Comparative Analysis of Model Order Reduction Techniques for Nonlinear Parametric Differential Problems

Luca Biscu, Leonardo Tredese

Abstract

This work aims to compare three different Model Order Reduction (MOR) approaches applied to a nonlinear differential problem, evaluating their effectiveness in terms of accuracy and computational costs. Initially, a Full Order Model (FOM) is solved using the Newton method for linearization alongside the classic Galerkin approach. Subsequently, a Reduced Basis Method (RBM) employing the classical Proper Orthogonal Decomposition (POD) is utilized. The analysis continues with the implementation of Machine Learning techniques to enhance simulation speed. In particular, POD with neural networks (POD-NN) and Physics Informed Neural Network (PINN) strategies are exploited. The findings indicate that the presence of a nonlinear term, along with a parameter within it, complicates the application of classical Model Order Reduction techniques. This complication arises because the nonlinear term must be computed during the online phase, significantly increasing computational time. Conversely, Machine Learning approaches appear to be faster, but controlling the error is more complex, leading to reduced precision.

GitHub Repository

Keywords: Model Order Reduction, RMB, POD, POD-NN, PINN

1. Introduction

Nonlinear parametric models play a crucial role across numerous scientific domains and applications. Many systems are described by nonlinear equations, which present significant computational challenges. Solving these nonlinear systems is often resource-intensive, necessitating effective strategies to manage and reduce their complexity.

Model Order Reduction (MOR) techniques prove to be useful tools in addressing this complexity by simplifying the models without substantial loss of accuracy.

However, traditional MOR techniques such as the Reduced Basis Method with Proper Orthogonal Decomposition (RBM-POD), which are effective for linear problems, encounter significant issues when applied to nonlinear contexts. The primary challenge arises from the inability to exploit the affine hypothesis, necessitating the online computation of the nonlinear terms and furthermore their recalculation at each step of Newton's method. Consequently, the speedup from using RBM-POD is minimal, leading to performance gains that are often negligible compared to the Full Order Model (FOM).

To overcome these limitations, neural networks provide an alternative. One such approach is the Proper Orthogonal Decomposition with neural networks (POD-NN), which bypasses the need for the affine hypothesis since the projection stage is not performed, thus the speedup is guaranteed. Another solution is the use of Physics-Informed Neural Networks (PINNs), which are mesh-free and do not require the variational form of the problem.

These techniques can pontentially change the way nonlinear parametric models are handled, providing efficient solutions that bridge the gap between accuracy and computational feasibility. The principal limitation is that it is challenging to estimate the error and guarantee convergence, which compromises their precision.

This study compares these three approaches in terms of accuracy and speed when applied to a simple nonlinear equation, and tries to draw conclusions regarding their advantages and disadvantage.

The implementation of the methods was carried out using a Python interface with the GEDIM [1] library, a C++ library responsible for mesh generation and solving the algebraic forms of discretized problems.

2. Case study: nonlinear Poisson equation

The case study consists of the following parametrized problem. Let us consider a two-dimensional spatial domain $\Omega = (0, 1)^2$, and a two-dimensional parametric space $\mathcal{P} = [0.1, 1]^2$.

the (nonlinear) problem is:

$$\begin{cases} given\ \boldsymbol{\mu} = (\mu_0, \mu_1) \in \mathcal{P},\ find\ u(\boldsymbol{\mu})\ s.t.: \\ -\Delta u(\boldsymbol{\mu}) + \frac{\mu_0}{\mu_1}(e^{\,\mu_1 u(\boldsymbol{\mu})} - 1) = g(\boldsymbol{x}; \boldsymbol{\mu}) & in\ \Omega \\ u = 0 & on\ \partial\Omega \end{cases} \tag{1}$$

where the forcing term is given by:

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

The weak-form of this problem is the following:

$$\begin{cases} given\ \boldsymbol{\mu} = (\mu_0, \mu_1) \in \mathcal{P},\ find\ u(\boldsymbol{\mu}) \in H^1(\Omega)\ s.t.: \\ \int_\Omega \nabla u \cdot \nabla v \, d\mathbf{x} + \frac{\mu_0}{\mu_1}\int_\Omega (e^{\,\mu_1 u} - 1) v d\mathbf{x} = \int_\Omega g(\mathbf{x}; \mu) v \, d\mathbf{x} \\ \qquad\qquad\qquad\qquad \forall v \in H^1_0(\Omega) \\ u = 0 \quad on\ \partial\Omega \quad (in\ trace\ sense) \end{cases} \tag{2}$$

where the u dipendence from the parameter $\boldsymbol{\mu}$ has been dropped for the sake of notation. The existence of a weak solution for this problem can be proved using, for instance, the sub-supersolution method [2].

