Scientific Machine Learning for Model Order Reduction of Large Scale Problems

Final Report

Honghao Wang

Supervisor: Garth N. Wells, Nirav Vasant Shah

September 3, 2024

1 Introduction

Solving partial differential equations (PDEs) underpins critical scientific advancements across diverse fields. Nowadays, enhanced computational capabilities combined with improved algorithms allow for high-fidelity numerical solutions to intricate problems using standard discretisation methods. For example, classical methods such as the finite element method (FEM) are used to solve the governing PDE. In this project, we are interested in solving the high-fidelity FEM model repeatedly. However, this could pose very high stress on the limited computational resources because solving high-fidelity models for various parameters usually involves computing many degrees of freedom for the solution. Hence, using the full-order model usually requires access to high-performance computing systems, which can be expensive.

To address the issue, model order reduction (MOR) helps to alleviate the computational burden by identifying a lower dimensional representation of the parametric dependence of the high-fidelity model. The basic idea on which MOR is based is related to the fact that often the parametric dependence of the problem at hand has an intrinsic dimension much lower than the number of degrees of freedom associated to the governing high-fidelity solution [**?**]. There are two stages in an MOR procedure. The first one is an offline stage. During this stage, a large number of solutions (solved under different initialisations of parameters) are collected together to form a solution snapshot matrix. The MOR algorithm can then be run on the snapshot matrix, which identifies the most significant modes that form the reduced basis space. Stage two is an online stage where information regarding the reduced basis space is fully utilised, which allows for the computing of the projections of new high-fidelity solutions onto the lower intrinsic dimensions.

We aim to develop a framework for data-driven MOR by combining Proper Orthogonal Decomposition (POD) with Artificial Neural Networks (ANN)[**?**]. The method involves identifying reduced basis space by performing POD, training the ANN with snapshots of PDE solutions as inputs, and using the trained ANN for a quick approximation of the solution field

of a PDE. While offering data-driven efficiency and reducing reliance on high-fidelity FEM models, the trade-offs include a sacrifice in accuracy and extended training times for the ANN. The loss in accuracy was particularly evident in the vector field. Achieving a balance between accuracy and efficiency is crucial for the successful implementation of this approach.

To accelerate the runtime of the programme, this project leverages supercomputers through data-parallel computing, which divides tasks into smaller sub-tasks, allowing multiple processors to execute computations simultaneously. Both CPU and GPU-based parallelisation were utilised in the project. The CPU-based parallelisation involves distributing the computational workload across multiple CPU cores to accelerate the generation of data. This involves solving large-scale PDEs with the FEM library FEniCSx [**?**], specifically for computing the FEM solutions for calculating the reduced basis functions in POD and for obtaining the ANN training data; In the context of ANN training, the GPU-based parallelisation was utilised with the help of TensorFlow. Partitions of the dataset are distributed to individual processors, each handling its subset of the dataset. Through distributed ANN training, parallel processing accelerates the learning process. The outputs during each training epoch (e.g. losses and gradients) are synchronised among processors.

