i=1}^{S} \left(f(x_i) - u_{|\theta_t - \eta d_{\theta_t}}(x_i)\right)^2$, using e.g. line search;
$\theta_{t+1} \leftarrow \theta_t - \eta_t d_{\theta_t}$;

until *stop criterion is met*;

auto-differentiation, which is implemented in libraries like Pytorch, Jax, Tensorflow. In general, the condition to choose the parametric model is to have enough expressive power, and to be able to calculate easily its derivatives with respect to the parameters. The cutoff ε to invert the matrix $\hat{\phi}_{\theta_t}$ is another important part algorithm, and gives a threshold for which singular values below ε are treated as 0 before inverting them. This takes into account the numerical errors that happen in practice when doing calculations on a machine. Taking a cutoff too small will take into account too much noises, while a cutoff too high will ignore potential important parts of the signal. In general a value between 10^{-1} and 10^{-6} works well for most cases. Finally, the line search allows us to find the best learning rate to minimize the loss using such an empirical natural gradient update. In pratice, we simply generate a list of candidates values for the learning rate η_t, and keep the one that minimizes the loss the most.

B.3 Applying ANaGRAM to PINN

In the previous section, ANaGRAM is presented in the context of L^2 regression, i.e. when we want to approximate a function $f \in L^2$ through the loss $\mathcal{L}(u_\theta) = \|u_\theta - f\|^2_{L^2}$. Now, we generalize this to the context of PINN, to solve differential equations. The only difference is that we compose the parametric model with differential operators. Thus, this section is just a rewriting of previous theory in the case of PINN.

Here we consider the setting with a boundary equation:

$$\begin{cases} D[u] = f & \text{in } \Omega \\ B[u] = g & \text{on } \partial\Omega, \end{cases} \tag{94}$$

In the main paper, we consider a special case where we don't have a boundary equation to simplify the notation. The equation 94 is a more general case, in particular to "remove" the boundary equation, we can take define $B[u] := 0$ and $g = 0$.

We can rewrite 94 in the form of a regression problem

$$(D, B) \circ u = (f, g), \tag{95}$$

where $(D, B) \circ u$ can be seen as a parametric model, defined by the composition of the differential operators with the parametric model u defined in 1:

$$(D, B) \circ u : \begin{cases} \mathbb{R}^P & \longrightarrow \mathcal{H} & \longrightarrow L^2(\Omega, \partial\Omega) := L^2(\Omega \to \mathbb{R}) \times L^2(\partial\Omega \to \mathbb{R}) \\ \theta & \longmapsto u_{|\theta} & \longmapsto (Du_{|\theta}, Bu_{|\theta}) \end{cases} \tag{96}$$

In particular the natural gradient descent can be written:

$$\theta_{t+1} = \theta_t - \eta_t (d(D,B) \circ u_{\theta_t})^\dagger \left(\Pi^{\perp}_{\mathcal{T}^{(D,B)}_{\theta_t}} (\nabla_u \mathcal{L}(u_{\theta_t})) \right). \tag{97}$$

As a consequence, we can rewrite the algorithm 1 in the context of PINN in the following way.

Algorithm 2: ANaGRAM for PINN (same as in Schwencke and Furtlehner 2024

Data: $u : \mathbb{R}^P \to L^2(\Omega, \mu)$, a parametric model, i.e. a neural network
$\theta_0 \in \mathbb{R}^P$, initialization of the parametric model
$f \in L^2(\Omega)$, target function of the quadratic regression
$D : \mathcal{H} \longrightarrow L^2(\Omega \to \mathbb{R})$, a differential operator.
$B : \mathcal{H} \longrightarrow L^2(\partial\Omega \to \mathbb{R})$, a boundary operator.
$f \in L^2(\Omega \to \mathbb{R})$, a source.
$g \in L^2(\partial\Omega \to \mathbb{R})$, A boundary value.
$(x_i^D) \in \Omega^{S_D}$, a batch in Ω
$(x_i^B) \in \Omega^{S_B}$, a batch in $\partial\Omega$
$\epsilon > 0$, cutoff level to compute the pseudo inverse

while *criterion is not met* **do**

$\hat{\phi}_{\theta_t} \leftarrow \left(\left(\partial_p D[u_{|\theta_t}](x_i)\right)_{i=1}^{S_D}, \left(\partial_p B[u_{|\theta_t}](x_i)\right)_{i=1}^{S_B} \right)_{1 \le p \le P}$, a $(S^D + S^B) \times P$ matrix computed *via* auto-differentiation;

$\hat{U}_{\theta_t}, \hat{\Delta}_{\theta_t}, \hat{V}^T_{\theta_t} \leftarrow \text{SVD}(\hat{\phi}_{\theta_t})$;

$\hat{\Delta}_{\theta_t} \leftarrow \left(\hat{\Delta}_{\theta_t,p} \text{ if } \hat{\Delta}_{\theta_t,p} > \epsilon \text{ else } 0 \right)_{1 \le p \le P}$;

$\widehat{\nabla \mathcal{L}} \leftarrow \begin{pmatrix} \left(D[u_{|\theta_t}](x_i^D) - f(x_i^D)\right)_{1 \le i \le S_D} \\ \left(B[u_{|\theta_t}](x_i^D) - g(x_i^B)\right)_{1 \le i \le S_B} \end{pmatrix}$;

$d_{\theta_t} \leftarrow \hat{V}_{\theta_t} \hat{\Delta}^\dagger_{\theta_t} \hat{U}^\top_{\theta_t} \widehat{\nabla \mathcal{L}}$;

$\eta_t \leftarrow \arg\min_{\eta \in \mathbb{R}^+} \sum_{i=1}^{S^D} \left(f(x_i^D) - D[u|_{\theta_t - \eta d_{\theta_t}}](x_i^D) \right)^2 + \sum_{i=1}^{S^B} \left(g(x_i^B) - B[u|_{\theta_t - \eta d_{\theta_t}}](x_i^B) \right)^2$, using e.g. line search;

$\theta_{t+1} \leftarrow \theta_t - \eta_t d_{\theta_t}$;

end

C Algorithms

C.1 Classical curriculum learning algorithm

Algorithm 3: Curriculum learning (Krishnapriyan et al. 2021)

Data: $\boldsymbol{\alpha_0} \in \mathbb{R}$, a starting parameter
$\boldsymbol{\alpha_1} \in \mathbb{R}$, a final parameter
$\alpha : [0,1] \mapsto \mathbb{R}$ such that $\alpha(0) = \boldsymbol{\alpha_0}$, and $\alpha_1 = \boldsymbol{\alpha_1}$.
$0 = t_0 < t_1 < \cdots < t_N < t_{N+1} = 1$, a discretization of time in $[0,1]$

$\theta_{init} \in \mathbb{R}^P$, randomly initialized weights for the neural network;
Get θ_0 by training PINN for the differential equation with parameter $\boldsymbol{\alpha_0}$;
$k \leftarrow 1$;
while $k \le N+1$ **do**
  Get $\theta(k)$ by training PINN for the differential equation with parameter $\alpha(k)$, by starting with weights θ_{k-1};
  $k \leftarrow k+1$;
end
return θ_{N+1}, *the neural network parameter to approximate solution for* $\boldsymbol{\alpha_1}$.

C.2 Implicit function theorem method

We designed a new curriculum learning based method to solve parameterized differential equations. This algorithm is justified by theorem 1 which gives the implicit update to perform in the parameter space when we do a small variation on the parameter α:

$$\theta_{k+1} = \theta_k + \eta_k \dot{\theta}_k = \theta_k - \eta_k d_\theta \left(D[\alpha(k), d_{\theta(k)} u(\cdot)] \right)^\dagger_{|\theta(k)} \left(\Pi^\perp_{\mathcal{T}^{(D)}_{\theta(k)}} \left(d_\alpha D[\dot{\alpha}(k), u(k)]_{|\alpha(k)} \right) \right). \tag{98}$$

We notice the connection between 98 and the natural gradient descent 97. The update is exactly the same, except that we project a different function. It means that we can define implicit updates by doing natural gradient descent. As we have seen in Appendix B, natural gradient descent may be intractable when the model's number of parameters P is high. However, as described in the same Appendix, Schwencke and Furtlehner 2024 proposed a empirical method to perform natural gradient descent in a tractable way. We can leverage the theorem 5 by replacing $\nabla \mathcal{L}$ (the function that we normally project in natural gradient descent) by

$$\mathcal{I} = d_\alpha D[\dot{\alpha}(k), u(k)]_{|\alpha(k)}. \tag{99}$$

We can define its empirical version $\hat{\mathcal{I}} = (\mathcal{I}(x_i))_{i=1}^{S^{(D)}}$. With such change, the theorem stays valid, and thus we can apply ANaGRAM algorithm 2. This defines the following algorithm:

Algorithm 4: Implicit function theorem method (our contribution).