3. Full Order Model

Let's begin by addressing the nonlinear problem using a high-fidelity approach.

Due to the nonlinear nature of the problem, Newton's method is needed. Thus, the equation has to be expressed in the form:

$$f_\mu(u; v) = 0 \quad \forall v \in H^1_0(\Omega)$$

where:

$$f_\mu(u; v) = \int_\Omega \nabla u \cdot \nabla v \, d\mathbf{x} + \frac{\mu_0}{\mu_1}\int_\Omega (e^{\,\mu_1 u} - 1) v d\mathbf{x} - \int_\Omega g(\mathbf{x}; \mu) v \, d\mathbf{x}$$

Now, using Newton's method, we want to find the zeros u of $f_\mu(u; v)\ \forall v \in H^1_0(\Omega)$. The method proceeds iteratively, so for a generic step $n + 1$, we compute the quantity:

$$u_{n+1} = u_n + \delta u_{(n+1)}$$

where $\delta u_{(n+1)}$ is the correction we need to perform on u_n that is known from the previous step. In order to find $\delta u_{(n+1)}$, the following linear problem has to be solved:

$$J^f(u_n, \boldsymbol{\mu}; v)[\delta u] = -f_\mu(u_n; v) \tag{3}$$

where $J^f(u_n, \boldsymbol{\mu}; v)[\delta u]$ is the Fréchet derivative of $f_\mu(\cdot; v)$ on u_n along δu , which is given by:

$$J^f(u_n, \boldsymbol{\mu}; v)[\delta u] = \int_\Omega \nabla \delta u \cdot \nabla v \, d\mathbf{x} + \mu_0 \int_\Omega \delta u e^{\,\mu_1 u_n} v d\mathbf{x}$$

Thus, the explicit form of the linearized problem becomes:

$$\int_\Omega \nabla \delta u \cdot \nabla v \, d\mathbf{x} + \mu_0 \int_\Omega \delta u e^{\mu_1 u} v \, d\mathbf{x} = \tag{4}$$
$$- \int_\Omega \nabla u_n \cdot \nabla v \, d\mathbf{x} - \frac{\mu_0}{\mu_1}\int_\Omega \left(e^{\mu_1 u_n(\mu)} - 1\right) v \, d\mathbf{x} + \int_\Omega g(\mathbf{x}; \boldsymbol{\mu}) v \, d\mathbf{x}$$

with the homogeneus boundary conditions. Thanks to the linearity, this problem can be solved with the Galerkin approach. The Newton method continues until the convergence is reached. Usually, the stop criterion requires:

$$\frac{||\delta u||_{H^1}}{||u_n||_{H^1}} < tol$$

3.1. FOM solver certification

Before solving the test case problem, we need to ensure that the FOM implementation converges. Thus, we take the following function:

$$u(x, y) = 16xy(1 - x)(1 - y)$$

and we solve the FOM model with a forcing term given by the substitution of $u(x, y)$ inside the LHS of our equation. From the theory on error estimations for elliptc problems, we know that the error for first and second-order finite elements (P1 and P2 respectively), has to satisfy the following inequalities:

$$||u - u_\delta||_{0,\Omega} \le C\delta^{\frac{r}{2}} |u|_{r,\Omega}\ , \quad u \in H^r(\Omega)\ ,\ q \le r \le k + 1$$

$$||u - u_\delta||_{1,\Omega} \le C\delta^{\frac{r-1}{2}} |u|_{r,\Omega}\ , \quad u \in H^r(\Omega)\ ,\ q \le r \le k + 1$$

where k is the polinomial degree of the Finite Element chosen, and δ is the area of the triangles. In case of maximum regularity for the solution, the expected convergence orders are:

