# Parametric Nonlinear Elliptic Problem:
# a comparison of POD, PINNs and POD-NN

Marta Di Ridolfi (LinkedIn profile),
Giulia Monchietto (LinkedIn profile).

**Abstract**

In the present study, the solution of a nonlinear elliptic problem is computed with the purpose of highlighting and analyzing the main differences between three methods of Model Order Reduction theory: POD, PINN and POD-NN. Firstly, the code implemented was tested on a benchmark problem and then it was adjusted in order to solve the project task. The numerical results are compared with the theoretical ones.

## 1. Introduction

The Reduced Order Methods are becoming, as time passes, more relevant. This is because, with classical Galerkin method, most of the parametric PDEs problems require a huge computational time to be solved. Model Order Reduction approaches provide an instrument to obtain a slightly worse approximation of the solution but in a much smaller computational time.

The paper is organised in three main sections, the first one of which briefly introduces the problem aim. The second section explains the theoretical features of the Reduced Order Methods (POD, PINN and POD-NN approaches) implemented to solve a parametric non-linear elliptic problem, followed by the numerical simulations of a benchmark problem and the project task. In conclusion the most relevant results are highlighted, by comparing the three methods used to solve the problem.

## 2. Parametric Nonlinear Elliptic Problem

The task of our project is to solve the following *Parametric Non-linear Elliptic Problem* (see [1]) $\forall \mu \in \mathcal{P}$

$$\begin{cases} -\Delta u(\mu) + \dfrac{\mu_0}{\mu_1}\left(e^{\mu_1 u(\mu)} - 1\right) = g & \text{in } \Omega \\ u = 0 & \text{in } \partial\Omega \end{cases} \tag{1}$$

where $\Omega = [0,1]^2$ and $\mu \in \mathcal{P} = [0.1, 1]^2$. Multiplying the left and the right hand side by a test function $v$ and integrating on $\Omega$, we obtain the weak formulation of the problem, i.e. Find $u(\mu) \in V := H_0^1(\Omega)$ s.t.

$$\underbrace{\int_\Omega \nabla u(\mu) \cdot \nabla v}_{f_1(u,v;\mu)} + \underbrace{\frac{\mu_0}{\mu_1}\int_\Omega \left(e^{\mu_1 u(\mu)} - 1\right) v}_{f_2(u,v;\mu)} - \underbrace{\int_\Omega g\, v}_{f_3(u,v;\mu)} = 0 \quad \forall v \in V, \ \ \forall \mu \in \mathcal{P}$$

or, equivalently, s.t.

$$f(u, v; \mu) := f_1(u, v; \mu) + f_2(u, v; \mu) + f_3(u, v; \mu) = 0 \quad \forall v \in V, \ \ \forall \mu \in \mathcal{P} \tag{2}$$

### 2.1. Benchmark problem

In order to validate the accuracy of the implemented algorithms, firstly we carry out an *a priori error estimation* on a benchmark problem: we will solve the Nonlinear Elliptic Problem choosing the exact solution

$$u_{ex}(\mathbf{x}; \mu) = 16\, x_0\, x_1\, (1 - x_0)(1 - x_1) \tag{3}$$

and computing the respective right hand side given by

$$g_{ex}(\mathbf{x};\mu) = -\Delta u_{ex}(\mathbf{x};\mu) + \frac{\mu_0}{\mu_1}\left(e^{\mu_1\, u_{ex}(\mathbf{x};\mu)} - 1\right) = \\ = 32(x_0(1-x_0) + x_1(1-x_1)) + \frac{\mu_0}{\mu_1}\left(e^{\mu_1\, 16\, x_0\, x_1\,(1-x_0)(1-x_1)} - 1\right) \tag{4}$$

Several theoretical results ensure that, by discretizing a variational problem using the *Galerkin method* with $\mathbb{P}_k$ finite elements (FE), if $u \in H^r(\Omega)$, $r \le k+1$, the following error bound holds:

$$\|u - u_h\|_{1,\Omega} \le Ch^{(r-1)/2}|u|_{r,\Omega}$$

where $h$ is the current greatest area of the discretization elements of $\Omega$. Thus, considering $\mathbb{P}_1$-FE and $u \in H^2(\Omega)$, we have

$$\mathcal{E}_0 = \|u - u_h\|_{L^2(\Omega)} \sim ch^1|u|_2 \quad \iff \quad \log(\mathcal{E}_0) \sim log(c\,|u|_{k+1}) + log(h) \tag{5}$$

$$\mathcal{E}_1 = \|u - u_h\|_{H^1_0(\Omega)} \sim ch^{1/2}|u|_2 \quad \iff \quad \log(\mathcal{E}_1) \sim log(c\,|u|_{k+1}) + \frac{1}{2}\,log(h) \tag{6}$$

Plotting $\mathcal{E}_0$ and $\mathcal{E}_1$ with respect to $h$ on a logarithmic base, we are expecting a linear decay in which the slopes $m_0 = 1$ and $m_1 = 1/2$ give the error convergence orders in $L^2-$norm and $H^1_0$-norm, respectively.

Therefore, knowing the exact solution (3) allows us to perform an a priori estimation of the error, which enables us to study the accuracy of the algorithms implemented to solve the problem of interest: (2) with the forcing term defined as described below.

### 2.2. Project task

Conducting an error analysis on the benchmark problem, we have an estimation of the accuracy of the solution provided by the implemented code. This allows us to proceed with our project task, which is to find the approximate solution of the problem (2) with

$$g(\mathbf{x};\mu) = 100\sin(2\pi x_0)\cos(2\pi x_1) \quad \forall, \mathbf{x} = (x_1, x_0) \in \Omega. \tag{7}$$

Given the absence of an exact solution for the problem, the error analysis will rely on the high-fidelity solution, which represents the most accurate approximation available for the parametric problem.

## 3. Model order reduction methods

This section introduces the cornerstones on which the computational solution relies on.

