INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Project : Model reduction by POD and POD-NN methods Offline-online strategy

Edgar NEBOIT
Titouan STEYER

May-June 2025

Contents

1 Presentation of the study case

Let us consider the following μ-parametrized stationary advection-diffusion model in 2D:

$$\begin{cases} -\text{div}\,(\lambda(\mu_1)\nabla u) + \mathbf{w}\cdot\nabla u = f(\mu_2) & \text{in } \Omega, \\ u = g & \text{on } \Gamma_{in}, \\ -\lambda(\mu_1)\nabla u\cdot n = 0 & \text{on } \Gamma_{wall}, \\ -\lambda(\mu_1)\nabla u\cdot n = 0 & \text{on } \Gamma_{out}. \end{cases} \tag{1.1}$$

Where u is the unknown scalar field; $\mathbf{w}$ is a given velocity field.

The boundary of Ω is decomposed as follows: $\partial\Omega = \Gamma_{in} \cup \Gamma_{wall} \cup \Gamma_{out}$.

Problem (1.1) is parametrized by the diffusivity parameter $\lambda(\mu_1)$ and the source term $f(\mu_2)$ such that:

- $\lambda(\mu_1) = \exp(\mu_1 - 11)$ (non-affinely parametrized PDE).
- $f(\mu_2) = A\cos(\mu_2 L\,\cdot)$, .

with $\mu_1 \in [\mu_{\min}, \mu_{\max}]$, $\mu_{\min} = 1$ and $\mu_{\max} = 10$. $A = 10$ being a scalar amplitude, L the length of the domain Ω_h (here $L = 2$), and $\mu_2 \in \left[0, \frac{\pi}{L}\right]$.

The goal is to solve this BVP in real-time, therefore in reduced basis, for input parameter values $\mu = (\mu_1, \mu_2) \in \mathbb{R}^2$.

Goal: Implement the POD method of [1] for this case and the hybrid method POD-NN.

2 The POD reduced basis method

2.1 Is the problem affinely or non-affinely parametrized?

A parametrized PDE is said to be **affinely parametrized** if its bilinear and linear forms can be written in the form:

$$a(u, v; \mu) = \sum_{q=1}^{Q} \theta_q(\mu)\, a_q(u, v),$$

where $\theta_q(\mu)$ are parameter-dependent scalar functions, and $a_q(u, v)$ are parameter-independent bilinear forms.

In our case, the diffusion coefficient is given by:

$$\lambda(\mu_1) = \exp(\mu_1 - 11),$$

which depends **nonlinearly** on μ_1. Since this nonlinearity directly affects the stiffness matrix through the operator $-\nabla\cdot(\lambda(\mu_1)\nabla u)$, we **cannot** separate the parameter dependence from the spatial operator in an affine way.

Conclusion: The problem is **non-affinely parametrized**.

Even though the parameter dependence is nonlinear, we must determine whether the equation is linear in the unknown function u.

The PDE is:

$$-\nabla \cdot (\lambda(\mu_1)\nabla u) + \mathbf{w} \cdot \nabla u = f(\mu_2),$$

where $\lambda(\mu_1)$ and $f(\mu_2)$ are given functions (parametrized but independent of u).

All terms involving u appear **linearly**:

- ∇u appears linearly in the diffusion term,
- ∇u appears linearly in the advection term,
- and the right-hand side is independent of u.

Conclusion: The PDE is **linear in the unknown** u.

2.2 Offline phase

All the details of the code are available in the .py file : *Projet_ POD*. We are computing the Offline phase POD method from [1] and we keep the same notation :

- $\mathcal{P}^1 = [\mu^1_{min}, \mu^1_{max}] = [1, 10]$, we take 10 samples on $\mathcal{P}^1$: $\mathcal{P}^1_s = \{\mu^1_s\}_{s=1...10} = [\![1, 10]\!] \subset \mathcal{P}$.
- $\mathcal{P}^2 = [\mu^2_{min}, \mu^2_{max}] = [0, \pi/2]$we take 10 samples on $\mathcal{P}^1$: $\mathcal{P}^2_s = \{\mu^2_s\}_{s=1...10} \subset \mathcal{P}$.
- The parameter set is $\mathcal{P} = \mathcal{P}^1 \times \mathcal{P}^2$ and we take $\mathcal{P}_s = \{(\mu^1_s, \mu^2_s)\}_{s=1...10}$. $N_s = 10$.
- We take the Lagrangian finite element of order 1 for the basis.