	P1 elements	P2 elements
L^2	1	3/2
H^1	1/2	1

Table 1: Theoretical convergence orders for max-regular solutions

By varying the area of the mesh generated by GeDim, we obtain the following L^2 and H^1 norm errors and the related convergence order:

Area	0.01	0.001	0.0001	Convergence
L^2	3.42e-2	2.63e-3	2.64e-4	1.0562
H^1	1.82e-1	5.13e-2	1.62e-2	0.5253

Table 2: 1st-order elements, log-log triangle area vs error in different norms

Area	0.01	0.001	0.0001	Convergence
L^2	7.15e-4	2.02e-5	6.31e-7	1.5269
H^1	1.14e-2	1.01e-3	1.00e-4	1.0274

Table 3: 2nd-order elements, log-log triangle area vs error in different norms

The results appear to match the theory, and thus the FOM is certified. It is worth noting that in this case, the presence of the Newton method due to the resolution of a nonlinear problem seems not to alter the convergence orders of the FOM compared to what is expected from linear theory.

3.2. FOM solution of the test case

After having certified the FOM, we proceed computing the high fidelity solution for the test case. A plot of the solution for the parameter value: $\boldsymbol{\mu} = (0.5, 0.6)$, with a mesh size of $Area = 0.001$, with elements of 1st order is shown in figure 1.

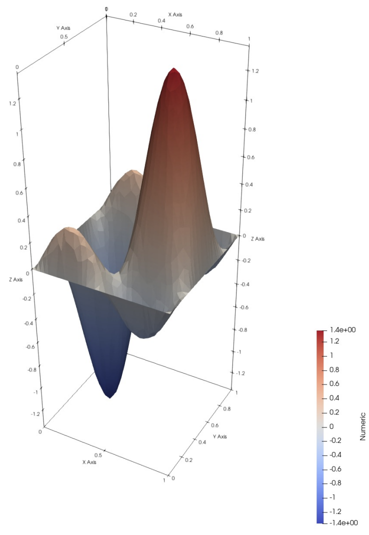
Figure 1: FOM solution for $\boldsymbol{\mu} = (0.5, 0.6)$, with 1st-order Finite Elements

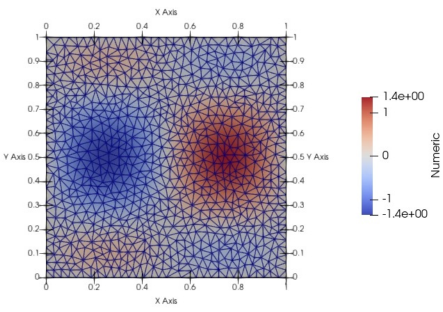
Figure 2: Mesh for FOM

The full order approach ensures convergence, however, it is expensive, so reduced order modeling is required. In the following sections, the previous FOM solution will be taken as a benchmark, and a comparison with the ROM solutions will be made.

4. Reduced Basis Method with POD

Let's now analyze the RBM-POD approach. The idea is to use this technique to solve the linear problem associated with δu at each step of the Newton method. It involves selecting a certain number of parameters $\mathcal{P}_{\mathbf{train}} = \{\boldsymbol{\mu}_i\}_{i=1}^{M}$, then, a set of snapshots (FOM solutions calculated for these parameters) is built:

$$S = \{u_\delta(\boldsymbol{\mu}_1), \cdots, u_\delta(\boldsymbol{\mu}_M)\}$$

Through the POD, a basis is extracted from S to build a reduced space V_N along with the corresponding basis matrix:

$$V_N = span\{\omega_1, \cdots, \omega_N\} \subset V_\delta$$

$$\mathbf{B} = [\boldsymbol{\omega_1}, \cdots, \boldsymbol{\omega_N}] \in \mathbb{R}^{N_\delta x N}$$

where V_δ is the FOM space and N_δ its dimension, and the generic $\boldsymbol{\omega}_i$ is the vector of the components of the ROM basis function ω_i with respect to the V_δ basis.

Finally the linearized FOM problem can be projected into the reduced space, that in algebraic form is:

$$\mathbf{B}^T \mathbf{J}_\delta^f(\mathbf{B}\mathbf{u}_n^N, \boldsymbol{\mu}; \nu)\mathbf{B}[\delta u_N] = -\mathbf{B}^T \mathbf{F}_\delta(\mathbf{B}\mathbf{u}_n^N, \boldsymbol{\mu}; \nu) \tag{5}$$

where: $\mathbf{u}_n^N$ is the vector which contains the components of the solution with respect to the V_N space, at each step n of the Newton method. Thus the LHS is the projection of the discretized Fréchet derivative on the reduced space, and the RHS is the projection of the discretized forcing term of the linearized problem.