To obtain a numerical solution, we employ the *Galerkin method* to discretize the variational problem (2). introducing a subspace $V_\delta = span\{\varphi_i\}_{1\le i\le N_\delta} \subset V$ of dimension $N_\delta < +\infty$, the problem transforms into: Find $u_\delta(\mu) \in V_\delta$ s.t.

$$f(u_\delta, v_\delta; \mu) = 0 \qquad \forall v_\delta \in V_\delta,\ \forall \mu \in \mathcal{P}$$

Specifically, we will use $\mathbb{P}_1$-FE for discretizing the problem. However, obtaining a *high-fidelity* solution $u_\delta(\mu)\ \forall \mu \in \mathcal{P}$ is tipically unfeasible, due to its huge computational cost.

Model Order Reduction methods aim to construct an approximate solution $u_N(\mu)$ using several techniques, such us the *reduced basis approximation* or *neural networks* (NN), which are designed to heavily reduce time and computational costs.

### 3.1. POD

The aim of *Proper Orthogonal Decomposition* (POD) is to leverage Galerkin orthogonality to construct a *low-fidelity* solution using $N$ high-fidelity solutions $\{w_1 = u_\delta(\mu_1), \ldots, w_N = u_\delta(\mu_N)\} \subset V_\delta$, known as *snapshots*, where $N << N_\delta$. This method is more efficient under the *affine hypothesis*, where the dependency of $f$ on variables $\mathbf{x}$ is separated from its dependency on parameters $\mu$. In such cases, quantities independent of $\mu$ are computed once, with parameter-dependent quantities subsequently calculated.

While POD is optimal for parametric linear problems, it can also be applied to solve nonlinear ones by appropriately linearizing them using a Newton method. However, nonlinearities invalidate the affine hypothesis, reducing the method's efficiency.

### 3.1.1. Newton schema

Let's notice that our problem is nonlinear due to the presence of the $f_2(u, v; \mu)$ term. Thus, in order to implement the POD, we linearize the problem through the Newton scheme. Specifically, we fixed a Newton tolerance of $tol_N = 10^{-6}$ and a maximum number of iteration equal to $N_{max} = 20$. For each $k$ iteration of the algorithm we solve the problem: Find $\partial u$ s.t.

$$J_f[\partial u]_{|_{u_k}} \partial u = -f(u_k, v; \mu) = 0 \quad \forall v \in V \tag{8}$$

where $J_f[\partial u]_{|_{u_k}}$ is the evaluation of the derivative $J_f$ of $f$ in the point $u_k$ along the unknown direction of $\partial u$. Notice that the functions $f_1(u, v; \mu)$ and $f_3(u, v; \mu)$ are linear in $u$ and their *Gateaux derivatives* are given respectively by

$$J_{f_1}[\partial u]_{|_{u_k}} = f_1(u; v) = \int_\Omega \nabla \partial u \cdot \nabla v \qquad \text{and} \qquad J_{f_3}[\partial u]_{|_{u_k}} = 0$$

On the other hand, the Gateaux derivative of $f_2(u, v; \mu)$ is given by

$$\begin{aligned}
J_{f_2}[\partial u]_{|_{u_k}} &= \lim_{h \to 0} \frac{f_2(u_k + \partial u\, h) - f(u_k)}{h} = \\
&= \frac{\mu_0}{\mu_1} \lim_{h \to 0} \frac{1}{h} \int_\Omega \left(e^{\mu_1 (u_k + \partial u\, h)} - 1\right) v - \int_\Omega \left(e^{\mu_1 u_k} - 1\right) v = \\
&= \frac{\mu_0}{\mu_1} \lim_{h \to 0} \frac{1}{h} \int_\Omega e^{\mu_1 u_k} \left(e^{\mu_1 \partial u\, h} - 1\right) v = \\
&= \mu_0 \int_\Omega e^{\mu_1 u_k}\, \partial u\, v
\end{aligned}$$

which is a reaction term. Summing up, for each $k$ iteration, fixed $u_k$, the problem (8) become: Find $\partial u$ s.t.

$$\int_\Omega \nabla \partial u \cdot \nabla v + \mu_0 \int_\Omega e^{\mu_1 u_k}\, \partial u\, v = -\int_\Omega \underbrace{\nabla u_k}_{g_1} \cdot \nabla v - \int_\Omega \underbrace{\frac{\mu_0}{\mu_1} \left(e^{\mu_1 u_k} - 1\right) v}_{g_2} + \int_\Omega g\, v \tag{9}$$

Let us highlight that the first and the last terms do not depend on the solution $u_k$; therefore, we will compute them once, before implementing the iterative Newton method.

### 3.1.2. POD algorithm: offline phase

For the implementation of the POD, a random set of $M = 300$ parameters of $\mathcal{P}$ was chosen. For each parameter we compute the high fidelity solution by solving (9) and assemble the *snapshot matrix*

$$\mathbf{W} = [\mathbf{w}_1 = \mathbf{u}_\delta(\mu_1), \ldots, \mathbf{w}_M = \mathbf{u}_\delta(\mu_M)]$$

By introducing the matrix associated to the inner product with respect to the $H_0^1$-norm

$$[\mathbf{X}_\delta]_{ij} = (\nabla \varphi_j, \nabla \varphi_i)_{L^2} \tag{10}$$

we define the *covariance matrix* as

$$\mathbf{C} = \mathbf{W}\,\mathbf{X}_\delta\,\mathbf{W}^T$$

and we compute its eigenvalues and eigenvectors $\{(\lambda_i, \mathbf{v}_i)\}_{1\le i\le M}$. We choose the dimension $N$ of the reduced space as the maximum integer such that

$$\frac{\sum_{i=1}^{N}\lambda_i}{\sum_{j=1}^{M}\lambda_j} \ge tol_R$$

where $tol_R = 10^{-7}$. Lastly, the *basis matrix*, which led us to reduce the high-fidelity system, is given by

$$\mathbf{B} = [\psi_1, \dots, \psi_N] \quad \text{where } \psi_i = \frac{\mathbf{W}^T\mathbf{v}_i}{\|\mathbf{W}^T\mathbf{v}_i\|_{H_0^1}} \quad 1 \le i \le N \tag{11}$$