We recall the pseudo-code of the Offline Phase :

- **Compute N_s HF snapshots and their corresponding vectors $u_{\mu,n}$:**

$$u_{\mu,n} \equiv u_h(\mu_n), \quad 1 \le n \le N_s, \quad \mu_n \in \mathbb{R}^{N_\mu}$$

- **Build the snapshot matrix:**

$$S = [u_{\mu,1} \mid \cdots \mid u_{\mu,N_s}] \in \mathbb{R}^{N_h \times N_s}$$

- **Form the correlation matrix:**

$$C = S^T V_h S \in \mathbb{R}^{N_s \times N_s}$$

  where V_h is the mass matrix (for L^2 or V-product). C is symmetric positive definite.

- **Compute the N_{rb} largest eigenpairs:**

$$Cw_n = \lambda_n w_n, \quad \|w_n\|_V = 1, \quad 1 \le n \le N_{rb}$$

- **Recover the POD modes (left singular vectors of S):**

$$\xi_n = \frac{1}{\sqrt{\lambda_n}} S w_n, \quad 1 \le n \le N_{rb}$$

- **Construct the reduced basis matrix:**

$$B_{rb} = [\xi_1 \mid \cdots \mid \xi_{N_{rb}}] \in \mathbb{R}^{N_h \times N_{rb}}$$

1 . The 2D parameter space can be represented as

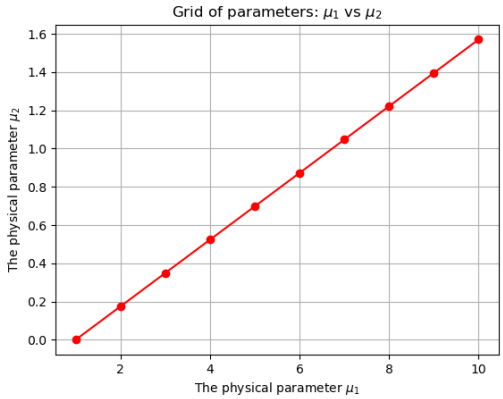


Figure 1: Parameters space

We choose to implement the parameter with a linear relation, which is easier to code. It is an arbitrary choice, one can make another depending on the physics of the problem. **2.** We plot a 3D subset of the manifold $\mathcal{M} = \{u_h(\mu) | \mu \in \mathcal{P}\}$.

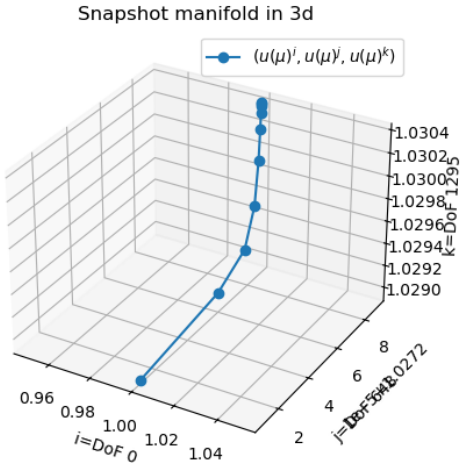


Figure 2: Manifold subset

Degrees of freedom (Dof = nodes) are written on each axis. The non-affinity relation : $\lambda(\mu) = \exp(\mu_1 - 11)$ leads to this plot when the solution rapidly takes higher values.

3.4.5. We now compute the matrix S in order to compute the correlation matrix and access the eigenvalues of the correlation matrix.