Here the main problem of this approach can be observed: since the problem is nonlinear and non-affine, it is not possible to exploit an offline phase for the projection of the matrices: the computation of the full terms on the RHS and LHS has to be made for each variation of the parameter and furthermore for each Newton's step. Thus the computation becomes extremely expensive, and this impacts on the speedup value with respect to the FOM, as we will se in the results section.

5. POD-NN

The first technique presented that makes use of neural networks is the POD-NN. In this case, the idea is to proceed by selecting a certain number of parameters $\mathcal{P}_{\mathbf{train}} = \{\boldsymbol{\mu}_i\}_{i=1}^{M}$ and computing the corresponding snapshots. A basis will be then extracted from this snapshots set, similar to the previous approach:

$$\{\omega_1, \cdots, \omega_N\}$$

this basis will be used to represent the reduced solution. Additionally, the snapshots now serve also another purpose: they are used to train a neural network that will finally provide the reduced model solution. In fact, the NN works as follows:

$$\boldsymbol{\Pi}^{NN} \; : \; \mathcal{P} \longrightarrow \mathbb{R}^N$$

$$\boldsymbol{\mu} \longmapsto \boldsymbol{\Pi}^{NN}(\boldsymbol{\mu}) = \mathbf{u}_N^{NN}(\boldsymbol{\mu})$$

Thus, it maps the parameter $\boldsymbol{\mu}$ to the vector $\boldsymbol{u}_N^{NN}$ that contains the components of the reduced solution written with respect to the basis computed from snapshots:

$$u_N^{NN}(\mathbf{x}; \boldsymbol{\mu}) = \sum_{i=1}^{N} \mathbf{u}_{N,i}^{NN}(\boldsymbol{\mu})\omega_i(\mathbf{x})$$

The neural network is a simple Multi Layer Perceptron (MLP) with 5 hidden layers each made of 10 neurons and uses hyperbolic tangent activations. The weights are optimized with AdamW with an initial learning rate of 10^{-3} and weight decay of $5 \cdot 10^{-2}$. Moreover every 1000 steps the lerning rate is multiplied by 0.95.

5.1. Training phase

As previously mentioned, the neural network has to be trained in a supervised way, and in order to do it, the snapshots are taken as training set:

$$(\boldsymbol{\mu}_i, u_\delta(\boldsymbol{\mu}_i))_{i=1}^{M}$$

Then, a loss function is defined:

$$Loss = \frac{1}{M} \sum_{\boldsymbol{\mu}_i \in \mathcal{P}_{\textbf{train}}} \| \boldsymbol{\Pi}^{NN}(\boldsymbol{\mu}_i) - \mathbf{u}_N(\boldsymbol{\mu}_i) \|^2 \tag{6}$$

where $\mathbf{u}_N(\boldsymbol{\mu}_i)$ is the vector containing the components of the projection of the snapshot $u_\delta(\boldsymbol{\mu}_i)$ on the reduced space V_N: This projection is needed in order to compare the NN output with the target functions (the snapshots). The projector definition rests on the following result [3].

Theorem 1. *there exists an optimal projection $\mathbb{P}$ with respect to the energy norm, that is given by:*

$$\mathbb{P}\mathbf{u}_\delta(\boldsymbol{\mu}) := \mathbf{B}(\mathbf{B}^T \mathbb{X}_\delta \mathbf{B})^{-1} \mathbf{B}^T \mathbb{X}_\delta \mathbf{u}_\delta(\boldsymbol{\mu}) \tag{7}$$

where $\mathbb{X}_\delta$ is the inner product matrix of V_δ.

Now, since u_N is the projection of u_δ on the reduced space V_N, it must hold:

$$\mathbb{P}\mathbf{u}_\delta(\boldsymbol{\mu}) = \mathbf{B}\mathbf{u}_N(\boldsymbol{\mu})$$

By substituting this relation into the one above, we obtain:

$$\mathbf{u}_N(\boldsymbol{\mu}) = (\mathbf{B}^T \mathbb{X}_\delta \mathbf{B})^{-1} \mathbf{B}^T \mathbb{X}_\delta \mathbf{u}_\delta(\boldsymbol{\mu})$$

and from this relation we can get $\boldsymbol{u}_N$, which will be used inside the loss function of the neural network.