### 3.1.3. POD algorithm: online phase

After the offline phase of the algorithm, we can find a low-fidelity solution of the problem in the following step: fixed a parameter $\mu \in \mathcal{P}$, for each $k$ iteration of the Newton schema, we:

1. compute the **full** matrices $\mathbf{A}_\delta^\mu$, $\mathbf{R}_\delta^\mu$ and forcing terms $\mathbf{g}_\delta^\mu$, $\mathbf{g_1}_\delta^\mu$, $\mathbf{g_2}_\delta^\mu$ associated to the high-fidelity problem (9);
2. **reduce** the system as follows

$$\mathbf{A}_N^\mu = \mathbf{B}^T\mathbf{A}_\delta^\mu\mathbf{B} \qquad \mathbf{R}_N^\mu = \mathbf{B}^T\mathbf{R}_\delta^\mu\mathbf{B} \qquad \mathbf{g}_N^\mu = \mathbf{B}^T\mathbf{g}_\delta^\mu \qquad \mathbf{g_1}_N^\mu = \mathbf{B}^T\mathbf{g_1}_\delta^\mu \qquad \mathbf{g_2}_N^\mu = \mathbf{B}^T\mathbf{g_2}_\delta^\mu$$

3. **solve** the reduced system finding the increment $\partial\mathbf{u}_N^k$ used to update the reduced solution

$$\mathbf{u}_N^{k+1} = \mathbf{u}_N^k + \partial\mathbf{u}_N^k$$

4. **project** back the reduced solution to the $V_\delta$ space with the basis matrix (11)

$$\mathbf{u}_{\delta,\,proj}^{k+1} = \mathbf{B}\,\mathbf{u}_N^{k+1}$$

   We will use the projection $\mathbf{u}_{\delta,proj}^{k+1}$ to build the matrices and the forcing terms associated to the high fidelity problem in the next step of the Newton method.

From the algorithm described above, it is evident how the POD for nonlinear problems requires a huge computational cost.

### 3.1.4. Numerical results

In order to validate the accuracy of the implemented POD method, firstly we carry out an *a priori error estimation* on the benchmark problem, with exact solution (3), introduced above.

Fixing $\mu_0 = 0.8$ and $\mu_1 = 0.4$, we implement the POD algorithm to compute the reduced solution of the benchmark problem for $h \in \{0.1, 0.01, 0.001\}$. The errors in norm $L^2$ and $H_0^1$ with respect to $h$, depicted in Fig.1 on a logarithmic base, clearly decrease as the mesh becomes finer and finer. Specifically, the error convergence orders reported in Table 1 follow the one theoretically expected from (5) and (6). Therefore, the POD algorithm converges and we can apply it in order to solve the project task with an unknown solution.

| Error convergence orders for POD | |
|---|---|
| in norm $L^2$ | in norm $H_0^1$ |
| 0.94 | 0.47 |

**Table 1:** Convergence orders of the errors in norm $L^2$ and $H_0^1$ depicted in Fig.1.

Following a procedure similar to that used for solving the benchmark problem, we address the problem (2) with (7) as right-hand-side. Initially, we implement the offline phase of the POD algorithm, considering a domain discretization with a meshsize of $h = 0.001$. Subsequently, with fixed parameter values $\mu_0 = 0.8$ and $\mu_1 = 0.4$, we compute the reduced solution of the problem using the online phase.

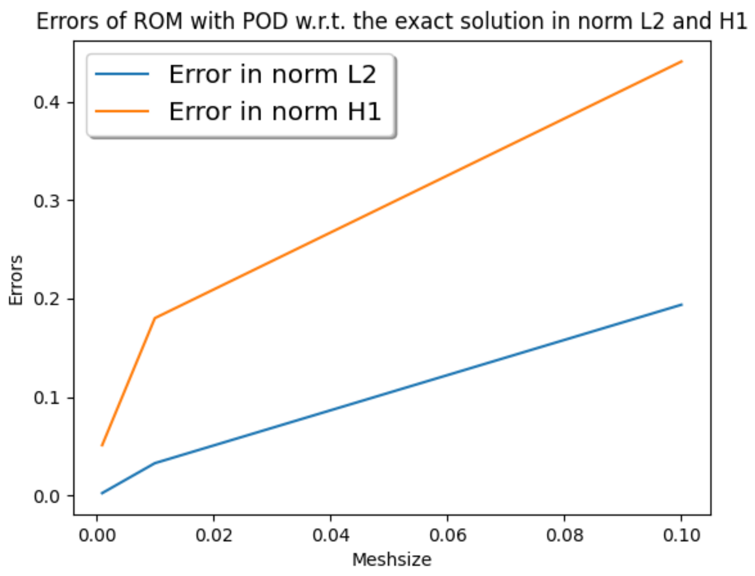

**Figure 1:** Errors between the exact solution $u_{ex}(\mathbf{x};\mu)$ and the reduced one $u_N(\mathbf{x};\mu)$, obtained with the POD method, in norm $L^2$ and $H_0^1$ with respect to the mesh size $h$, on a logarithm base.

The result is depicted in Fig. 2, alongside the high-fidelity solution, for the reasons outlined in the project task introduction. It can be deduced that, in the norm of the eye, the POD approach provides a good approximation of the high-fidelity solution. However, the approximation is slightly worse at points where the high-fidelity solution exhibits steep gradients.

To be more accurate we perform an **error analysis**. We compute the reduced solution $u_N(\mu)$ of the parametric nonlinear elliptic problem for parameter values $(\mu_0, \mu_1)$ chosen from a test set $\mathcal{P}_{test}$ of 10 pair of parameters randomly selected from $\mathcal{P}$. For each parameters value $\mu \in \mathcal{P}_{test}$, we compute the relative errors, w.r.t. the high-fidelity solution $u_\delta(\mu)$, measured in the $L^2$ and $H_0^1$ norms, defined respectively as

$$\mathcal{E}_0^{rel} = \frac{\|u_\delta(\mu) - u_N(\mu)\|_{L^2(\Omega)}}{\|u_\delta(\mu)\|_{L^2(\Omega)}} \tag{12}$$

$$\mathcal{E}_1^{rel} = \frac{\|u_\delta(\mu) - u_N(\mu)\|_{H_0^1(\Omega)}}{\|u_\delta(\mu)\|_{H_0^1(\Omega)}} \tag{13}$$

By averaging the resulting values, we obtain the errors shown in Table 2. It is evident that the reduced solution found with the POD method give a good approximation of the discretized one, obtained with the Galerkin method.

| Averaged relative errors for POD | |
|---|---|
| in $L^2$ norm | in $H_0^1$ norm |
| $1.29 \cdot 10^{-5}$ | $1.04 \cdot 10^{-4}$ |

**Table 2:** Averaged relative errors of the reduced solution with POD w.r.t. the high-fidelity one, in the $L^2$ and $H_0^1$ norms.