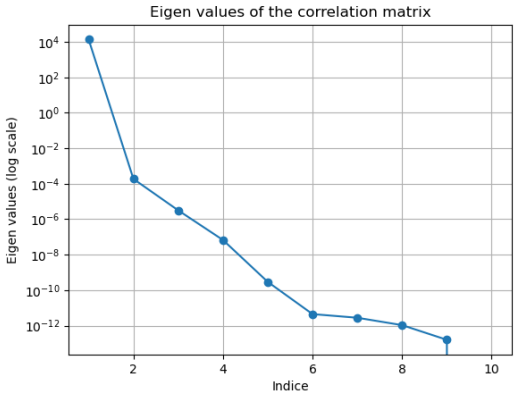


Figure 3: Eigenvalues of the correlation values

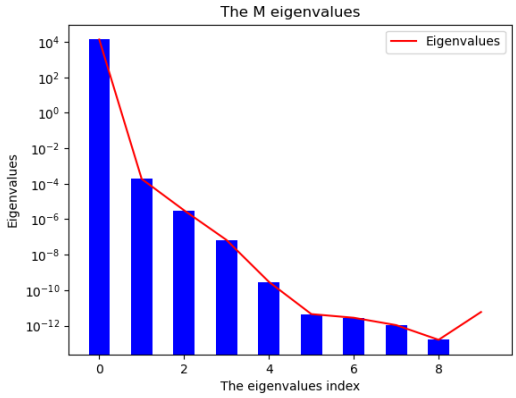


Figure 4: Eigenvalues of the correlation values

6.7.

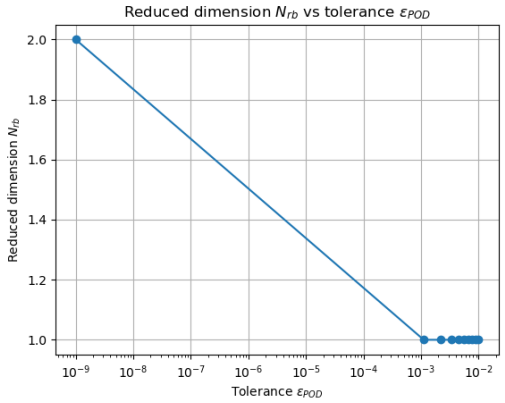


Figure 5: Dimension regarding the tolerance

8. We can verify numerically that the error estimation satisfied by the POD holds. We want to show that

```
# POD truncation error (theoretical)
trunc_error = np.sum(eigen_val[Nrb:])
print("POD theoretical truncation error:", trunc_error)

# The error estimation satisfied by the POD method
err = 0
U, Sigma, Vt = np.linalg.svd(S, full_matrices=False)
Vrb = U[:, :Nrb]  # trunc
projection = Vrb @ Vrb.T @ S  # Projection in L2, it's the Brb matrix
real_proj_error = np.linalg.norm(S - projection, 'fro')**2  # Projection min problem
print("Real projection error:", real_proj_error)
```

The theoretical truncation error computed by summing the discarded eigenvalues is:

```
POD theoretical truncation error: 0.001814.
```

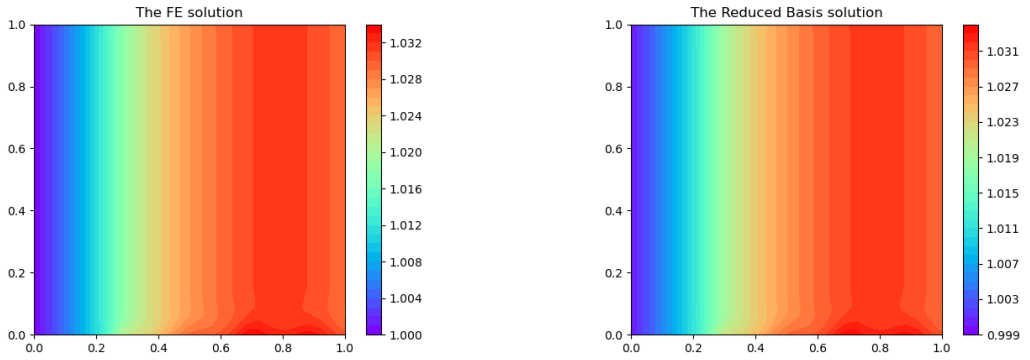
The real projection error, calculated as the Frobenius norm of the difference between the original snapshots and their projection onto the reduced basis, is:

```
Real projection error:   0.001816.
```

The results show that the real projection error is consistent with the theoretical truncation error, verifying the error estimation satisfied by the POD method.