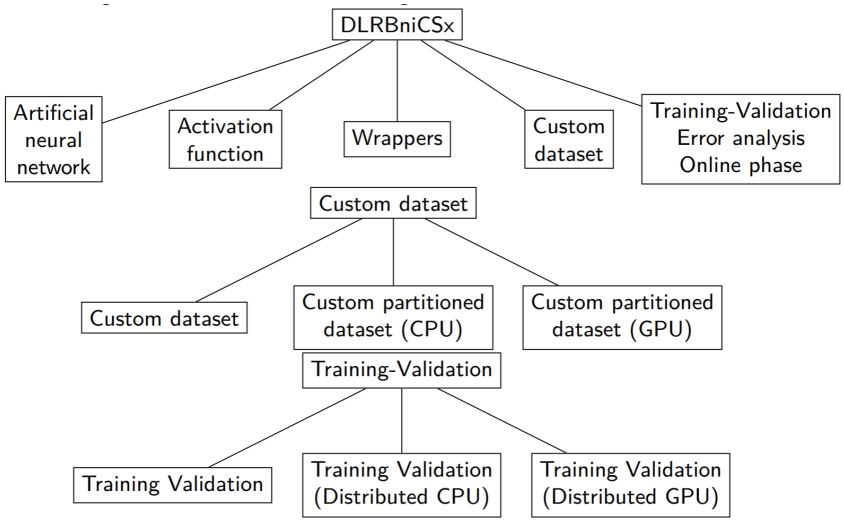


Figure 1: Project outline

Notes for Nirav: could you please correct any misunderstandings I may have about the full picture of DLRBniCSx in the below paragraph? I was not so sure when drafting this paragraph, thank you. The entire process above is currently an ongoing work of the DLRBniCSx python package [**?**], where an ANN-POD model has been developed for some types of PDE such as the linear/non-linear Poisson equations, the Navier-Stokes equations etc. DLRBniCSx is an open-source library currently built on PyTorch (for building and training ANN), RBniCSx [**?**](for performing MOR), and FEniCSx (for solving PDEs using FEM), designed for deep learning-based reduced-order modelling. It offers a range of functionalities such as building the ANN, setting the activation functions, wrappers, and custom dataset management. The custom dataset feature encompasses both CPU and GPU partitioned datasets tailored for both Proper Orthogonal Decomposition (POD) and neural network training. Additionally, the library provides tools for training-validation error analysis. The whole picture

of DLRBniCSx is shown in figure 1.

To enhance the usability of DLRBniCSx, a significant part of the project is dedicated to offering additional Tensorflow [**?**] support to DLRBniCSx, and to designing a simple user interface based on Tensorflow. In addition, this project also focuses on the implementation of the mixed formulation for the Poisson equation, which included an extra vector term σ that represented the flux of the scalar field in the Poisson equation. The parametrisation of PDEs incorporated changing variables in the governing equations, allowing for more flexibility in the system. In this report, we also explain the choices of the parameters through the MOR results and the range of parameters through the ANN results.

Below is a summary of the workflow of this project, which will be discussed in greater depth in this report. In Section 2, we will introduce the governing equation of the mixed formulation for the Poisson equation and how the traditional finite element method is used to solve for the full-order solution. Section 3 focuses on the concept of parametrised PDEs and explain our choices of parameters focused on the project. In Section 4, the idea of MOR is explained in detail, and we will discuss the result and the effectiveness of MOR in reducing dimensionality, along with a more detailed explanation of the choice for parameterisation. The use of ANN will then be covered in Section 5, followed by a presentation of the numerical results in Section 6. Lastly, the concept of parallelisation is covered in Section 7.

2 Governing Equation and Finite Element Methods

The Finite Element Method(FEM) is a numerical method for approximating the solution of the governing PDE by discretising the problem domain into smaller elements. Over the discretised domain, the solution is approximated using piecewise polynomial functions. The weak formulation converts the PDE into an equivalent variational form. The general method of weak formulation involves multiplying the PDE by a test function v and integrating it over the domain Ω, which would allow us to perform integration by parts on those terms with second-order derivatives, simplifying the solution process [**?**].

Work has been done on some PDEs such as the linear/non-linear Poisson equations, the Navier-Stokes equations etc. In this project, the problem of interest is the mixed formulation for the Poisson equation, which introduces an additional vector variable to represent the gradient of the scalar unknown, namely the (negative) flux, leading to the new PDE:

$$\vec{\sigma} - \nabla u = 0 \quad , \text{in } \Omega \tag{1}$$
$$\nabla \cdot \vec{\sigma} = -f \quad , \text{in } \Omega \tag{2}$$

Subject to the Neumann and Dirichlet boundary conditions:

$$\vec{\sigma} \cdot \vec{n} = g \quad , \text{on } \Gamma_N \tag{3}$$
$$u = u_0 \quad , \text{on } \Gamma_D \tag{4}$$

Here u and $\vec{\sigma}$ are the scalar and the flux terms, and $\vec{n}$ is the outward pointing normal vector on the boundary.