Lastly, since the main purpose of ROMs is to reduce the time and computational costs of the classical Galerkin method, we focus on studying the **speed-up** ratio, computed as follows

$$SU = \frac{T_{\text{a, FOM}} + T_{\text{r, FOM}}}{T_{\text{a, ROM}} + T_{\text{r, ROM}}} \tag{14}$$

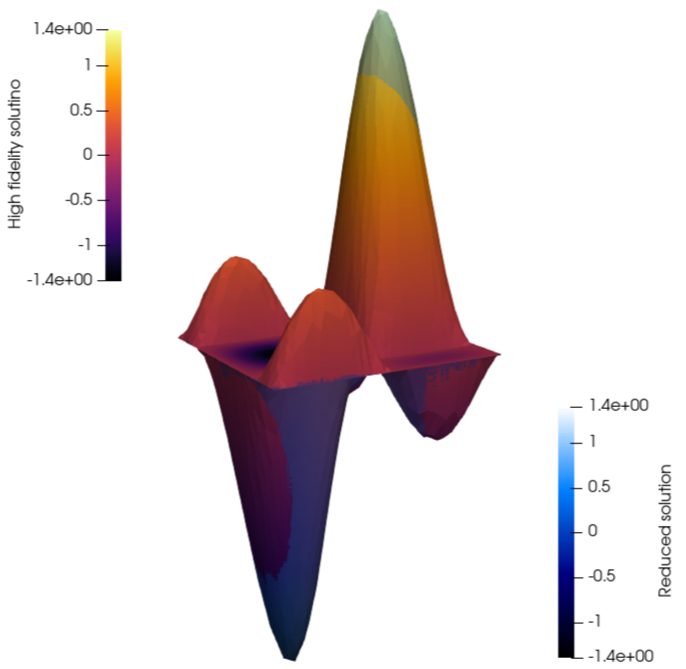

**Figure 2:** Plot of the reduced solution (in blue with opacity) to the task problem using the POD algorithm, compared to the high-fidelity solution.

where $T_{\text{a, FOM}}$ and $T_{\text{r, FOM}}$ denote the assembly and resolution times of the problem solved with the Full Order Method, respectively, and similarly, $T_{\text{a, ROM}}$ and $T_{\text{r, ROM}}$ those of the solution computed with the Reduced Order Method. Similar to the error analysis, we compute the speed-up for each $\mu \in \mathcal{P}_{test}$, resulting in an average speed-up of

$$SU_{mean} = 0.79$$

Thus, the FOM turns out to be slightly faster w.r.t. the POD. Indeed, as previously discussed in the theoretical introduction of the POD method, despite the reduced system's solution time being lower compared to that of the full system, the assembly time of the former is found to be greater than the FOM one. The latter behaviour can be attributed to the non-linearity of the problem, which necessitates the majority of the terms to be built online at each Newton iteration, thus increasing the computation cost.

Summing up, the POD method provides a good approximation of the high-fidelity solution for the parametric nonlinear elliptic problem, but it is suboptimal in terms of computational time.

## 3.2. PINN

Physics-Informed Neural Networks (PINNs) represent a novel approach for solving partial differential equations (PDEs), including parametric nonlinear elliptic problems, by leveraging the capabilities of neural networks while embedding physical laws directly into the loss function. Unlike traditional numerical methods, PINNs do not require the discretization of the computational domain and can handle high-dimensional parameter spaces efficiently. This makes them particularly suitable for parametric PDEs where computational costs are a significant concern.
The key idea is to train a neural network, with domain points $\mathbf{x} \in \Omega_{train} \subset \Omega$ and parameters $\mu \in \mathcal{P}_{train} \subset \mathcal{P}$ as inputs, to approximate the solution $u(\mathbf{x}; \mu)$ such that the network output satisfies the given PDE and the associated boundary conditions. The essential components of PINNs include the neural network architecture, the definition of the loss function incorporating the residual of the problem, and the training process.

### 3.2.1. PINN's architecture

As architecture a feed-forward neural network with $L = 4$ hidden layers and sigmoid activation functions, has been chosen. The number of input and output channels has been determined according to the *fixed-width and variable-depth Universal Approximation theorem*. The latter is indeed satisfied, since sigmoid activation function is infinitely differentiable. The input dimension is $n = 4$ (2 dimension for the domain and 2 parameters) and the output dimension is $m = 1$, thus for each layer $l \in \{2, ..., L-1\}$ of the NN we fixed

$$N_l = n + m + 2 = 7$$

nodes.