Data: $\boldsymbol{\alpha_0} \in \mathbb{R}$, a starting parameter
$\boldsymbol{\alpha_1} \in \mathbb{R}$, a final parameter
$\alpha : [0, 1] \mapsto \mathbb{R}$ such that $\alpha(0) = \boldsymbol{\alpha_0}$, and $\alpha_1 = \boldsymbol{\alpha_1}$.
$0 = t_0 < t_1 < \cdots < t_N < t_{N+1} = 1$, a discretization of time in $[0, 1]$
$\theta_{init} \in \mathbb{R}^P$, randomly initialized weights for the neural network;
Get θ_0 by training PINN for the differential equation with parameter $\boldsymbol{\alpha_0}$;
$k \leftarrow 1$;
while $k \leq N + 1$ **do**
  Get $\theta(k)$ by doing one step of algorithm 2 by replacing $\widehat{\nabla \mathcal{L}}$ by $\hat{\mathcal{I}}$ with weights θ_{k-1} and parameter $\alpha(k)$;
  $k \leftarrow k + 1$;
end
return θ_{N+1}, *the neural network parameter to approximate solution for* $\boldsymbol{\alpha_1}$.

D Experimental setups

D.1 Benchmark criteria

To be able to measure the performance of our method, we define some metrics. We first define the L_2 *relative error*:

$$E^{rel} := \frac{\sqrt{\sum_{i=1}^S \left(u_{|\theta}(x_i) - u^*(x_i) \right)^2}}{\sqrt{\sum_{i=1}^S u(x_i)^2}}, \tag{100}$$

and the L_2 *absolute error*:

$$E^{abs} := \sqrt{\frac{1}{S} \sum_{i=1}^S \left(u_{|\theta}(x_i) - u^*(x_i) \right)^2}, \tag{101}$$

where $(x_i)_{i=1}^S$ corresponds to a dataset of size S, and u^* is the desired solution of the differential equation, and u_θ is the approximation of the solution obtained through any parametric method. However, in some cases, we do not have an analytical solution of the differential equation, and in this

case we cannot calculate the relative and absolute errors. For this reason, we define another metric which is the L_2 *evaluation loss*:

$$\frac{1}{2S'_D}\sum_{i=1}^{S'_D}\left(D[u_\theta](x_i^D)-f(x_i^D)\right)^2+\frac{1}{2S'_B}\sum_{i=1}^{S'_B}\left(B[u_\theta](x_i^B)-g(x_i^B)\right)^2. \tag{102}$$

where we consider equation (3) and a boundary condition $Bu = g$. We can easily generalize this loss (and the methods) to any differential equations when there are more than 2 equations.

We will apply different methods to solve several differential equations with a target parameter $\boldsymbol{\alpha_1}$. The basic method consists of training the PINN with any optimizer such as Adam, ANaGRAM, and L-BFGS for the differential equation parameterized by $\boldsymbol{\alpha_1}$. The classical curriculum learning method is performed using the algorithm 3 and the implicit function when starting at a given $\boldsymbol{\alpha_0}$, moving to $\boldsymbol{\alpha_1}$. In both cases, except if stated otherwise, the initial PINN for the parameter $\boldsymbol{\alpha_0}$ is trained using ANaGRAM algorithm with the same hyperparameter as "ANaGRAM method". Hyperparameters are given for each case. The differential equations are presented in Appendix A. For each differential equation, we repeat the experiments several times for random seeds, and calculate some statistics in consequences: the mean, minimum, maximum, standard deviation and the median of relative/absolute errors and evaluation loss.

D.2 Computational details

For the implementation of neural networking training, Adam, L-BFGS and ANaGRAM, we used PyTorch version 2.0.1+cu117 (Paszke et al. 2019). All experiments were run on a NVIDIA A40 GPU. We also note that, unless otherwise specified, a uniform dataset of the domain is used. ANaGRAM method needs the inversion of a matrix of size $S \times P$, where S is the number of sampled points. Thus, to have a tractable algorithm, we consider, unless otherwise specified, that we sample 1000 points in the uniform dataset for ANaGRAM. If the dataset has less than 1000 points, then we use the whole dataset.

D.3 Curriculum learning: choice for the parameter path $\alpha(t)$

The path $\alpha(t)$ can be chosen freely, as long as it satisfies the boundary conditions $\alpha(0) = \boldsymbol{\alpha_0}$ and $\alpha_1 = \boldsymbol{\alpha_1}$. A general form is $\alpha(t) = \boldsymbol{\alpha_0} + \phi(t)(\boldsymbol{\alpha_1} - \boldsymbol{\alpha_0})$, where $\phi : [0, 1] \to [0, 1]$ is a differentiable function with $\phi(0) = 0$ and $\phi(1) = 1$. The choice of ϕ determines the scheduling of the parameter change. For a linear progression, we can choose $\phi(t) = t$, which results in a constant velocity $\dot{\alpha}(t) = \boldsymbol{\alpha_1} - \boldsymbol{\alpha_0}$. For a logarithmic schedule, which can be useful for certain problems, we can use:

$$\phi(t) = \frac{\log(1 + at)}{\log(1 + a)}, \tag{103}$$

where $a > 0$ is a hyperparameter controlling the curvature of the schedule.

D.4 Results and comparisons with other methods

D.4.1 Burgers equation

For Burgers' equation (see Appendix A.0.4), we consider the target parameter $\nu = 0.001$. In the literature, researchers tend to consider $\nu = 0.01/\pi \approx 0.0032$ which makes the solution easier to approximate. For computational proposes, we will consider only one seed. The neural network will has 3 layers of size 32, with $tanh$ activation function, except at the last layer, for a total of 2241 parameters. For ANaGRAM, we consider a cutoff of 0.001, 1500 epochs. For Adam we use a learning rate of 0.001 with 10000 epochs. For L-BFGS, we use Strong-Wolfe for the line search, maximum iteration of 500 and a learning rate of 1, we train the PINN with 50 epochs. For curriculum learning based methods, we start with $\boldsymbol{\nu_0} = 0.1$, apply 200 initial epochs (with ANaGRAM), and then perform 1200 updates. Because the solution of Burgers equation tends to be more steep around $x = 0$, we choose to sample more points around this point to have better performance. In time t, we consider a regular grid D_t of n points in [0,1], but at space x consider a non-regular grid D_x defined by:

- A regular grid of $\frac{n}{4}$ points on $[-1, -0.25)$.

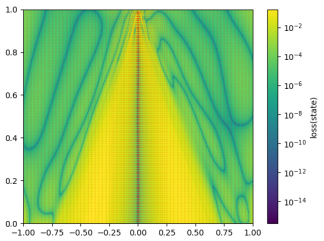
- A regular grid of $\frac{n}{4}$ points on $(0.25, 1]$.
- A grid defined as $-2^{D_{log}}$ on $[-0.25, 0)$, where D_{log} is a regular grid of $\frac{n}{4}$ points in $[2, 32]$.
- A grid defined as $2^{D_{log}}$ on $(0, 0.25]$, where D_{log} is a regular grid of $\frac{n}{4}$ points in $[2, 32]$.

We select $n = 100$, while we take 200 for the evaluation. Using such grids, we define the interior of the domain as $D_t \times D_x$. For the boundaries, we consider 102 points in $[0, 1] \times \{-1, +1\}$ and 51 points in $\{0\} \times [-1, 1]$. In the case of Adam, we use the whole dataset, while with L-BFGS and ANaGRAM we sample uniformly 1000 points in the discretization of the domain, because of memory issue.