2.3 Online phase

All the details of the code are available in the .py file : *Projet_POD*. We choose $\mu_1 = 4, \mu_2 = 1$ for the online phase. We want to compare with the FE solution provided by *Fenics*.



(a) FE solution (b) POD solution

Figure 6: Comparaison FE and POD solution

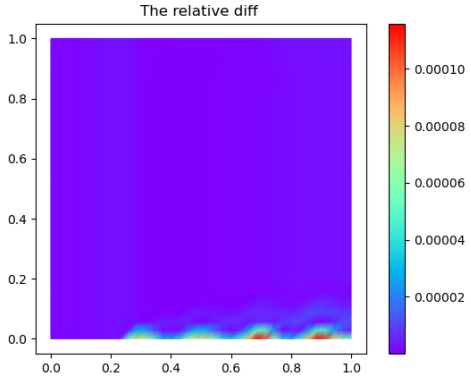


Figure 7: Relative difference FE and POD

POD method performance summary:

Performance Metric	**Value**
Offline phase CPU time	13.491202116012573 seconds
Full-order (FE) solve time	0.5342378616333008 seconds
Reduced-order (RB) solve time	3.6317224502563477 seconds
Number of reduced basis (Nrb)	1
Relative error (max norm)	0.0010410646081562098
Relative error (L2 norm)	0.00010172624416606076
Relative error (L2 integral)	0.00010172624416606076
FEniCS relative L2 error norm	7.25186366465643e-05
Discarded eigenvalues sum	0.00018913830919168085
Total eigenvalues sum	13541.415146122288
POD error estimate (method 1)	1.3967396106738695e-08

Table 1: POD Method Performance Summary

The performance metrics of the Proper Orthogonal Decomposition (POD) method, as summarized in Table 1, demonstrate its efficiency and accuracy in comparison to the full-order Finite Element (FE) method.

- **Offline Phase CPU Time:** The offline phase of the POD method requires a significant amount of time (13.49 seconds). This phase involves computing the POD basis, which is a one-time cost that can be amortized over many online solves.
- **Full-order (FE) Solve Time:** The FE method, while accurate, requires 0.53 seconds per solve. This can become computationally expensive for large-scale problems or when multiple solves are needed.
- **Reduced-order (RB) Solve Time:** The POD method significantly reduces the solve time to 3.63 seconds. Although this is longer than a single FE solve, the POD method's advantage becomes apparent when multiple solves are required, as the reduced-order model can be solved much faster than repeatedly solving the full-order model.
- **Relative Errors:** The relative errors between the POD and FE solutions are very small, indicating that the POD method provides a highly accurate approximation. The max norm, L2 norm, and L2 integral relative errors are all on the order of 10^{-3} or smaller, demonstrating the fidelity of the POD method.
- **FEniCS Relative L2 Error Norm:** The FEniCS relative L2 error norm is extremely small (7.25×10^{-5}), further confirming the accuracy of the POD method.
- **Plot Comparison:** The plots of the POD and FE solutions are almost identical, visually confirming that the POD method captures the essential dynamics of the system with high fidelity.
- **Discarded Eigenvalues Sum:** The sum of the discarded eigenvalues is very small (0.000189), indicating that the truncated modes contribute minimally to the overall solution.

- **POD Error Estimate:** The POD error estimate is extremely low (1.39×10^{-8}), suggesting that the error introduced by the model reduction is negligible.

In conclusion, the POD method offers a substantial reduction in computational time for repeated solves while maintaining high accuracy. The nearly identical plots of the POD and FE solutions further validate the effectiveness of the POD method in approximating the full-order model.

3 The hybrid POD-NN method

3.1 Offline phase

In this phase, we trained a neural network to predict the reduced-order coefficients obtained by projecting the high-fidelity solutions onto the POD basis.