### 3.2.2. Loss function

As mentioned above, the physics in PINNs is introduced in the loss, defined as the sum of two components

$$\mathcal{L} = \mathcal{L}_r + \lambda \cdot \mathcal{L}_b \tag{15}$$

where $\mathcal{L}_r$ is the mean squared residual of the problem and $\mathcal{L}_b$ is the mean squared error of the NN prediction on the boundary points. Specifically, for the problem (1) we have

$$\mathcal{L}_r = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} [R(\mathbf{x}_i; \mu_i)]^2 \quad \text{with} \quad (\mathbf{x}_i; \mu_i) \in \Omega_{train} \times \mathcal{P}_{train} \subset \Omega \times \mathcal{P}$$

where

$$R(\mathbf{x}; \mu) = \left[ -\Delta u(\mu) + \frac{\mu_0}{\mu_1} \left( e^{\mu_1 u(\mu)} - 1 \right) - g \right] \Bigg|_{(\mathbf{x};\mu)}$$

and, due to the homogeneous Dirichlet boundary condition

$$\mathcal{L}_b = \frac{1}{N_{train}} \sum_{i=1}^{N_{train}} [u(\mathbf{x}_i^b; \mu_i)]^2 \quad \text{with} \quad (\mathbf{x}_i^b; \mu_i) \in \partial\Omega_{train} \times \mathcal{P}_{train} \subset \partial\Omega \times \mathcal{P}$$

A coefficient, $\lambda = 3000$, is introduced to balance these terms, ensuring they are of the same order of magnitude. Indeed, it has been estimated by the ratio of the first epoch values of $\mathcal{L}_r$ and $\mathcal{L}_b$.
Thus, by minimizing $\mathcal{L}$ during the training process, the network learns to approximate the solution $u(\mathbf{x}; \mu)$ such that it satisfies both the PDE and the given boundary condition of (1).

### 3.2.3. PINN's training

The training of the Physics-Informed Neural Network (PINN) involves an unsupervised learning process where the network parameters are optimized to minimize the loss (15), without the use of given labels. In the present study, it was used a *meshless method*. Indeed, we randomly selected a set of $N_{train} = 800$ points $\mathbf{x}_i \in \overset{\circ}{\Omega}$ within the domain, a set of $N_{train} = 800$ points $\mathbf{x}_i^b \in \partial\Omega$ on the boundary (200 for each edge of the square) and a set of $N_{train} = 800$ pairs of parameters $(\mu_0, \mu_1) \in \mathcal{P}$. The training process has been conducted using a number of epochs, chosen based on the analysis of the training loss function plot, ensuring the network does not overfit the data and maintains a balance between training accuracy and generalization. Hyperparameter tuning was conducted for what concerns the optimal learning rate for training the PINN.

After the training phase, in order to perform an error analysis we discretize the spatial domain $\Omega$ with a mesh. By fixing a pair of parameter values $(\mu_0, \mu_1)$, we predict the solution at the degrees of freedom of the discretization $\Omega_h$.

### 3.2.4. Numerical results

Unlike what was done for the POD analysis, in this case we do not perform a priori error estimation on the benchmark problem to validate the PINN. Indeed, this latter approach is unrelated to the theoretical results concerning the a priori error estimation used for the validation of POD. Therefore, we directly address the project task using a PINN with the architecture, loss function, and training set described previously.

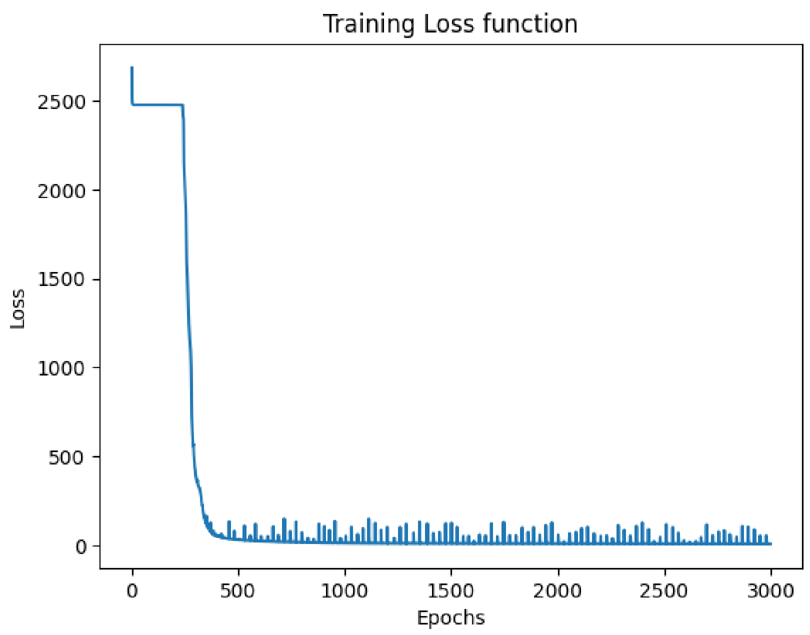

**Figure 3:** Loss function during the PINN training.

First, we analyze the training loss over the epochs. The corresponding plot is reported in Fig.3 where the $x$-axis represents the $(i \cdot 10)$-th epoch, as the loss was recorded every 10 epochs. The learning process appears to stabilize after approximately $N_{epochs} = 1 \cdot 10^4$ epochs, thus we choose this value as the termination point for the PINN's training.

Moreover, to optimize the learning rate $lr$ for the PINN, we implement three different numerical simulations with $lr \in \{0.1, 0.01, 0.001\}$. Analyzing the average of the loss function $\mathcal{L}$ values over the last 20 epochs, a learning rate of $lr = 0.01$ resulted in the lowest averaged value of the loss function, $\mathcal{L}^{20}_{mean} \approx 0.32$, as shown in Table 3. Consequently, we selected a learning rate of $lr = 0.01$ for the final model training.

| Learning rate tuning for PINN | | | |
|---|---|---|---|
| $lr$ | 0.1 | 0.01 | 0.001 |
| $\mathcal{L}^{20}_{mean}$ | 0.71 | 0.32 | 10.12 |

**Table 3:** Learning rate tuning for the PINN, with $lr \in \{0.1, 0.01, 0.001\}$

By considering a domain discretization $\Omega_h$ with a meshsize of $h = 0.001$ and fixing the parameter values $\mu_0 = 0.8$ and $\mu_1 = 0.4$, we predict the reduced solution of the problem on the degrees freedom of the mesh, using the trained PINN. The result is depicted in Fig. 4, alongside the high-fidelity solution obtained with FOM for the same parameter values and the same space discretization $\Omega_h$ chosen for the PINN prediction. Visually, it can be observed that the solution predicted by the PINN does not closely approximate the high-fidelity solution, particularly at points where the high-fidelity solution exhibits steep gradients.