Since FEniCSx is implemented based on the FEM, the first step is to convert the PDE into its variational form. Multiplying Equation 1 and 2 by test functions $\vec{\tau}$ and v, and integrating the gradient term by parts in the first equation, the following variational form is obtained:

$$\int_\Omega (\vec{\sigma} \cdot \vec{\tau} + \nabla \cdot \vec{\tau}\, u)\, d\Omega = \int_\Gamma \vec{\tau} \cdot \vec{n}\, u\, ds \quad \forall \vec{\tau} \in \Sigma \tag{5}$$

$$\int_\Omega \nabla \cdot \vec{\sigma}\, v\, d\Omega = -\int_\Omega f\, v\, d\Omega \quad \forall v \in V \tag{6}$$

Inserting the boundary conditions from equation 3 and 4, we can obtain the following equation to be solved for $(\vec{\sigma}, u) \in \Sigma_g \times V$ [**?**]:

$$a((\vec{\sigma}, u), (\vec{\tau}, v)) = L((\vec{\tau}, v)) \quad \forall (\vec{\tau}, v) \in \Sigma_0 \times V \tag{7}$$

$$a((\vec{\sigma}, u), (\vec{\tau}, v)) = \int_\Omega \vec{\sigma} \cdot \vec{\tau} + \nabla \cdot \vec{\tau}\, u + \nabla \cdot \vec{\sigma}\, v dx \tag{8}$$

$$L((\vec{\tau}, v)) = -\int_\Omega f v\, dx + \int_{\Gamma_D} u_0 \vec{\tau} \cdot \vec{n} ds \tag{9}$$

Where $\Sigma_g = \{\vec{\tau} \in H(\text{div}) \text{ such that } \vec{\tau} \cdot \vec{n}|_{\Gamma_N} = g\}$ and $V = L^2(\Omega)$ [**?**]. To discretise the formulation in equation 7 for FEM to work properly and stably, we used the Brezzi-Douglas-Fortin-Marini function space with a polynomial degree of k to represent Σ, and

the Discontinuous Galerkin (DG) space with a degree of $k-1$ to represent V. The BDMCF space is defined on each mesh element and is designed to provide accurate approximations for vector quantities. The DG space allows discontinuities in the solution between neighbouring elements, making it particularly useful for problems with rapid changes or sharp gradients. Listing 1 shows the example codes for defining the geometric function space of a mixed-formulation problem. In particular, we have set up a function space **V**, which is created by mixing the vector and scalar elements. The vector component (flux) **Q_el** rests on a BDMCF element, and the scalar **P_el** rests on DG space.

Listing 1: Defining geometric parameters / Creating function space

```
Q_el = basix.ufl.element("BDMCF", domain.basix_cell(), k)
P_el = basix.ufl.element("DG", domain.basix_cell(), k - 1)
V_el = basix.ufl.mixed_element([Q_el, P_el])
V = fem.functionspace(domain, V_el)
```

The problem is defined and solved with code listing 2, where a LinearProblem object that brings everything together is created. The weak forms a and L define the PDE, *bcs* includes the functions defining the boundary conditions. Special attention is drawn to the *petsc_options*, which specifies that the LU decomposition is used as a linear solver. The choice of solvers significantly impacts the efficiency and accuracy of solving linear systems in finite element simulations. Direct solvers compute the solution to a linear system of equations by performing a series of arithmetic operations to decompose the coefficient matrix into lower and upper triangular matrices (LU decomposition) and then solve the resulting triangular systems. They can provide accurate solutions for smaller systems and are more useful for dense matrix (which is the case in solving PDEs with FEM), but may become computationally expensive as the problem size increases. Iterative solvers are also commonly found in linear problems, examples include the Conjugate Gradient (CG) or Generalized Minimal Residual (GMRES). They are suitable for large, sparse linear systems, but they are usually more complicated and require pre-conditioning to improve convergence [**?**].

Listing 2: Defining and solving the porblem

```
problem = fem.petsc.LinearProblem(a, L, bcs=bcs,
                                  petsc_options={"ksp_type":"preonly","pc_type":"lu"})
w_h = problem.solve()   # solution manifold containing u and sigma
```

TODO: Model discussion on mesh choices? size of discretisation? Function spaces

3 Parametrisation of PDEs

Parametrised PDEs find applications across numerous scientific and engineering domains, including fluid dynamics, structural mechanics, electromagnetics, and more. They provide a powerful framework for modelling complex phenomena where multiple factors influence the system's behaviour. Parameterised PDEs introduce additional parameters $\boldsymbol{\mu}$ into the equation, allowing for a more flexible representation of physical systems. These parameters can represent various factors such as material properties, boundary conditions, or geometric features. By incorporating parameters, the PDEs become functions of these variables, enabling the analysis of how changes in parameters affect the system's behaviour.