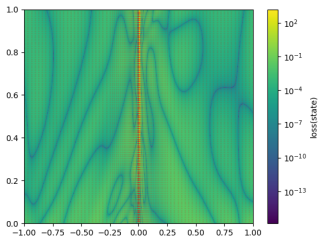
	Evaluation loss
Adam	1.32e-02
ANaGRAM	1.23e-01
L-BFGS	1.78e+00
Classical curriculum learning 3	1.40e-02
Implicit function method	**7.21e-03**

Table 2: Evaluation loss for Burgers equation with $\nu = 0.001$

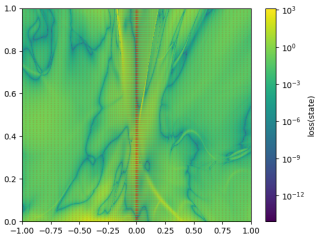
In the following picture, we show the loss at each point for the different methods.



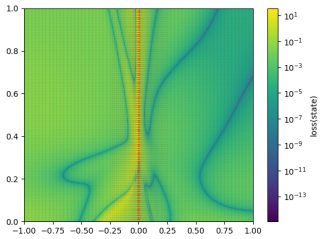
(a) Adam



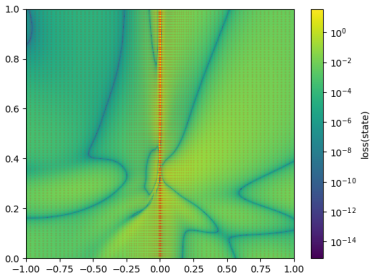
(b) L-BFGS



(c) ANaGRAM



(d) Classical curriculum learning



(e) Implicit function theorem

As we can see in the loss plots, the approximation is especially difficult around $x = 0$. Also we can see that the loss of Adam is very specific, and give a complete different solution (see Figure 7). The reference solution 3 shows that it is highly likely that Adam optimizer fell into a non optimal local minimizer.

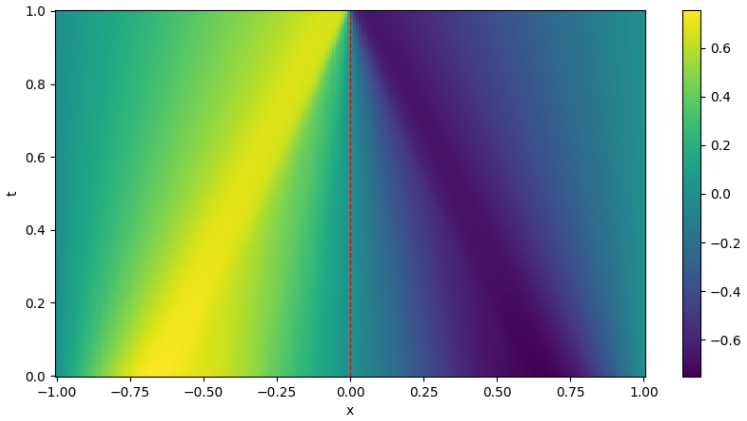


Figure 7: Approximation with Adam

D.4.2 Eikonal equation

We consider the eikonal equation 18 defined in Appendix A.0.5 with the target parameter $\varepsilon = 0$. The number of seeds is 5. We consider a neural network with 3 hidden layers of size 32 with in total 2209 parameters (weights and biases). Hyperbolic tangent activation function is used, except in the last layer. For ANaGRAM, a cutoff of 0.001 were chosen. For L-BFGS, we use Strong-Wolfe for the line search, maximum iteration of 200 and a learning rate of 1, we train the PINN with 50 epochs. For curriculum learning like implicit function theorem which is implemented using ANaGRAM, we also use a cutoff of 0.001. For curriculum learning based methods, we start with $\varepsilon = 1$, and apply 10 initial epochs to train the PINN. Then, we choose 200 values for ε_t in a logarithm space in $[0, 1]$. For the datasets, we take uniformly 50 points in $\Omega = (-1, 1)$ and 100 points in $\partial\Omega = \{-1, 1\}$ (50 for each). For evaluation dataset, 200 points for Ω and 400 points for $\partial\Omega$ were used.

	Mean	min	max	std	median
Adam	8.02e-01	**1.18e-03**	2.00e+00	9.78e-01	**4.88e-03**
ANaGRAM	9.27e-01	4.17e-01	1.59e+00	3.93e-01	8.08e-01
L-BFGS	1.20e+00	9.95e-04	2.00e+00	9.66e-01	1.98e+00
Classical curriculum learning, algorithm 3	**1.67e-02**	5.48e-03	**3.06e-02**	**8.15e-03**	1.64e-02
Implicit function method, algorithm 4	2.05e-02	1.24e-02	4.20e-02	1.09e-02	1.63e-02

Table 3: Relative errors for eikonal equation

	Mean	min	max	std	median
Adam	4.63e-01	**6.83e-04**	1.15e+00	5.65e-01	**2.82e-03**
ANaGRAM	5.35e-01	2.41e-01	9.20e-01	2.27e-01	4.66e-01
L-BFGS	6.94e-01	5.74e-04	1.16e+00	5.58e-01	1.14e+00
Classical curriculum learning, algorithm 3	**9.64e-03**	3.16e-03	**1.76e-02**	**4.71e-03**	9.49e-03
Implicit function method, algorithm 4	1.18e-02	7.14e-03	2.42e-02	6.31e-03	9.42e-03

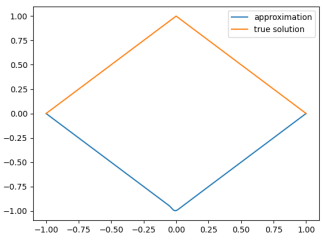
Table 4: Absolute errors for eikonal equation

The experiments on eikonal with the different methods show several behaviors. The first behavior often happens with methods like ANaGRAM where the solution obtained is not valid, even when the

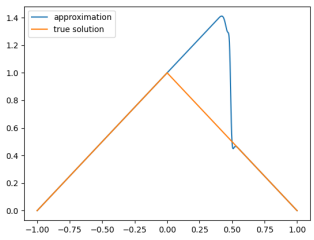
	Mean	min	max	std	median
Adam	2.89e-03	2.46e-03	3.27e-03	3.22e-04	2.76e-03
ANaGRAM	1.32e-01	8.58e-02	1.75e-01	2.84e-02	1.34e-01
L-BFGS	9.00e-01	2.78e-03	3.07e+00	1.16e+00	3.27e-01
Classical curriculum learning, algorithm 3	**2.77e-03**	2.45e-03	**3.12e-03**	**2.38e-04**	2.78e-03
Implicit function method, algorithm 4	8.62e-03	**2.28e-03**	3.01e-02	1.08e-02	**2.66e-03**

Table 5: Evaluation loss for eikonal equation

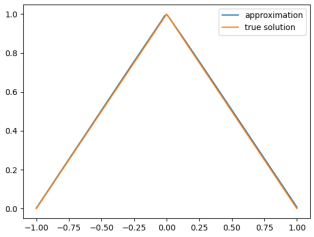
loss is very low (Figure 8b). Another behavior happens sometimes with Adam, where the model is a valid solution, but not the viscosity solution (Figure 8a): this explains why sometimes the relative and absolute error are high for Adam even though the evaluation loss is low. We can see that using curriculum learning based methods, we can guide the approximation towards the viscosity solution by starting with a simple differential equation (Figure 8c).



(a) Adam



(b) ANaGRAM



(c) Implicit function theorem

D.5 Hamilton-Jacobi-Bellman equation

As we seen in the Appendix A.0.6, there is an infinite number of Hamilton-Jacobi-Bellman, that are problem-dependent. In the following experiment, we will consider the one defined in example 1. We perform the experiments on 5 randomly generated seeds. The neural network has 2 layers of size 32 with $\tanh$ activation function, except at the last layer, for a total of 1153 parameters. For ANaGRAM we consider a cutoff of 0.001, and train for 500 epochs. For Adam, we use a learning rate of 0.001 and train it for 10000 epochs. For L-BFGS, we use Strong-Wolfe as line search, a maximum iteration of 500, a learning rate of 1, and train it with 50 epochs. For curriculum learning based methods, we start with a viscosity parameter $\varepsilon_0 = 0.1$ and finish with $\varepsilon_1 = 0$, doing 5000 initial epochs with Adam optimizer. Then, we perform 500 curriculum learning updates. The dataset is uniform in the domain. For the training, we sample uniformly 100 points in the interior and 200 points on the boundary. For the evaluation we sample 200 points in the interior and 400 points in the boundary.