Similarly to what we have seen in the **error analysis** of the POD approach, we compute the predicted solution $u_N(\mu)$ in the degrees of freedom of $\Omega_h$ for parameter values $(\mu_0, \mu_1)$ chosen from the same test set $\mathcal{P}_{test}$ of 10 pair of parameters randomly selected for the POD method. For each parameters value $\mu \in \mathcal{P}_{test}$, we compute the relative errors, w.r.t. the high-fidelity solution $u_\delta(\mu)$, measured in the $L^2$ and $H^1_0$ norms, defined respectively as in (12) and (13). By averaging the values, we obtain the errors shown in Table 4. It is evident that the approximated solution predicted with the PINN method give a rough approximation of the high-fidelity one, obtained with the Galerkin method.

Lastly, we perform the **speed-up analysis**. Since the time $T_p$ taken by the PINN to provide predictions for a solution is very short, it tends to be highly variable, even for predicting the same

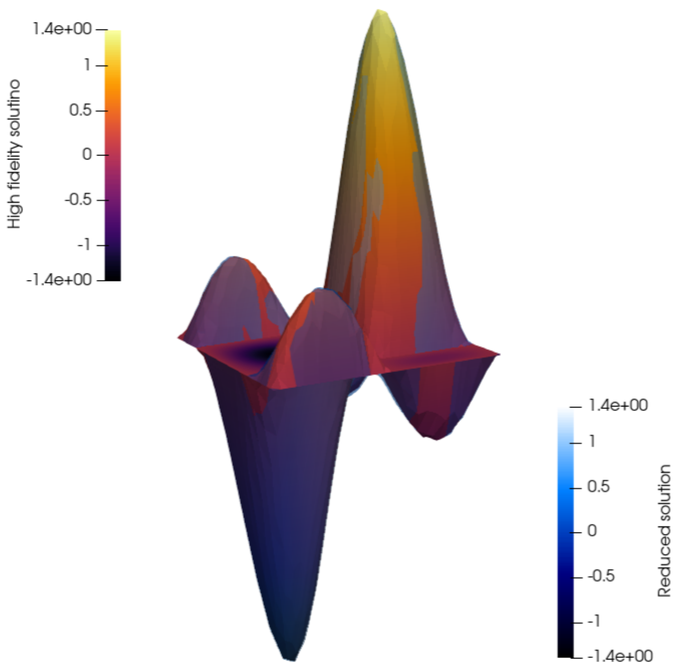

**Figure 4:** Plot of the predicted solution (in blue with opacity) of the task problem using the PINN algorithm, compared to the high-fidelity solution.

| Averaged relative errors for PINN | |
|---|---|
| in $L^2$ norm | in $H^1_0$ norm |
| $1.36 \cdot 10^{-2}$ | $4.72 \cdot 10^{-2}$ |

**Table 4:** Averaged relative errors of the reduced solution with PINN w.r.t. the high-fidelity one, in the $L^2$ and $H^1_0$ norms.

solution with fixed inputs. Therefore, considering the spatial discretization $\Omega_h$, for a given parameter $\mu \in \mathcal{P}$, we define the *prediction time* of the PINN as the average time taken to perform 100 predictions of the same approximated solution

$$T_{PINN}(\mu) = \frac{1}{100}\sum_{i=1}^{100} T_{p,i}(\mu)$$

Subsequently, for each pair of the 10 parameters in the test set, we compute the prediction time of the PINN $T_{PINN}(\mu)$ and the speed-up, defined as in (14), by substituting the denominator with $T_{PINN}(\mu)$. The average speed-up across all $\mu \in \mathcal{P}_{test}$ turns out to be

$$SU_{mean} \approx 278$$

Thus, the PINN rapidly provides an approximation of the PDE solution compared to the FOM for computing the high-fidelity solution. However, the predicted solution exhibits lower accuracy compared to the latter.

## 3.3. POD-NN

The POD-NN method combines the strengths of both POD and neural networks to create a powerful tool for the approximation of parametric solutions. Proper Orthogonal Decomposition, as explained above, captures the most significant features of the solution space, allowing for a reduced representation, while neural networks are exploited for approximating nonlinear mappings. Within this framework, the methodology of POD-NN is founded upon three components: the construction of the reduced space $V_N$ with the POD techniques; the development and training of a neural network $\Pi^{NN} : \mathcal{P} \to V_N$ to predict the reduced solution $u_N(\mu)$; the reconstruction of the discretized solution $u_{\delta,\,proj} \in V_\delta$.

### 3.3.1. POD-NN's architecture and training

During the development and training phase of the feedforward neural network (FFNN) in the POD-NN methodology, the network is trained to predict a reduced solution of the problem given a parameter pairs in input. This approach helps in improving the performance of the POD algorithm for nonlinear problems by eliminating the projection steps typically performed at each iteration of the Newton method, which lead to increased computational time and cost.

The FFNN designed for this purpose in the present work comprises $L = 5$ hidden layers, each utilizing the Softplus activation function (see [2]). The selection of Softplus was substantiated through empirical performance evaluations, wherein various activation functions were tested to determine the most effective one. A trial-and-error methodology was also employed to find the optimal number of nodes per hidden layer. It was determined that an architecture with $N_l = 30$ nodes per hidden layer, where $1 \le l \le L$, provided good performance.

For the objective function, the mean squared error (MSE) loss was employed, with a proper projection of the basis functions which generate the reduced space $V_N = span\{\psi_1, \dots, \psi_N\}$

$$\mathcal{L} = \sum_{\mu \in \mathcal{P}_{train}} \|\Pi^{NN}(\mu) - \mathbf{P}_g \mathbf{u}_\delta(\mu)\|^2$$

with $\mathcal{P}_{train} = \{\mu_1, \dots, \mu_N\} \subset \mathcal{P}$. It can be proved that an optimal orthogonal projector $\mathbf{P}_g$ is given by

$$\mathbf{P}_g \mathbf{u}_\delta^\mu = (\underbrace{\mathbf{B}^T \mathbf{X}_\delta \mathbf{B}}_{\mathbf{X}_N})^{-1} \mathbf{B}^T \mathbf{u}_\delta^\mu$$

where $\mathbf{B}$ is the basis matrix defined as in (11) and $\mathbf{X}_\delta$ is the inner product matrix defined as in (10). Thus, using that $\mathbf{P}_g \mathbf{u}_\delta^\mu = \mathbf{u}_N^\mu$ we obtain

$$\mathbf{u}_N^\mu = \mathbf{X}_N^{-1} \mathbf{B}^T \mathbf{u}_\delta^\mu \tag{16}$$

where $\mathbf{X}_N = \mathbf{B}^T \mathbf{X}_\delta \mathbf{B}$ is the reduced version of the inner product.