The materials parameters $\boldsymbol{\mu}_m$ intuitively refer to the varying material properties in the problem. The boundary conditions parameters $\boldsymbol{\mu}_b$ refer to variables that influence the boundary conditions imposed on the problem. The geometric parameter $\boldsymbol{\mu}_g$ refers to a variable that represents the geometric features or configurations of the system being modelled. These parameters capture aspects such as the shape, size, orientation, or position of geometric entities within the domain of the PDE.

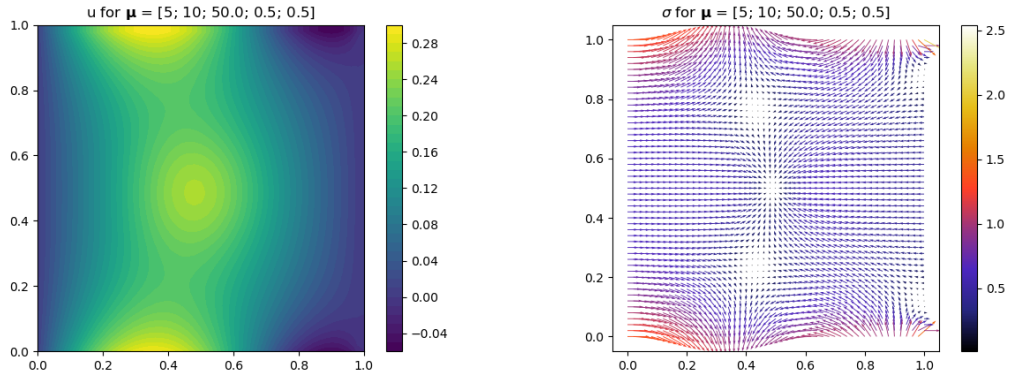
For demonstration purposes, this project focuses on working on a fixed geometric domain of a 1 by 1 grid. The mixed formulation problem is defined with the following conditions:

$$u = 0 \quad \text{on } \Gamma_D = \{(0, y) \cup (1, y) \in \partial\Omega\} \tag{10}$$

$$\text{Flux: } g = \sin(\mu_1 x) \quad \text{on } \Gamma_N = \{(x, 0) \cup (x, 1) \in \partial\Omega\} \tag{11}$$

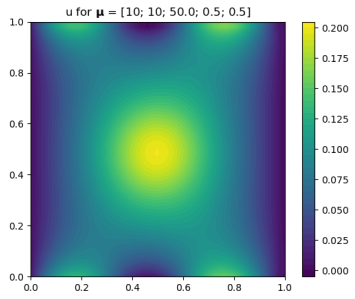
$$\text{Source: } f = \mu_2 \exp\{-\mu_3[(x - \mu_4)^2 + (y - \mu_5)^2]\} \tag{12}$$

In this particular case, Equations 10 and 11 specify the boundary conditions. The flux exhibits a sinusoidal wave pattern at the top and bottom, while the function u is constrained to zero at the left and right boundaries. Equation 12 defines the behaviour of the source, where u exhibits an exponential decay depending on how far away it is from the source.

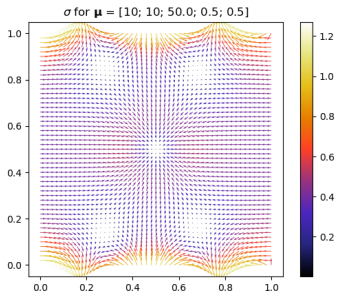


(a) base case u (b) base case σ

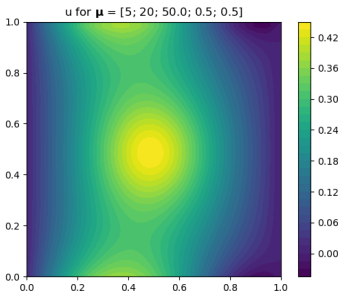
Figure 2: Base case solutions for $\boldsymbol{\mu} = [5; 10.0; 50.0; 0.5; 0.5]$