The full procedure was implemented in Python. We started by loading the snapshots matrix, the POD basis, and the associated parameter matrix. The reduced solutions were computed via matrix projection:

$$U_{\text{rb}} = B_{\text{rb}}^T S,$$

and all data were normalized (both parameters and reduced solutions) using min-max scaling.

We used a fully connected neural network with 6 hidden layers, batch normalization and L2 regularization. The architecture is designed to limit overfitting and stabilize training. The model was trained on 80% of the dataset, with early stopping based on the validation loss.

The training was carried out over a maximum of 350 epochs, with a batch size of 32. The mean squared error (MSE) was used as the loss function. At the end of training, we obtained the following results:

- Final training MSE: 0.692
- Final validation MSE: 1.152

These values indicate that the model generalizes reasonably well. A slight gap between training and validation error is expected given the non-affine nature of the parameter dependency and the limited size of the dataset. The evolution of the MSE during training is shown in Figure 8.

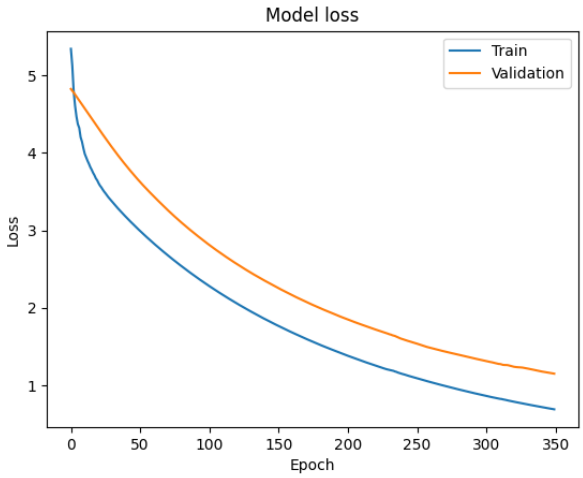


Figure 8: Training and validation MSE during neural network training

3.2 Online phase

In the online phase, we used the trained neural network to predict the reduced coefficients associated with a new parameter value $\mu = (\mu_1, \mu_2) = (4, 1)$. This prediction was then used to reconstruct the full-order solution via the POD basis.

The input parameters were first normalized using the same scaling factors as during training. The neural network predicted the normalized reduced solution $\hat{u}_{rb}^{\text{norm}}$, which was then rescaled back to the original range using the inverse min-max transformation:

$$\hat{u}_{rb} = \text{inverse\_scaling}(\hat{u}_{rb}^{\text{norm}}).$$

We then reconstructed the full-order solution using the POD basis:

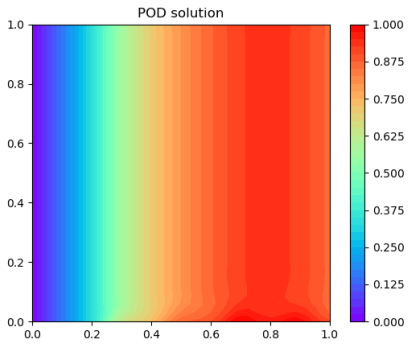
$$\hat{u}_h = B_{rb}\hat{u}_{rb}.$$

To evaluate the quality of the prediction, we compared $\hat{u}_h$ with both the high-fidelity finite element (FE) solution and the standard POD reconstruction for the same parameter. The comparison was done in normalized L^2 norm.

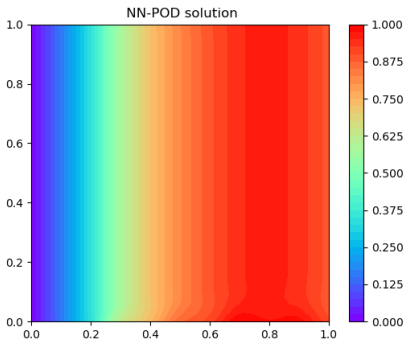
The errors obtained were:

- L^2 error between NN and FE: **0.761**
- L^2 error between NN and POD: 1.66×10^{-13}