During the training process, it was observed that the training loss exhibited oscillations after reaching a certain number of epochs. To address this instability and enhance the convergence behavior, a learning rate scheduler was introduced. Specifically, a step scheduler was implemented, which reduces the learning rate by a factor of 0.5 every 1000 epochs. This adjustment improved the training process, leading to lower and more stable loss metrics, as we will later see in the numerical results.

### 3.3.2. POD-NN algorithm

Summing up, we can outline the POD-NN algorithm in the following steps:

- **Offline phase**:
  1. build the **reduced space** $V_N$, with the offline phase of the POD algorithm, computing the basis matrix $\mathbf{B}$ defined as in (11);
  2. **solve** the linear system
  $$\mathbf{X}_N \mathbf{u}_{N,i} = \mathbf{B}^T \psi_{\mathbf{i}} \quad 1 \le i \le N$$
  3. **train** the neural network $\Pi^{NN}$ with the reduced solutions $\{\mathbf{u}_{N,1}, \dots, \mathbf{u}_{N,N}\}$ computed at the previous step.

- **Online phase**: given a new parameter $\mu^*$, the reduced solution is predicted by the trained neural network
  $$\Pi^{NN}(\mu^*) = \mathbf{u}_N^{NN}(\mu^*)$$
  Then it is projected back to the discretized space $V_\delta$ by $\mathbf{u}_{\delta,proj}^\mu = \mathbf{B}\mathbf{u}_N^{NN}$.

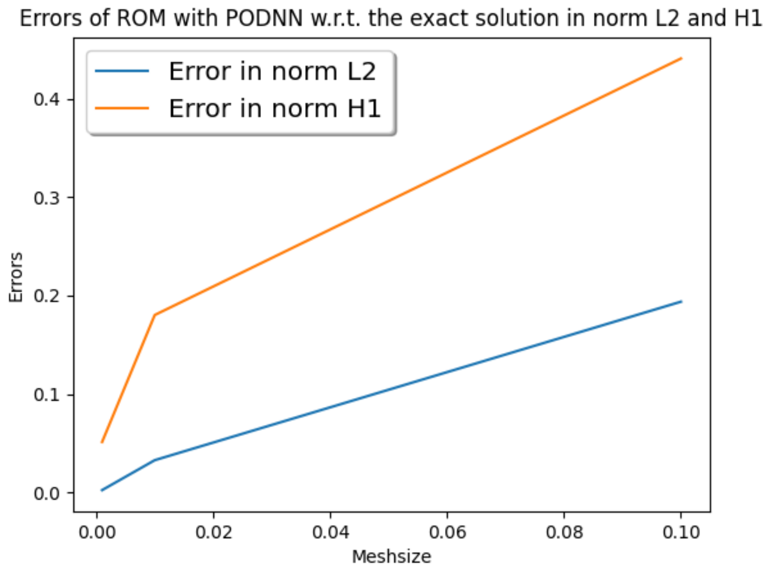

**Figure 5:** Errors between the exact solution $u_{ex}(\mathbf{x};\mu)$ and the reduced one $u_N(\mathbf{x};\mu)$, obtained with the POD-NN method, in norm $L^2$ and $H_0^1$ with respect to the mesh size $h$, on a logarithm base.

### 3.3.3. Numerical results

Similarly to what we have performed for the POD method, we firstly proceed by validating the POD-NN algorithm, applying it to solve the benchmark problem. Fixing $\mu_0 = 0.8$ and $\mu_1 = 0.4$, we compute the POD-NN's predictions of the benchmark problem solution for different mesh sizes $h \in \{0.1, 0.01, 0.001\}$ and we study the errors in $L^2$ and $H_0^1$ norms with respect to $h$. Depicting them on a logarithmic base in Fig.5, we can notice an evident decrease of the errors as the mesh becomes finer and finer. Specifically, the error convergence orders reported in Table 5 follow the one theoretically expected from (5) and (6). Therefore, the POD-NN algorithm is well-posed and converges. We can now perform the method in order to solve our project task with an unknown solution.

| Error convergence orders for POD-NN | |
|---|---|
| in $L^2$ norm | in $H_0^1$ norm |
| 0.96 | 0.47 |

**Table 5:** Convergence orders of the errors in norm $L^2$ and $H_0^1$ depicted in Fig.5.

First, we analyze the training loss over the epochs to assess the performance and convergence of the neural network. By plotting the log-scaled training loss over the epochs in Fig. 6, it is evident that the loss stabilizes at a low value, in the order of $10^{-4}$, after approximately 40000 epochs, indicating that convergence has been achieved.

By considering a domain discretization $\Omega_h$ with a meshsize of $h = 0.001$ and fixing the parameter values $\mu_0 = 0.8$ and $\mu_1 = 0.4$, we predict the reduced solution of the problem using the trained POD-NN. The result is depicted in Fig. 7, alongside the high-fidelity solution obtained with FOM for the same parameter values and the same space discretization used for the training and the prediction of the POD-NN. In the norm of the eye, it seems that the POD-NN approach provides a very good approximation of the high-fidelity solution.