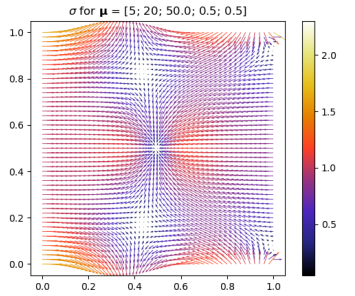
(a) Solution for u, $\mu_1 = 10$



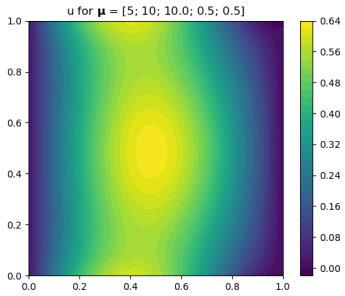
(b) Solution for σ , $\mu_1 = 10$



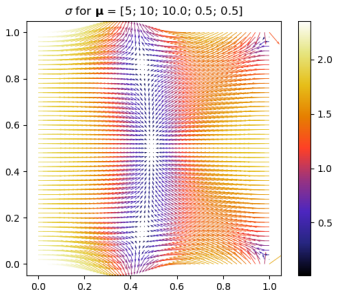
(c) Solution for u, $\mu_2 = 20$



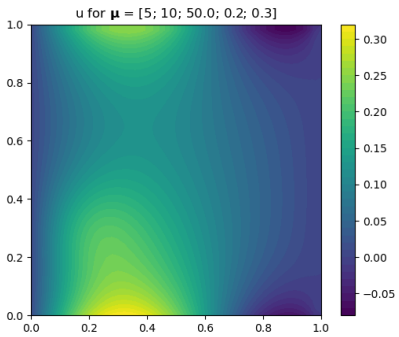
(d) Solution for σ, $\mu_2 = 20$



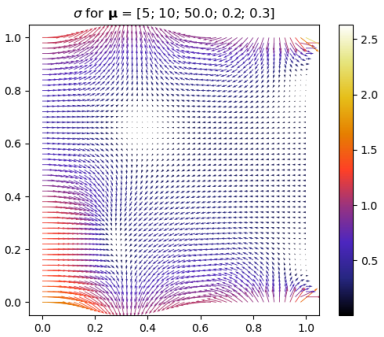
(e) Solution for u, $\mu_3 = 10$



(f) Solution for σ , $\mu_3 = 10$



(g) Solution for u, $\mu_4, \mu_5 = 0.2, 0.3$



(h) Solution for σ , $\mu_4, \mu_5 = 0.2, 0.3$

Figure 3: Solutions for different parameter variations with base case $\boldsymbol{\mu} = [5; 10.0; 50.0; 0.5; 0.5]$

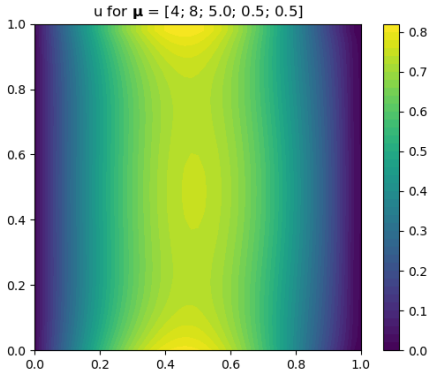
Solutions are then compared with different initialisations of μ_i. During the early stages of the project, we focused on changing all five parameters μ_i, where $i \in \{1, 2, 3, 4, 5\}$. As a based case for comparison, the parameters are set to be $\boldsymbol{\mu} = [5, 10.0, 50.0, 0.5, 0.5]$. Figure 2 shows the behaviours of the scalar and the flux term under the base case. The parameters were then varied and the effect of each μ_i are found below, based on results from Figure 3:

- μ_1 controls the frequency of the flux boundary.
- μ_2 scales the overall amplitude of u. The flux would also increase with a larger μ_2, especially in between the source and the boundary, where $u = 0$ is fixed.
- μ_3 controls the decay rate of the propagation of u from the source. For smaller μ_3, u is

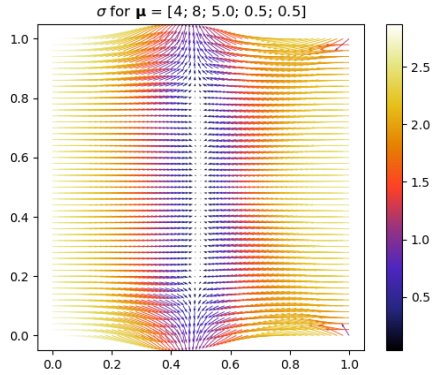
larger in unconstrained regions. However, due to fixed boundary conditions, the flux is larger at some regions near the boundary, counterfeiting the effect of smaller decay.

- μ_4 and μ_5 simply control the source position.