In this example, we see that even a simple case of the HJB equation is difficult to approximate. For non-curriculum learning methods, it completely fails. The most satisfying would be the solution obtained using L-BFGS, however the boundary condition at 0 is not fulfilled. For the curriculum learning method, it is better but still not satisfactory. This shows that PINNs might need a lot of training to perform well. In particular, if we increase the number of updates in the curriculum learning

	Mean	min	max	std	median
Adam	5.56e-01	3.11e-01	7.22e-01	1.99e-01	7.14e-01
ANaGRAM	1.38e+00	4.80e-01	2.51e+00	6.62e-01	1.36e+00
L-BFGS	7.09e-01	6.87e-01	7.34e-01	**1.74e-02**	7.08e-01
Classical curriculum learning, algorithm 3	3.64e-01	2.04e-01	**4.70e-01**	8.98e-02	**3.81e-01**
Implicit function method, algorithm 4	**3.44e-01**	**9.20e-02**	5.90e-01	2.05e-01	4.14e-01

Table 6: Relative errors

	Mean	min	max	std	median
Adam	9.45e-01	5.29e-01	1.23e+00	3.38e-01	1.21e+00
ANaGRAM	2.34e+00	8.17e-01	4.26e+00	1.12e+00	2.32e+00
L-BFGS	1.20e+00	1.17e+00	1.25e+00	**2.96e-02**	1.20e+00
Classical curriculum learning, algorithm 3	6.18e-01	3.47e-01	**7.99e-01**	1.53e-01	**6.48e-01**
Implicit function method, algorithm 4	**5.84e-01**	**1.56e-01**	1.00e+00	3.49e-01	7.04e-01

Table 7: Absolute errors

	Mean	min	max	std	median
Adam	1.22e-01	**1.56e-02**	2.83e-01	1.31e-01	**1.56e-02**
ANaGRAM	8.98e-02	2.22e-02	2.32e-01	8.32e-02	3.38e-02
L-BFGS	**1.56e-02**	**1.56e-02**	**1.56e-02**	**4.89e-07**	**1.56e-02**
Classical curriculum learning, algorithm 3	3.65e-02	1.86e-02	6.82e-02	1.72e-02	3.07e-02
Implicit function method, algorithm 4	4.73e-02	1.90e-02	7.50e-02	2.18e-02	4.12e-02

Table 8: Evaluation loss

method we can obtain a better approximation. By doing 100 epochs every update, i.e. 100 epochs for each new value of parameter ε_t, we can obtain the Figure 10.

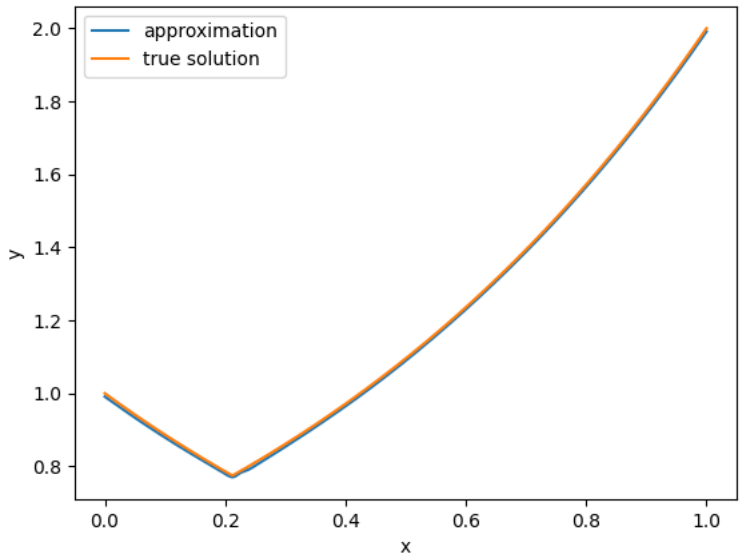
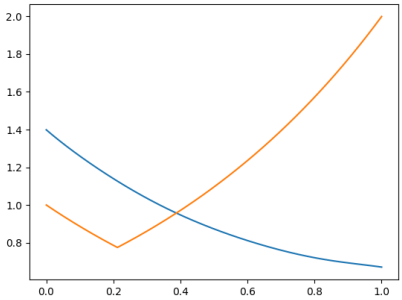
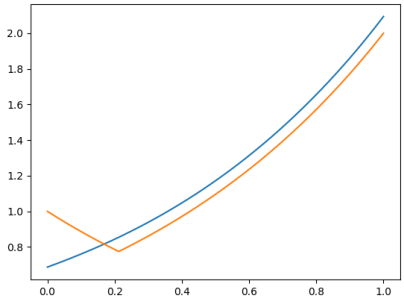


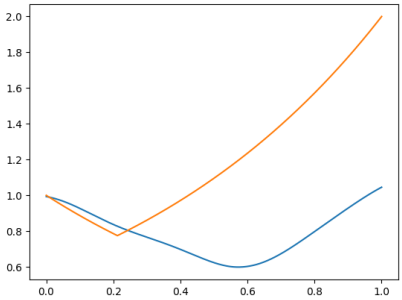
Figure 10: Implicit function theorem method with additional epochs between each update.



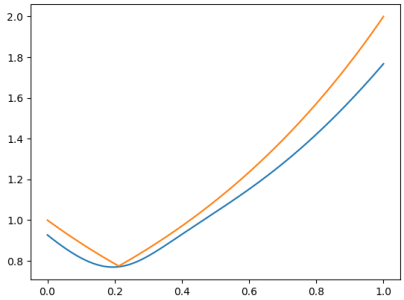
(a) Adam



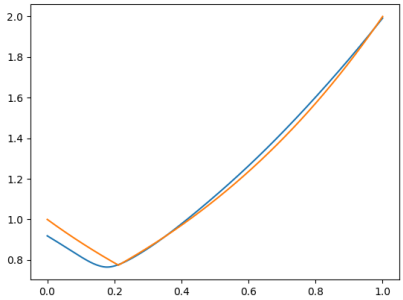
(b) L-BFGS



(c) ANaGRAM



(d) Classical curriculum learning



(e) Implicit function theorem

Figure 9: Approximations with different methods. The yellow curve is the true solution and the blue curve is the approximation.

Meanwhile, increasing the number of epochs for non-curriculum learning methods does not work. For instance, we tried to train the PINN with Adam optimizer using 10 times more epochs than the previously, going up to 100000 epochs. In this case we obtain the Figure 11. We observe that the model is very instable, exactly as we observed with ANaGRAM optimizer in the eikonal experiments. The loss is very low because the differential equation is respected almost everywhere, even though the approximation is pretty far from the true solution.

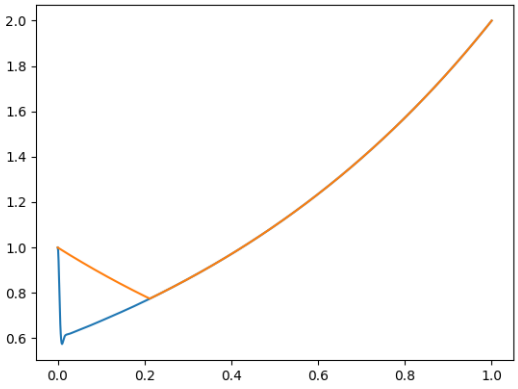
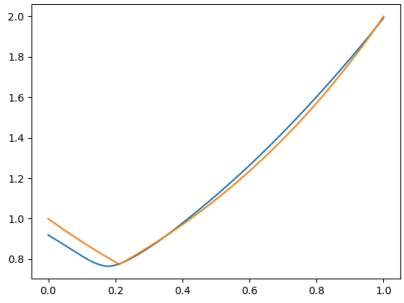
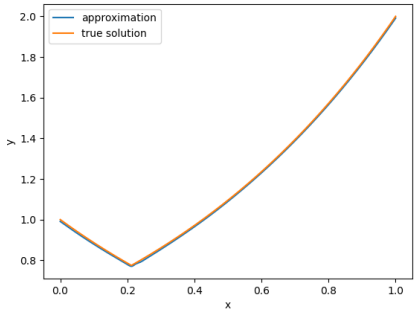


Figure 11: Adam method with 100000 epochs.



(a) Direct application of the implicit function theorem.



(b) Between each implicit update, we perform several Adam iterations on the loss.

Figure 12: Applying optimization steps between implicit updates for the HJB equation. The yellow curve is the true solution and the blue curve is the approximation.