In practice, the projection stage is performed by solving not the linear problem just written, but the following one:

$$(\mathbf{B}^T \mathbb{X}_\delta \mathbf{B}) \, \mathbf{u}_N(\boldsymbol{\mu}) = \mathbf{B}^T \mathbb{X}_\delta \mathbf{u}_\delta(\boldsymbol{\mu}) \tag{8}$$

in order to avoid inverting matrices, which is a rather costly operation.

6. Physics Informed Neural Networks

The physics informed neural network approach involves using a neural network which takes as an input both the position in the domain and the parameters: $(\mathbf{x}, \boldsymbol{\mu}) \in \Omega \; x \; \mathcal{P}$ and gives back directly the value $\tilde{u}\,(\mathbf{x}, \boldsymbol{\mu})$ of the predicted solution.

The neural network used is the same one implemented for the POD-NN.

6.1. Training phase

The PINN is trained in an unsupervised way by writing a loss function based on the strong form of the problem. Specifically, we define a residual:

$$R(\,\tilde{u}(\mathbf{x},\boldsymbol{\mu})) = -\Delta\tilde{u}(\mathbf{x},\boldsymbol{\mu}) + \frac{\mu_0}{\mu_1}(e^{\,\mu_1 \tilde{u}(\mathbf{x},\boldsymbol{\mu})} - 1) - g(\mathbf{x}, \boldsymbol{\mu})$$

so the loss function will be given by:

$$L := L_p^{\mu} + L_b^{\mu}$$

where:

$$L_p^{\mu} := \frac{1}{N_p} \sum_{k=1}^{N_p} |\, R(\,\tilde{u}(\,\mathbf{x}_k^p, \boldsymbol{\mu}_k^p)\,|^2 \,, \quad (\,\mathbf{x}_k^p, \boldsymbol{\mu}_k^p) \in \Omega \; x \; \mathcal{P}$$

and

$$L_b^{\mu} := \frac{1}{N_b} \sum_{k=1}^{N_b} |\, \tilde{u}(\,\mathbf{x}_k^b, \boldsymbol{\mu}_k^b)\,|^2 \quad (\,\mathbf{x}_k^b, \boldsymbol{\mu}_k^b) \in \partial\Omega \; x \; \mathcal{P}$$

In our training phase, we set a uniform grid of 2500 points of the domain Ω in order to train the network, while the parameters are chosen with a random uniform distribution over the parametric train set. The weights are optimized in the same way as in the POD-NN network.

7. Results and comparisons

This section analyzes the performance of different reduced order modeling techniques across approximation accuracy and computational efficiency.

7.1. Approximation Accuracy

The accuracy of each ROM solution is assessed computing the relative error with respect to the full order model with the H^1 and L^2. The errors are computed on test parameters $\boldsymbol{\mu}$ not used during the training phase. The effect of increasing the reduced basis dimension on POD and POD-NN approximation error is initially examined. As Figure 3 shows, the H^1 error for the POD solution appears to decrease exponentially as the reduced basis size increases, consistently with the theory. A similar behavior can be observed for the L^2 error. Notice that while the theoretical results specifically pertain to the linearized problem (i.e.computation of δu), experimental evidence suggests that it extends to the solution obtained through the Newton method as well. On the other hand, the POD-NN error does not improve and remains close to the same error level, according to the fact that the basis for the POD-NN is used to represent the solution in the reduced space and build the projector, but this has not impact on the accuracy of the output of the neural network. We also tried to train the network for more steps and on more snapshots, but there was no improvement.