To be more accurate, the POD-NN algorithm has been tested employing the same test set of parameters values $\mathcal{P}_{test}$ as previous methods. Computing the average $L^2$ and $H_0^1$ errors, the values in Table 6 have been obtained. The numerical results of the POD-NN method exhibit promising outcomes. The average error in $H_0^1$ norm demonstrates slightly worse performance than that achieved by the FOM, affirming the method's efficacy in capturing the solution's gradient behavior. However,

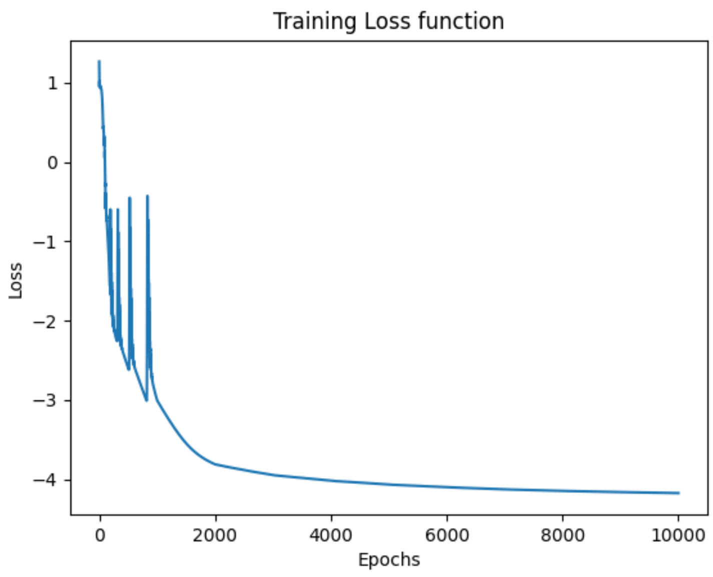

**Figure 6:** FFNN log scaled training loss of POD-NN method applied to solve the project task.

the average error in $L^2$ norm shows a slight deviation from the FOM method, indicating slight inferior accuracy in the overall solution approximation. Nonetheless, the good approximation of the FOM obtained, can be attributed to the leverage of FOM solutions as target values during the neural network's training phase.

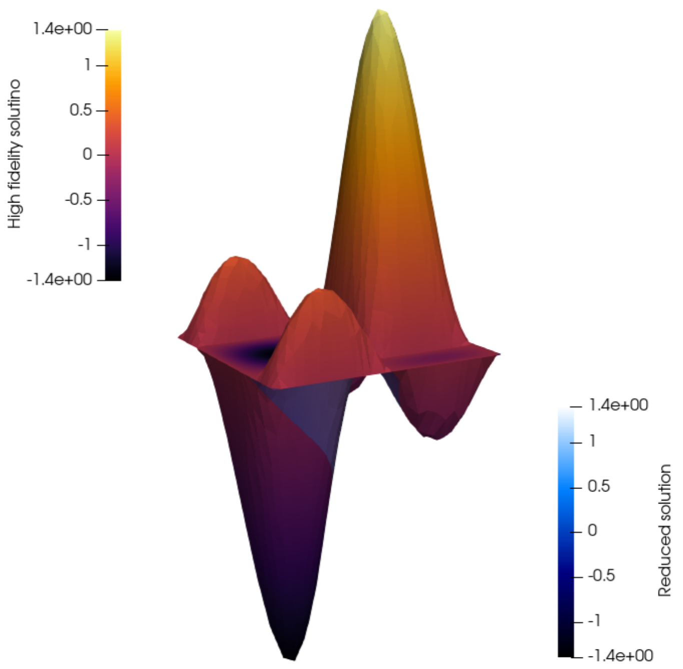

**Figure 7:** Plot of the reduced solution (in blue with opacity) to the task problem using the POD-NN algorithm, compared to the high-fidelity solution, reported in a opaque blue.

| Averaged relative errors for POD-NN | |
|---|---|
| in $L^2$ norm | in $H_0^1$ norm |
| $0.914 \cdot 10^{-3}$ | $0.915 \cdot 10^{-3}$ |

**Table 6:** Averaged relative errors of the reduced solution with POD-NN w.r.t. the high-fidelity one, in the $L^2$ and $H_0^1$ norms.

Particularly noteworthy is the substantial computational acceleration achieved, evidenced by a speed-up factor of $SU_{mean} \approx 3217$ compared to FOM. This acceleration, accompanied by a slightly worse accuracy than the FOM, highlights the efficiency and practical utility of the POD-NN method.

## 4. Conclusions

The three techniques evaluated in this report—Proper Orthogonal Decomposition (POD), Physics-Informed Neural Networks (PINNs), and POD-Neural Network (POD-NN)—represent three different approaches to model order reduction, aiming to decrease time and computational costs while maintaining a reasonable accuracy in the solution of a parametric nonlinear elliptic problem.

By comparing the performance of the three methods we can highlight that the POD algorithm is the most accurate in terms of average relative errors in $L^2$ and $H_0^1$ norms w.r.t. the high fidelity solution, as reported in Table 7. However, due to the non-linearities in our problem, it fails to significantly reduce the computational time of FOM. In contrast, the PINN approach demonstrates enhanced inference speed but lower precision in solution prediction, with errors three orders of magnitude greater than those of the POD. Ultimately, the POD-NN method emerges as the most promising, achieving a balance between inference speed and precision. It provides results nearly as precise as those of the POD but with significantly lower inference time. In conclusion, the POD method is preferable in case of linear variational problems; the PINN is the worst ROM and the POD-NN the best one for the approximation of a reduced solution of the parametric nonlinear elliptic problem solved in the present work.

| | Error in $L^2$ norm | Error in $H_0^1$ norm | Speed-up |
|---|---|---|---|
| POD | $1.29 \cdot 10^{-5}$ | $1.89 \cdot 10^{-4}$ | 0.79 |
| PINN | $1.36 \cdot 10^{-2}$ | $4.72 \cdot 10^{-2}$ | 278 |
| POD-NN | $0.914 \cdot 10^{-3}$ | $0.915 \cdot 10^{-3}$ | 3217 |

**Table 7:** ROM's comparison in terms of the average relative errors in $L^2$ and $H_0^1$ norms, and the speed-up.

## References

[1] F. Pichi, B. Moya, J. S. Hesthaven, A graph convolutional autoencoder approach to model order reduction for parametrized pdes, Journal of Computational Physics 501 (2024) 112762.

[2] J. Hesthaven, S. Ubbiali, Non-intrusive reduced order modeling of nonlinear problems using neural networks, Journal of Computational Physics 363 (2018) 55–78.