E Moore-Penrose pseudoinverse

In this section, we will define the Moore-Penrose inverse, also called pseudoinverse. It can be seen as a generalization of inverse for non-square matrices. It can also be generalized to continuous linear operator defined on Hilbert space under some assumptions. The goal of this appendix is to justify the use of pseudoinverse in our work.

E.1 Pseudo inverse of a matrix

For a rectangular matrix (potentially square), we can define Moore-Penrose inverse algebraically. But to obtain a simple form computable easily on computer, the simplest approach is to define it through its Singular Value Decomposition (SVD).

Definition 6 (Moore-Penrose inverse)**.** Let $A \in \mathbb{R}^{M\times N}$ and $r = rank(A) \leq min(M, N)$. We can always consider the SVD of such matrix, and write it $A = U\Delta V^T$, with $\Delta = diag(\sigma_i) \in \mathbb{R}^{r\times r}$, and $U \in \mathbb{R}^{M\times r}$, $V \in \mathbb{R}^{N\times r}$ are orthogonal matrices, i.e. $U^TU = V^TV = I_r$. We define the pseudoinverse as:

$$A^\dagger := V\Delta^\dagger U^T \in \mathbb{R}^{N\times M}, \tag{104}$$

where the pseudoinverse of a diagonal matrix is:

$$\Delta_i^\dagger = \begin{cases} 0 & \text{if } \sigma_i = 0, \\ \frac{1}{\sigma_i} & \text{otherwise.} \end{cases} \tag{105}$$

In particular, it is interesting to see that the pseudoinverse is equals to the classical inverse when the matrix A is square and invertible.

Proposition 4. *If A is a square matrix of size N that is invertible, then $A^\dagger = A^{-1}$.*

Proof. We can decompose A with the SVD $A = U\Delta V^T$ with U, V orthogonal matrices. Since A is invertible, then we have $A^{-1} = (V^T)^{-1}\Delta^{-1}U^{-1}$, in particular $\Delta^{-1} = \Delta^\dagger$ and because U, V are orthogonal, then, $A^{-1} = V\Delta^\dagger U^T = A^\dagger$. □

Proposition 5 (Properties of pseudoinverse)**.** *Let $A \in \mathbb{R}^{M\times N}$ a matrix. The pseudoinverse has the following properties:*

1. $AA^\dagger A = A$,
2. $A^\dagger AA^\dagger = A^\dagger$,
3. $(A^\dagger A)^T = A^\dagger A$,
4. $(AA^\dagger)^T = AA^\dagger$,
5. $A^\dagger A$ *is an orthogonal projection in* ImA^T,
6. $AA^\dagger$ *is an orthogonal projection in* ImA.

Proof. Let's consider the SVD of A: $A = U\Delta V^T$, where U and V are orthogonal matrices. Proof of 1: We have $AA^\dagger A = (U\Delta V^T)(V\Delta^\dagger U^T)(U\Delta V^T) = U(\Delta\Delta^\dagger\Delta)V^T$ because U and V are orthogonal matrices. We also have $\Delta\Delta^\dagger\Delta = \text{diag}(\sigma_i\sigma_i^\dagger\sigma_i) = \text{diag}(\sigma_i) = \Delta$. Thus $AA^\dagger A = U\Delta V^T = A$. The proof of 2, 3 and 4 can also be deduced in a very similar way using the SVD. Now we are going to prove 5: Let's write $Q = A^\dagger A$, first, let's check that Q is an orthogonal projection matrix: we have $Q^2 = A^\dagger AA^\dagger A = A^\dagger A = Q$ by property 2. Also $Q^T = (A^\dagger A)^T = A^\dagger A = Q$ by property 3. Now that we know that Q is an orthogonal projection, we want to show its image is $\text{Im}A^T$. By property 1, we have $A = AA^\dagger A = AQ$, thus $A^T = Q^T A^T = QA^T$. We deduce that $ImA^T \subset ImQ$. In converse, since $Q^T = Q$, then $ImQ = ImQ^T = ImA^T(A^\dagger)^T \subset ImA^T$. We prove 6 in a very similar way. □

Remark 7. In particular, the properties 1, 2, 3, and 4 are a way to define the pseudoinverse algebraically. As we shown in the proof, the properties 5 and 6 are consequences of those properties.

The next propositions justify the use of pseudoinverse to solve a matrix equation.

Proposition 6. *Let $A \in \mathbb{R}^{M\times N}$ a matrix. We have $AA^\dagger y = y$ if and only if $y \in \text{Im}\, A$.*

Proof. If $AA^\dagger y = y$, then $y \in \text{Im}\, A$ by definition of Im. Now if we suppose that $y \in \text{Im}\, A$, then since $AA^\dagger$ is a projection in $\text{Im}\, A$ by property 6 of proposition 5, then $AA^\dagger y = y$. □

When $y \notin \text{Im}\, A$, we can interpret the pseudoinverse as the minimal norm solution of the corresponding least squares problem.

Proposition 7 (Least squares)**.** *Let's consider the optimization problem:*

$$\arg\min_x \frac{1}{2}\|Ax - y\|_2^2, \tag{106}$$

then $x^\dagger := A^\dagger y$ is a solution of this problem and also it is the one with the minimal norm $\|.\|_2$.

Proof. The least squares problem is a convex continuous and differentiable problem, thus we can calculate the derivative of the objective problem and derive that the solutions should verify the so-called normal equation:

$$A^T Ax = A^T y. \tag{107}$$

We show that $x^\dagger$ verifies the normal equation, indeed by considering the SVD decomposition $A = U\Delta V^T$, we have:

$$\begin{aligned} A^T Ax^\dagger &= A^T AA^\dagger y && (108)\\ &= V\Delta U^T U\Delta V^T V\Delta^\dagger U^T y && (109)\\ &= V\Delta^2\Delta^\dagger U^T y && (110)\\ &= V\Delta U^T y && (111)\\ &= A^T y. && (112) \end{aligned}$$

Now let's show that $x^\dagger$ is indeed the solution of minimal norm. Let $\hat{x}$ another solution of the optimization problem, in particular it verifies the normal equation: $A^T A\hat{x} = A^T y$. By substracting with the normal equation on $x^\dagger$ we have:

$$A^T A(\hat{x} - x^\dagger) = 0. \tag{113}$$

Thus $\hat{x} - x^\dagger \in \ker A^T A = \ker A$. So it exists a $w \in \ker A$ such that:

$$\hat{x} = x^\dagger + w. \tag{114}$$

The key now is to show that $x^\dagger \in \operatorname{Im} A^T = (\ker A)^\perp$: because of the property 5 of proposition 5, we have $A^\dagger Ax^\dagger \in \operatorname{Im} A^T$. But also by property 2 we have $A^\dagger Ax^\dagger = A^\dagger AA^\dagger y = A^\dagger y = x^\dagger$. Thus $x^\dagger \in \operatorname{Im} A^T = (\ker A)^\perp$, thus, we can apply Pythagorean theorem:

$$\|\hat{x}\|_2^2 = \|x^\dagger\|_2^2 + \|w\|_2^2. \tag{115}$$

To have the minimal norm, we have to choose $w = 0$, and thus in this case $\hat{x} = x^\dagger$. □

E.2 Pseudoinverse of the differential for a parametric model

In this work, we often consider the differential in $\theta \in \mathbb{R}^P$ noted $du_{|\theta}$ associated to a model u_θ such that $du_\theta : \mathbb{R}^P \longrightarrow \mathcal{H}$, where $\mathcal{H}$ is a Hilbert space. We want to be able to consider the pseudoinverse $du^\dagger : \mathcal{H} \longrightarrow \mathbb{R}^P$. It is possible to define the pseudoinverse for a general bounded linear operator $A : \mathcal{H}_1 \longrightarrow \mathcal{H}_2$ with $\mathcal{H}_1$, $\mathcal{H}_2$ two general Hilbert spaces (bounded means that there exists $M > 0$ such that $\|Ax\|_{\mathcal{H}_2} \leq M\|x\|_{\mathcal{H}_1}$ for all $x \in \mathcal{H}_1$), and where the image $\operatorname{Im} A$ is closed. However, in the case that interests us (for du_θ), $\mathcal{H}_1 = \mathbb{R}^P$ is a vector space of finite dimension, which simplifies such an extension of the pseudoinverse. Since the parameter space $\mathbb{R}^P$ has finite dimension, the image of du_θ also has a finite dimension. In particular $\operatorname{Im} du = Span(\{du(e_1), \dots, du(e_P)\})$. Thus, du is a bounded linear operator with a closed image. Also, we can define the SVD of such operator, which allows us to define the pseudoinverse:

Proposition 8 (Pseudo inverse of du_θ)**.** *Let $\theta \in \mathbb{R}^P$. For $du_\theta : \mathbb{R}^P \longrightarrow \mathcal{H}$ as defined in 2. We can apply SVD on such operator, i.e. for a $r \leq P$ there exists $(v_i)_{i=1}^r$ orthonormal vectors in $\mathbb{R}^P$, and $(u_i)_{i=1}^r$ an orthonormal basis of $\operatorname{Im} du$, and $(\sigma_i)_{i=1}^r$ non negative numbers such that:*

$$du_\theta(h) = \sum_{i=1}^r \sigma_i \langle h, v_i\rangle u_i. \tag{116}$$

Thus, we define the pseudo inverse of $du_\theta^\dagger : T_\theta \longrightarrow \mathbb{R}^P$ as a bounded linear operator *such that for all $y \in T_\theta := \operatorname{Im}(du)$, we have:*

$$du_\theta^\dagger(y) = \sum_{i=1}^r \sigma_i^\dagger \langle y, u_i\rangle_{\mathcal{H}} v_i, \tag{117}$$

with

$$\sigma_i^\dagger = \begin{cases} 0 & \text{if } \sigma_i = 0, \\ \frac{1}{\sigma_i} & \text{otherwise.} \end{cases} \tag{118}$$

Proof. Let $(e_p)_{p=1}^P$ the canonical basis of $\mathbb{R}^P$. Let $g_i := du_\theta(e_i) \in \mathcal{H}$. In particular, we have $T_\theta = Span(g_i : 1 \leq i \leq P)$. Let $G \in \mathbb{R}^{P\times P}$ the Gram matrix such that $G_{ij} = \langle g_i, g_j\rangle_{\mathcal{H}}$. We can find diagonalize this symmetrical real matrix and in particular we can find orthonormal vectors $(v_i)_{i=1}^r$ such that $Gv_i = \lambda_i v_i$ and $\langle v_i, v_j\rangle = \delta_{ij}$ for all $i, j \in \{1, \dots, r\}$. Let's define $\sigma_i = \sqrt{\lambda_i}$. We claim that

$$u_i = \frac{1}{\sigma_i}\sum_{j=1}^P v_{ij} g_j, \tag{119}$$

is an orthonormal basis of T_θ. We can see that because for $i, n \in \{1, \dots, r\}$:

$$\begin{aligned}
\langle u_i, u_n\rangle_{\mathcal{H}} &= \frac{1}{\sigma_i\sigma_n}\sum_{j=1}^P\sum_{k=1}^P v_{ij}v_{nk}\langle g_j, g_k\rangle_{\mathcal{H}} && (120)\\
&= \frac{1}{\sigma_i\sigma_n}\sum_{j=1}^P\sum_{k=1}^P v_{ij}v_{nk}G_{jk} && (121)\\
&= \frac{1}{\sigma_i\sigma_n} v_n^T G v_i && (122)\\
&= \frac{\lambda_i}{\sigma_i\sigma_n} v_n^T v_i && (123)\\
&= \frac{\lambda_i}{\sigma_i\sigma_n}\delta_{i,n} && (124)\\
&= \delta_{in}. && (125)
\end{aligned}$$

Now, for $h \in \mathbb{R}^P$, we can write it as: $h = \sum_{k=1}^P h_k e_k$, and thus by linearity $du_\theta(h) = \sum_{k=1}^P h_k g_k$. Using the orthonormality of $(u_i)_{i=1}^r$, we have:

$$\begin{aligned}
du_\theta(h) &= \sum_{i=1}^r \langle du_\theta(h), u_i\rangle_{\mathcal{H}} u_i && (126)\\
&= \sum_{i=1}^r \left\langle \sum_{k=1}^P h_k g_k, \frac{1}{\sigma_i}\sum_{j=1}^P v_{ij} g_j \right\rangle_{\mathcal{H}} u_i && (127)\\
&= \sum_{i=1}^r\sum_{k=1}^P\sum_{j=1}^P \frac{1}{\sigma_i} h_k v_{ij} G_{kj} u_i && (128)\\
&= \sum_{i=1}^r \sigma_i \langle x, v_i\rangle u_i. && (129)
\end{aligned}$$

□

Proposition 9 (Adjoint of the differential)**.** *We recall that the adjoint of an operator is a generalization of the transpose for a matrix. Let* $T : \mathcal{H}_1 \longrightarrow \mathcal{H}_2$, a bounded linear operator *with* $\mathcal{H}_1$, $\mathcal{H}_2$ *two Hilbert spaces. Rudin 1991 (section 12.9 in second edition) show that there is a unique adjoint for* T *noted* T^* *such that:*

$$\langle Th, y\rangle = \langle h, T^* y\rangle \quad \textit{for all } h, y. \tag{130}$$

In particular, we can write an explicit formula for the adjoints associated to du_θ *and its pseudoinverse. For any* $h \in \mathbb{R}^P$ *and* $k \in \mathcal{H}$*.*

$$du_\theta^*(y) = \sum_{i=1}^r \sigma_i \langle y, u_i\rangle_{\mathcal{H}} v_i, \tag{131}$$

$$(du_\theta^\dagger)^*(h) = \sum_{i=1}^r \sigma_i^\dagger \langle h, v_i\rangle u_i. \tag{132}$$

Proof. Using proposition 8 with the same notations, we can write:

$$\langle du_\theta(h), y\rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^{r} \sigma_i \langle h, v_i\rangle u_i, y \right\rangle \tag{133}$$

$$= \left\langle h, \sum_{i=1}^{r} \sigma_i \langle u_i, y\rangle_{\mathcal{H}} v_i \right\rangle_{\mathcal{H}}. \tag{134}$$

We conclude that:

$$du_\theta(y)^* = \sum_{i=1}^{r} \sigma_i \langle u_i, y\rangle_{\mathcal{H}} v_i \tag{135}$$

In the same way, we have:

$$\left\langle du_\theta^\dagger(y), h \right\rangle = \left\langle \sum_{i=1}^{r} \sigma_i^\dagger \langle y, u_i\rangle_{\mathcal{H}} v_i, h \right\rangle \tag{136}$$

$$= \left\langle y, \sum_{i=1}^{r} \sigma_i^\dagger \langle v_i, h\rangle u_i \right\rangle_{\mathcal{H}}. \tag{137}$$

We conclude that:

$$(du^\dagger)^*(h) = \sum_{i=1}^{r} \sigma_i^\dagger \langle v_i, h\rangle u_i. \tag{138}$$

□

From this definition of the pseudoinverse, we can generalize the properties 5:

Proposition 10. *Let $\theta \in \mathbb{R}^P$. For any $h \in \mathbb{R}^P$ and $y \in \mathcal{H}$, we have the following properties:*

1. $du_\theta \circ du_\theta^\dagger \circ du_\theta(h) = du_\theta(h)$,
2. $du_\theta^\dagger \circ du_\theta \circ du_\theta^\dagger = du_\theta^\dagger$,
3. $(du_\theta^\dagger \circ du_\theta)^*(h) = du_\theta^\dagger \circ du_\theta(h)$,
4. $(du_\theta \circ du_\theta^\dagger)^*(y) = du_\theta \circ du_\theta^\dagger(y)$,
5. $du_\theta^\dagger \circ du_\theta$ *is an orthogonal projection in* $\operatorname{Im} du_\theta^*$,
6. $du_\theta \circ du_\theta^\dagger$ *is an orthogonal projection in* $\operatorname{Im} du_\theta$.