It is worth noting that in our experiments, we were able to capture 99% of the variance of the snapshots set with just one basis of the reduced space, and this can be attributed to the fact that the solution to our problem changes little with the variation

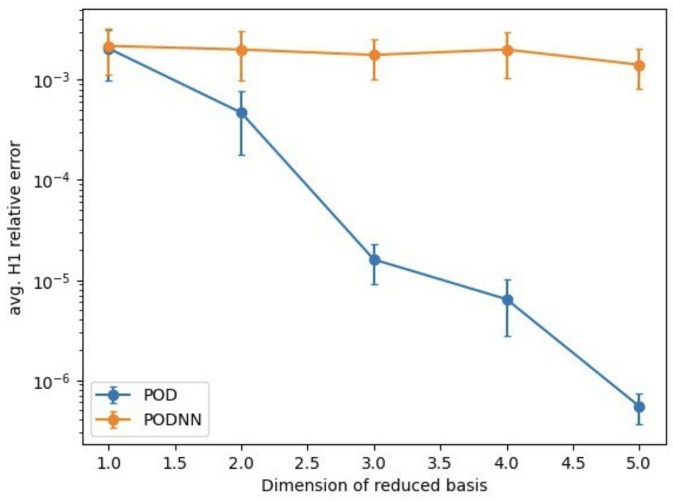

Figure 3: Relative H^1 Error vs. Reduced Basis Dimension

of the parameter in our parameter space. Nevertheless, as observed earlier, the POD error improves as the dimension of the reduced basis increases.

Then, we continue by analyzing how the PINN's L^2 and H^1 errors with respect to the FOM solution evolve during training. In Figure 4, we can clearly see that the network stops improving roughly at 20,000 training steps. Even at its lowest point, the test error is quite high, yet the small gap between the L^2 and H^1 error shows that the PINN approximates correctly the gradient of the FOM solution. We tried increasing the size of the network, the number of training points, changing the learning rate schedule, and adjusting the weights of the boundary and residual loss, but these efforts did not result in further improvements.

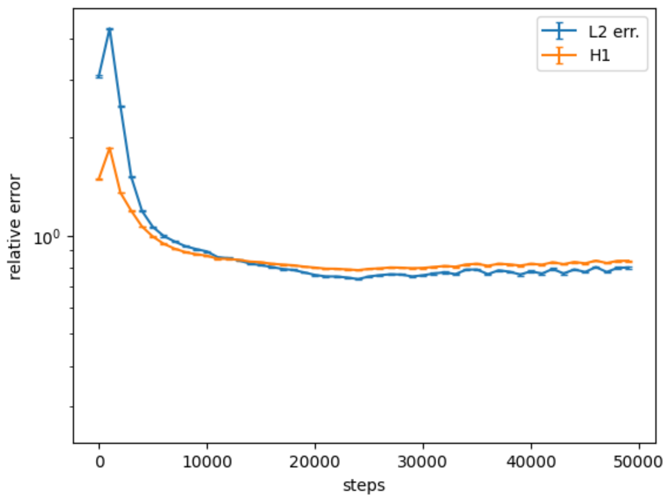

Figure 4: PINN Relative Error during Training

In Figures 5, 6 and 7, the graphs of the differences between FOM and POD, FOM and PODNN, and FOM and PINN are plotted to qualitatively show the errors made by the different reduced solutions compared to the high-fidelity solution.

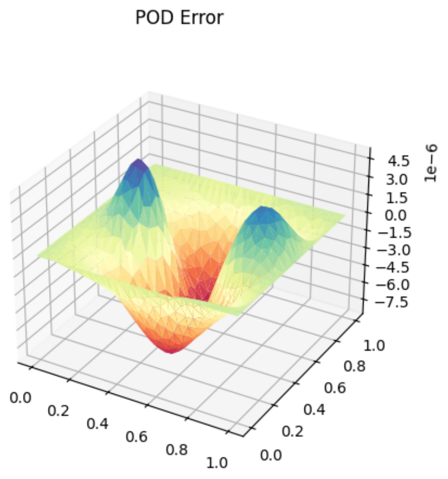

Figure 5: Difference between FOM and POD solution with $\boldsymbol{\mu} = (0.5, 0.6)$

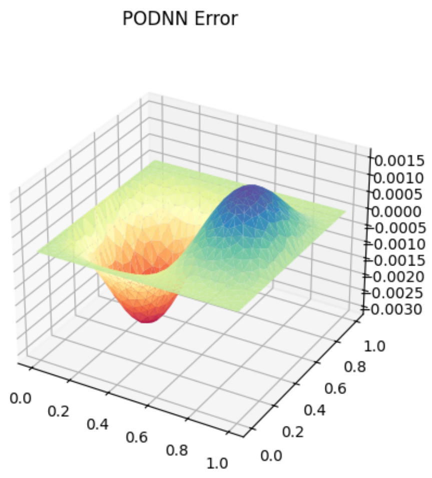

Figure 6: Difference between FOM and PODNN solution with $\boldsymbol{\mu} = (0.5, 0.6)$