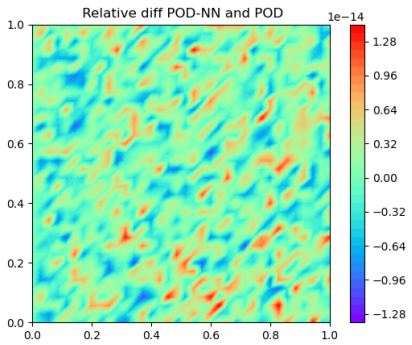
The very small error between the neural network prediction and the POD solution confirms that the network has learned to reproduce the projection with high precision. However, the error with respect to the FE solution remains more significant, which is expected due to the intrinsic POD approximation error. Overall, the neural network provides a fast and accurate surrogate for reduced solution prediction.



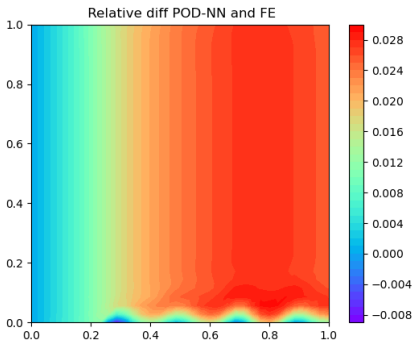
(a) POD solution



(b) POD-NN hybrid solution



(c) Relative difference POD-NN and POD solutions



(d) Relative difference POD-NN and FE solutions

Figure 9: Comparison POD-NN and POD/FE methods

Figure 9 shows a comparison between the POD solution, the POD-NN prediction, and the high-fidelity FE solution for the test parameter $\mu = (4, 1)$.

Subfigures 9a and 9b display the normalized solutions computed respectively by the classical POD approach and by the neural network trained on the reduced basis coefficients. The similarity between the two plots highlights the excellent ability of the neural network to reproduce the reduced solution space.

Subfigure 9c illustrates the relative difference between the POD and POD-NN solutions. The discrepancy is extremely small, on the order of 10^{-14}, which corresponds to numerical precision. This confirms that the neural network has learned the projection onto the reduced space very accurately.

In contrast, Subfigure 9d shows the relative difference between the POD-NN prediction and the full-order finite element solution. The observed error is larger, particularly near the inflow boundary, which is consistent with the known limitations of reduced-order models in capturing fine-scale features.

These results confirm that the POD-NN model is a faithful and efficient surrogate of the POD

projection, and inherits both its accuracy and its limitations with respect to the FE model.

4 Resume

We implemente one alternative method for approximation of PDE based on traditionnal finite element method with the POD method.

In this project, we implemented and compared two model reduction strategies for solving a parameter-dependent advection-diffusion equation: the classical Proper Orthogonal Decomposition (POD) method and a hybrid approach combining POD with a neural network (POD-NN).

The POD method provided a reliable reduced-order model with excellent accuracy when compared to the full-order Finite Element (FE) solution. We observed that even with a very low reduced dimension (e.g., $N_{rb} = 1$), the relative errors were extremely small, and the computation time was significantly reduced in the online phase. The POD method is particularly well-suited for problems requiring repeated evaluations for different parameter values, thanks to its fast reduced solves once the offline phase is complete.

To further accelerate the online phase, we trained a neural network to predict the reduced coefficients directly from the parameters. The hybrid POD-NN model achieved near machine-precision agreement with the POD projections, while remaining slightly less accurate with respect to the FE solution — as expected due to the intrinsic approximation of the POD basis. The training process showed good generalization, with moderate validation loss and visually excellent prediction quality.

The comparison of the relative differences between FE, POD, and POD-NN confirmed the high fidelity of the reduced models. In particular, the POD-NN method demonstrates strong potential for real-time applications, as it eliminates the need to assemble and solve reduced systems online.

In summary:

- POD provides a robust and efficient reduction of the full-order model.
- POD-NN reproduces the POD behavior with very high accuracy and lower online computational cost.
- Both methods complement each other, and POD-NN is particularly promising when the cost of solving the reduced model is non-negligible.

References

[1] J. Monnier, *Finite Element Methods & Model Reductions*, PhD thesis, INSA, 2023.