Proof. Proof of 1: from proposition 8, we have:

$$du_\theta \circ du_\theta^\dagger \circ du_\theta(h) = du_\theta \circ du_\theta^\dagger \left(\sum_{j=1}^{r} \sigma_j \langle h, v_j \rangle u_j \right) \tag{139}$$

$$= du_\theta \left(\sum_{i=1}^{r} \sigma_i^\dagger \left\langle \sum_{j=1}^{r} \sigma_j \langle h, v_j \rangle u_j, u_i \right\rangle_{\mathcal{H}} v_i \right) \tag{140}$$

$$= du_\theta \left(\sum_{i=1}^{r} \sum_{j=1}^{r} \sigma_i^\dagger \sigma_j \langle h, v_j \rangle \langle u_j, u_i \rangle v_i \right) \tag{141}$$

$$= du_\theta \left(\sum_{j=1, \sigma_j \neq 0}^{r} \langle h, v_j \rangle v_j \right) \tag{142}$$

$$= \sum_{i=1}^{r} \sigma_i \left\langle \sum_{j=1, \sigma_j \neq 0}^{r} \langle h, v_j \rangle v_j, v_i \right\rangle u_i \tag{143}$$

$$= \sum_{i=1}^{r} \sum_{j=1, \sigma_j \neq 0}^{r} \sigma_i \langle h, v_j \rangle \langle v_j, v_i \rangle u_i \tag{144}$$

$$= \sum_{i=1}^{r} \sigma_i \langle h, v_i \rangle u_i. \tag{145}$$

The proof of property 2 is very similar. For the proof of property 3, we can use the proposition 9, we have:

$$(du_\theta^\dagger \circ du_\theta)^*(h) = du_\theta^* \circ (du_\theta^\dagger)^*(h) \tag{146}$$

$$= du_\theta^* \left(\sum_{j=1}^{r} \sigma_j^\dagger \langle h, v_j \rangle u_j \right) \tag{147}$$

$$= \sum_{i=1}^{r} \sigma_i \left\langle \sum_{j=1}^{r} \sigma_j^\dagger \langle h, v_h \rangle u_j, u_i \right\rangle v_i \tag{148}$$

$$= \sum_{i=1}^{r} \sum_{j=1}^{r} \sigma_i \sigma_j^\dagger \langle h, v_j \rangle \langle u_j, u_i \rangle_{\mathcal{H}} v_i \tag{149}$$

$$= \sum_{i=1, \sigma_i \neq 0}^{r} \langle h, v_i \rangle v_i \tag{150}$$

$$= du_\theta^\dagger \circ du_\theta(h). \tag{151}$$

The proof for 4 is similar. For the proof of properties 5 and 6 we have seen in the proof of proposition 5 that this is a consequence of properties 1,2, 3 and 4. □

We can also generalize the proposition 7:

Proposition 11 (Least squares for the differential)**.** *Let $\theta \in \mathbb{R}^P$, Let's consider the optimization problem:*

$$\arg\min_{h \in \mathbb{R}^P} \frac{1}{2} \|du_\theta(h) - y\|_{L^2}^2, \tag{152}$$

then $h^\dagger := du_\theta^\dagger(y)$ is a solution of this problem and also it is the one with the minimal norm $\|.\|_2$.

Proof. The proof is identical to the one of proposition 7, except that we have to justify the fact that solutions of the least squares problem verifies the normal equation. As justified in proposition 9 the

linear operator du_θ admits a unique adjoint noted $du_\theta^* : \mathcal{H} \longrightarrow \mathbb{R}^P$ that verifies:

$$\langle du_\theta(h), k\rangle_{\mathcal{H}} = \langle h, du_\theta^*(k)\rangle_2, \quad \text{for } h \in \mathbb{R}^P, k \in \mathcal{H}. \tag{153}$$

From this, we deduce that for any $h, h' \in \mathbb{R}^P$:

$$\frac{1}{2}\|du_\theta(h+h') - y\|_{\mathcal{H}}^2 = \frac{1}{2}\|du_\theta(h) - y\|_{\mathcal{H}}^2 + \langle du_\theta(h'), du_\theta(h) - y\rangle_{\mathcal{H}} + \|du_\theta(h')\|_{\mathcal{H}}^2 \tag{154}$$

$$= \frac{1}{2}\|du_\theta(h) - y\|_{\mathcal{H}}^2 + \langle h', du_\theta^*(du_\theta(h) - y)\rangle_2 + o(\|h\|). \tag{155}$$

We deduce that the gradient of $\frac{1}{2}\|du_\theta(h) - y\|_{\mathcal{H}}^2$ is equal to $du_\theta^*(du_\theta(h) - y)$. In particular, for any solution h^* of the least squares problem, we have the normal equation:

$$du_\theta^*(du_\theta(h^*) - y) = 0. \tag{156}$$

We can verify the rest of the proof in a similar way than the proof of proposition 7. □

E.3 Pseudoinverse for the differential with respect to a function

Until now, we considered the differential du_θ with respect to θ a parameter in a finite dimension space. However, it can be useful to consider the differential with respect to a function. For example, $d_u D[\hat{u}]$ would be the differential of the operator D evaluated a function $\hat{u}$. The issue is that in such case, the operator is defined on two potentially infinite dimension Hilbert space, and thus we need some assumptions to obtain results as before. First, we need to suppose that $d_u D[\hat{u}]$ is a bounded operator to be able to define the pseudoinverse (Engl and Ramlau 2015). However, the pseudoinverse in this case is not necessarily bounded as shown in the following example:

Example 3 (Unbounded pseudoinverse)**.** Let $\ell^2(\mathbf{N})$ the sequence space, equipped with the scalar product $\langle w, z\rangle = \sum_{n=0}^{+\infty} w_n z_n$. Let us consider the operator

$$A : \begin{cases} \ell^2(\mathbf{N}) & \longrightarrow \ell^2(\mathbf{N}) \\ (w_0, w_1, \dots) & \longmapsto (\sigma_0 w_0, \sigma_1 w_1, \dots) \end{cases}, \tag{157}$$

where $\sigma_n := \frac{1}{n+1}$. A is a bounded operator: $\|Aw\| \leq \|w\|$ for all $w \in \ell^2(\mathbf{N})$. We can define the pseudoinverse in $\mathrm{Im}(A)$ as:

$$A^\dagger y = \left(\frac{1}{\sigma_1} y_1, \frac{1}{\sigma_2} y_2, \dots\right), \quad \text{where } y \in \mathrm{Im}(A). \tag{158}$$

However, if we consider the canonical basis of $\ell^2(\mathbf{N})$: $e^{(k)} = (0, 0, \dots, 1, 0, \dots)$, we have:

$$y^{(k)} := Ae^{(k)} = (0, 0, \dots, \sigma_k, 0, \dots). \tag{159}$$

Applying the pseudoinverse on $y^{(k)}$ yields back:

$$A^\dagger y^{(k)} = e^{(k)}. \tag{160}$$

However we have:

$$\frac{\|A^\dagger y^{(k)}\|}{\|y^{(k)}\|} = \frac{1}{\sigma_k} = k + 1 \xrightarrow[k \to +\infty]{} +\infty \tag{161}$$

An unbounded pseudoinverse implies that the pseudoinverse is discontinuous, which leads to instability in approximating the solution of the least-squares problem. It can be shown that the pseudoinverse is bounded if and only if the image of the operator $d_u D(\hat{u})$ is closed (Engl and Ramlau 2015).

F Technical details about the implicit function theorem

Theorem 6 (Implicit function theorem, functional space)**.** *Let $\alpha : [0, 1] \to \mathcal{A}$, be a function that verifies $\alpha(0) = \boldsymbol{\alpha_0}$, and $\alpha_1 = \boldsymbol{\alpha_1}$, with $\mathcal{A} \subset \mathbb{R}$. Let $u(t) : [0, 1] \longrightarrow \mathcal{H}$ a solution of the problem 3 with α. Additionnally, we suppose that the linear operator $d_u D$ is bounded and that its pseudoinverse $(d_u D)^\dagger$ is also bounded (see Appendix E.3). Supposing that $u(0) = \mathbf{u_0}$, we have:*

$$\dot{u}(t) = -d_u \left(D[\alpha(t), \cdot]\right)^\dagger_{|u(t)} \left(d_\alpha D(\cdot, u(t))_{|\alpha(t)}[\dot{\alpha}(t)]\right), \tag{162}$$

is the derivative of u in time, optimal in the least squares sense with the minimal norm.