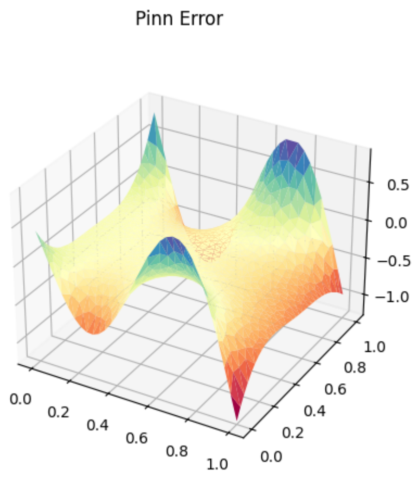

Figure 7: Difference between FOM and PINN solution with $\boldsymbol{\mu} = (0.5, 0.6)$

We observe that the PINN does not match well with the FOM solution at the boundaries, possibly because the loss function is sublinear. In fact, since the difference between the PINN solution and the boundary condition ($g = 0$ for this problem) is less than 1 and is squared in the loss definition, the loss boundary term may not be optimized effectively.

Overall we can state that looking only at the errors the best technique appears to be the POD as it clearly outperforms neural approaches. Now we will see how this consideration is completely overturned if we also account for the execution time of the algorithm.

7.2. *Execution Time*

The primary motivation for using reduced order methods is to accelerate the computation of solutions, even if accuracy is sacrificed to some extent. To assess the speed of the algorithms, being able to include also the PINN in the comparison, we executed all the methods and we compared their computation time with that required by the PINN to be evaluated the spatial points that constitute the degrees of freedom for the Full Order Model of the 1st order and of the 2nd order.

The results are shown in Figures 8 and 9. Since the y-axis is logarithmic, the vertical distance between a ROM point and the Full Order Method (FOM) represents the logarithm of the speedup factor.

In Figure 8, where only a single basis for the reduced space is used, we clearly see that POD does not provide a significant speedup over the FOM, as it grows at a similar rate. Both FOM and POD require computing large matrices at each Newton step, with POD having a slight advantage due to solving a smaller linear system.

On the other hand, PINN is much faster, but its execution time still increases with finer meshes, as the number of evaluation points increases. Additionally, PINN exhibits high variance, possibly due to PyTorch's handling of large inputs.

The POD-NN method is remarkably fast because it processes a single $\boldsymbol{\mu}$ input and outputs the vector of the reduced solution components in the reduced space. The only part of the process affected by the number of degrees of freedom is the matrix multiplication needed to write the reduced solution with respect to the FOM space base.

In Figure 9, with a basis dimension of 10, we observe an immediate sharp increase in execution time for POD and POD-NN. A possible explanation is that these two methods involve projections between lower and higher dimensional spaces, which requires progressively larger matrices. When the matrices become too large to fit in the processor cache, parts must be read from RAM, adding computational overhead and reducing the effective speedup over the FOM. The PINN is not affected by this issue as it does not require any representation with respect to a basis, so the matrix multiplications are avoided.

In the table 4, the H^1 error and the speedups of the three methods are reported by way of example, using a mesh triangle's area of 10^{-3}, 1st-order elements for the FOM space and 4 basis elements for the reduced space.