Proof. Let $t \in [0,1]$, from the assumptions, we have $D[\alpha_t, u(t)] = f$. We can derivate it with respect to t:

$$\frac{d}{dt} D[\alpha(t), u(t)] = 0, \tag{163}$$

By using the chain rule, we have:

$$d_\alpha D[\cdot, u(t)]_{|\alpha(t)} \left(\dot{\alpha}(t)\right) + d_u D[\alpha(t), \cdot]_{|u(t)} \left(\dot{u}(t)\right) = 0, \tag{164}$$

thus,

$$d_u D[\alpha(t), \cdot]_{|u(t)} \left(\dot{u}(t)\right) = -d_\alpha D[\cdot, u(t)]_{|\alpha(t)} \left(\dot{\alpha}(t)\right). \tag{165}$$

We can consider the associated least squares problem:

$$\min_{w \in \mathcal{H}} \frac{1}{2} \left\| d_u D[\alpha(t), \cdot]_{|u(t)} (w) + d_\alpha D[\cdot, u(t)]_{|\alpha(t)} \left(\dot{\alpha}(t)\right) \right\|_{L^2}^2. \tag{166}$$

We know that the minimal norm solution of this problem can be defined using the pseudoinverse:

$$\dot{u}(t) = -d_u \left(D[\alpha(t), \cdot]\right)^\dagger_{|u(t)} \left(d_\alpha D(\cdot, u(t))_{|\alpha(t)}[\dot{\alpha}(t)]\right). \tag{167}$$

□

The theorem 6 gives a first-order infinitesimal variation of the function on the solution path. Thus, starting at $\mathbf{u_0}$, we could for instance perform a Euler approximation:

$$u_{k+1} = u_k + \eta_k \dot{u}_k \tag{168}$$

$$= u_k - \eta_k d_u \left(D[\alpha(k), \cdot]\right)^\dagger_{|u_k} \left(d_\alpha D(\cdot, u_k)_{|\alpha(k)}[\dot{\alpha}(k)]\right), \tag{169}$$

with $\eta_k \in (0,1]$ the size of the step, and $u_k := u(t_k)$ where $(t_k)_{k=1}^K$ is a time discretization such that $t_0 = 0$ and $t_K = 1$. The issue is that we cannot perform updates in the functional space. We would like to obtain the variation to perform in the parameter space to move along the solution curve. The first-order variation in the parameter space is given by the next theorem:

Theorem 7 (Implicit function theorem in parameter space)**.** *Let $\alpha : [0,1] \to \mathcal{A}$, a function that verifies $\alpha(0) = \boldsymbol{\alpha_0}$, and $\alpha_1 = \boldsymbol{\alpha_1}$, with $\mathcal{A} \subset \mathbb{R}$. Let $\theta : [0,1] \to \mathbb{R}^P$ the parameters of the parametric model such that $u : \begin{cases} [0,1] & \to \mathcal{H} \\ t & \mapsto u_{|\theta(t)} \end{cases}$ is a solution of the problem 3 with α. Suppose $u(0) = \mathbf{u_0}$, we have:*

$$\dot{\theta}(t) = -d_\theta \left(D[\alpha(t), u(\cdot)]\right)^\dagger_{|\theta(t)} \left(\Pi^\perp_{\mathcal{T}^{(D)}_{\theta(t)}} \left(d_\alpha D(\dot{\alpha}(t), u(t))_{|\alpha(t)}\right)\right), \tag{170}$$

is the derivative of θ in time, optimal in the least squares sense for the minimal norm, where $\mathcal{T}^{(D)}_\theta$ is the tangent space on the parametric model $D \circ u(\cdot)$ evaluated in θ (see Appendix B).

Proof. Let $t \in [0,1]$, from the assumptions, we have $D[\alpha_t, u(\theta(t))] = f$. We can derivate it with respect to t:

$$\frac{d}{dt} D[\alpha(t), u(\theta(t))] = 0, \tag{171}$$

We can apply chain rules, and obtain:

$$d_\alpha D[\cdot, u(\theta(t))]_{|\alpha(t)} \left(\dot{\alpha}(t)\right) + d_\theta D[\alpha(t), \cdot]_{|\theta(t)} \left(\dot{\theta}(t)\right) = 0, \tag{172}$$

thus,

$$d_\theta D[\alpha(t), u(\cdot)]_{|\theta(t)} \left(\dot{\theta}(t)\right) = -d_\alpha D[\cdot, u(\theta(t))]_{|\alpha(t)} \left(\dot{\alpha}(t)\right). \tag{173}$$

We can write the associated least squares problem:

$$\dot{\theta}(t) \in \arg\min_{h \in \mathbb{R}^P} \frac{1}{2} \left\| d_\theta D[\alpha(t), u(\cdot)]_{|\theta(t)} (h) + d_\alpha D[\cdot, u(\theta(t))]_{|\alpha(t)} \left(\dot{\alpha}(t)\right) \right\|_{L^2}^2. \tag{174}$$

The minimal norm solution of this optimization problem is the one defined using pseudoinverse (proposition 11):

$$\dot{\theta}(t) = -d_\theta \left(D[\alpha(t), u(\cdot)]\right)^\dagger_{|\theta(t)} \left(\Pi^\perp_{\mathcal{T}^{(D)}_{\theta(t)}} \left(d_\alpha D(\dot{\alpha}(t), u(t))_{|\alpha(t)}\right)\right). \tag{175}$$

□

G Generalization with a boundary equation

Previously, we considered the equation 3. However, we can generalize implicit function theorem when we add boundary equation. The new setting is the following:

$$\begin{cases} D[\alpha, u] & = f \quad \text{in } \Omega \\ B[u] & = g \quad \text{on } \partial\Omega. \end{cases} \tag{176}$$

We can rewrite this differential equation:

$$(D, B) \circ u = (f, g), \tag{177}$$

as presented in the appendix B.3.

In practice, we can add as many equations we want, however to ease the presentation we consider only the addition of a boundary equation. We can generalize the implicit function theorem 1 in the following way:

Theorem 8. *Let $\alpha : [0,1] \to \mathcal{A}$, a function that verifies $\alpha(0) = \boldsymbol{\alpha_0}$, and $\alpha(1) = \boldsymbol{\alpha_1}$, with $\mathcal{A} \subset \mathbb{R}$. Let $\theta : [0,1] \to \mathbb{R}^P$ be the parameters of the parametric model such that $u : \begin{cases} [0,1] & \to \mathcal{H} \\ t & \mapsto u_{|\theta(t)} \end{cases}$ is a solution of the problem 176 with α. Suppose $u(0) = \mathbf{u_0}$, we have:*

$$\dot{\theta}_t = -d_\theta \left((D, B) \circ (\alpha_t, u(\cdot))\right)^\dagger_{|\theta_t} \left(\Pi^\perp_{\mathcal{T}^{(D,B)}_{\theta_t}} \left[d_\alpha D(\cdot, u(\theta_t)_{|\alpha_t}(\dot{\alpha}_t), 0) \right] \right). \tag{178}$$

is the derivative of θ in time, optimal in the least squares sense for the minimal norm, where $\mathcal{T}^{(D,B)}_\theta$ is the tangent space in the parametric model $(D, B) \circ u(\cdot)$ evaluated in θ (see Appendix B).

Proof. Let $t \in [0,1]$, from the assumptions, we have $(D, B) \circ (\alpha_t, u(t)) = (f, g)$. We can derivate it with respect to t:

$$\frac{d}{dt}(D, B) \circ (\alpha_t, u(t)) = 0. \tag{179}$$

By chain rule, we have:

$$0 = d_\alpha((D, B) \circ (\cdot, u(t)))_{|\alpha_t}[\dot{\alpha}_t] + d_\theta((D, B) \circ (\alpha_t, u(\cdot)))_{\theta_t} \left(\dot{\theta}(t) \right). \tag{180}$$

Thus,

$$d_\theta((D, B) \circ (\alpha_t, u(\cdot)))_{\theta_t} \left(\dot{\theta}(t) \right) = -d_\alpha((D, B) \circ (\cdot, u(\theta_t)))_{|\alpha_t}[\dot{\alpha}_t] \tag{181}$$

$$= - \left(d_\alpha D(\cdot, u(\theta_t))_{|\alpha_t}, 0 \right). \tag{182}$$

Exactly as in the proof of theorem 7, we can take the pseudoinverse to conclude the demonstration of the theorem.

$$\dot{\theta}_t = -d_\theta \left((D, B) \circ (\alpha_t, u(\cdot))\right)^\dagger_{|\theta_t} \left(\Pi^\perp_{\mathcal{T}^{(D,B)}_{\theta_t}} \left[d_\alpha D(\cdot, u(\theta_t)_{|\alpha_t}(\dot{\alpha}_t), 0) \right] \right). \tag{183}$$

□