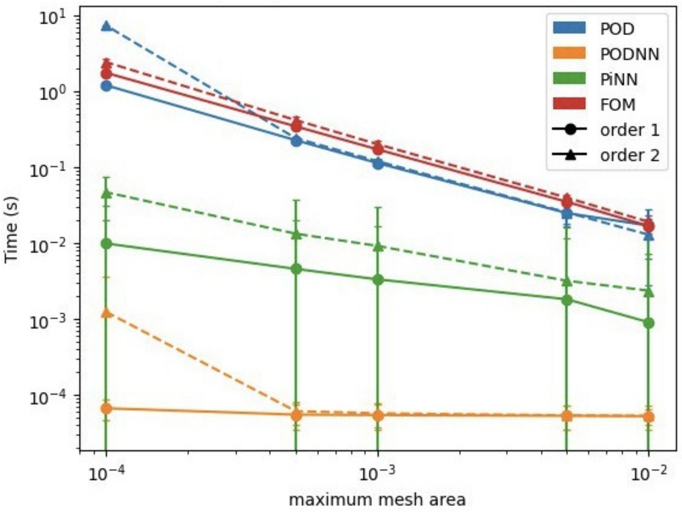

Figure 8: Execution times with basis of dimension 1

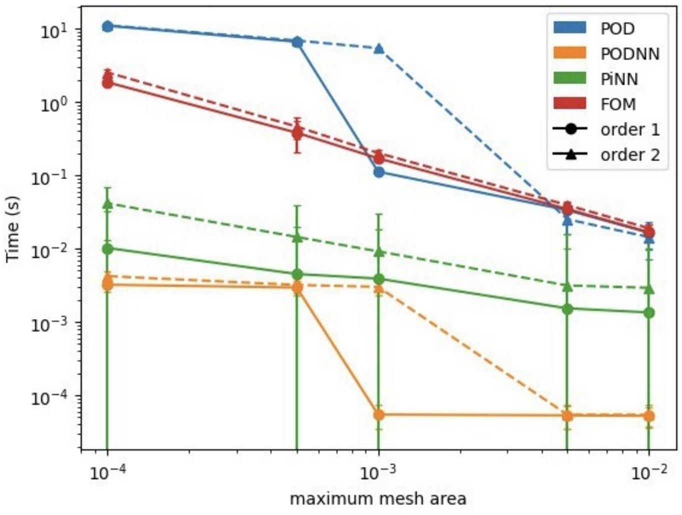

Figure 9: Execution times with basis of dimension 10

Model	H1 Rel. Error	Speedup
POD	**7.84e-06 ± 6.25e-06**	1.0 ± 0.004
POD-NN	4.07e-03 ± 2.23e-03	**2e+03 ± 3.2e+02**
PINN	6.71e-01 ± 9.71e-04	5.8e+02 ± 2.8e+02

Table 4: speedups and H1 errors

8. Conclusions

We performed experiments to compare three different MOR methods: RMB-POD, POD-NN and PINN. The first method appears to be an effective approach only in linear problems when the affine hypothesis holds, otherwise it is too expensive and does not provide any speedup improvement with respect to the Full Order Model. The PINN turns out to be too inaccurate, and the error is difficult to reduce beyond a certain threshold.

Overall, it can be said that the best compromise between accuracy and speed is perhaps the POD-NN, because on one hand, it is much faster than the classic RBM-POD thanks to the use of a neural network, and on the other hand, it is more accurate than the PINN due to the supervised training, and because its output consists of the solution components written with respect to a basis that preserves the behavior of the high-fidelity solutions since it is extracted from the snapshots set.

References

[1] F. Vicini, Gedim github repository, `https://github.com/fvicini/gedim.git`.

[2] L. Evans, Partial Differential Equations, Graduate studies in mathematics, American Mathematical Society, 2010.
URL `https://books.google.it/books?id=Xnu0o_EJrCQC`

[3] A. Quarteroni, A. Manzoni, F. Negri, Reduced Basis Methods for Partial Differential Equations: An Introduction, UNITEXT, Springer International Publishing, 2015.
URL `https://books.google.it/books?id=e6FnCgAAQBAJ`

[4] L. Tredese, L. Biscu, Github repository, `https://github.com/LeonardoTredese/ModelOrderReduction.git`.

[5] A. Quarteroni, Numerical Models for Differential Problems, 2nd Edition, Springer Publishing Company, Incorporated, 2013.

[6] J. S. Hesthaven, G. Rozza, B. Stamm, Certified Reduced Basis Methods for Parametrized Partial Differential Equations, 1st Edition, Springer Briefs in Mathematics, Springer, Switzerland, 2015. `doi:10.1007/978-3-319-22470-1`.

[7] H. Brezis, Functional Analysis, Sobolev Spaces and Partial Differential Equations, Universitext, Springer New York, 2010.
URL `https://books.google.it/books?id=GAA2XqOIIGoC`