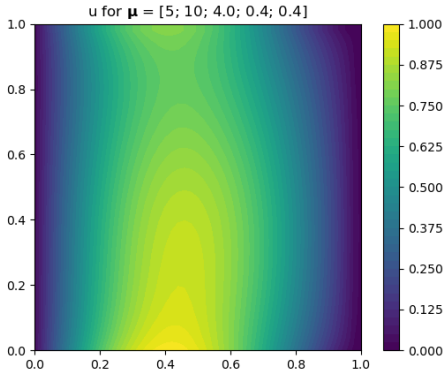
As the project progressed, we observed that certain parameters had a greater influence on the variability of the solution manifolds. Notably, through extensive experimentation, it was observed that the parameters μ_3, μ_4, and μ_5 exhibited considerable fluctuations in the solution. These fluctuations led to a major issue during the later stage of the project, where the MOR procedure could not accurately capture the dominant modes of the solutions, particularly in reducing the dimensionality of the solution for σ.



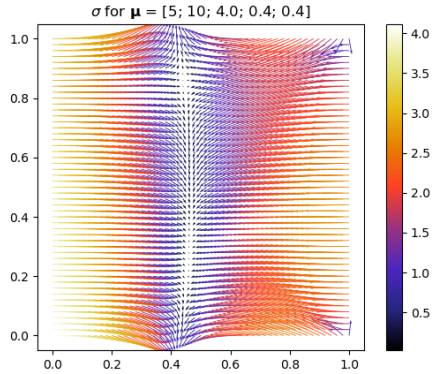
(a) Solution for u, changing $[\mu_1, \mu_2]$



(b) Solution for σ, changing $[\mu_1, \mu_2]$



(c) Solution for u, changing $[\mu_3, \mu_4, \mu_5]$



(d) Solution for σ , changing $[\mu_3, \mu_4, \mu_5]$

Figure 4: Solutions when varying $[\mu_1, \mu_2]$ and $[\mu_3, \mu_4, \mu_5]$

Figure 4 shows a comparison of the case when we decrease the value of parameters $[\mu_1, \mu_2]$ each by 20% and $[\mu_3, \mu_4, \mu_5]$ each by 20%, with respect to the base case. Compared with the base case in Figure 2, a larger variation in terms of both shape and magnitude is observed when changing $[\mu_3, \mu_4, \mu_5]$. This is especially the case for σ when comparing figures 2b, 4b, and 4d. Physically, this could be explained by the fact that distorting the source location and the decay rate of u while keeping the boundary fixed would lead to a large fluctuation in the flux change.

Consequently, to streamline the analysis, we decided to simplify the problem formulation. In this revised approach, the investigation would primarily concentrate on the parameters μ_1 and μ_2, as they were found to have a more pronounced impact on the solution behaviour and a better level of controllability. By narrowing the focus to these key parameters, the complexity of the problem could be effectively reduced while still capturing the essential dynamics and trends. This allows us to study the effectiveness of the ANN-POD approach more closely.

The effect of the number of parameters on the accuracy of POD will be discussed in more details in Sections 4 and 6, where we will provide concrete numerical results to support our stand of using only two parameters.

4 Model Order Reduction on Solutions

The underlying idea of MOR is that each high-dimensional solution can be represented in a low-fidelity space through a reduced number of global basis functions because the parametric dependence has a lower intrinsic dimension. By extracting the most significant features from high-dimensional datasets, MOR enables the creation of reduced-order models that retain essential dynamical behaviour on a reduced basis space. This approach proves particularly advantageous in scenarios where computational resources are limited or rapid simulations are required.

The MOR procedure usually comprises 2 stages. The first stage is performed offline, which is more expensive and computationally demanding. It involves forming the solution snapshot matrix $\mathbf{S}$ that includes N high-fidelity solutions, each of M degrees of freedom, solved by the classical FEM. The reduced basis space $\mathbf{V}_{rb}$ is then found through specific MOR techniques (i.e. the reduced basis functions that form the space). There are many techniques for generating the reduced basis spaces, such as the Proper Orthogonal Decomposition (POD), the Proper Generalized Decomposition (PGD) and the Principle Component Analysis(PCA). In this project, we focused on using POD on the snapshot matrix to obtain the reduced basis space, and the Python RBniCSx package provided full support for the related operations in POD.

This project also incorporates the training of the ANN model in the offline stage. During this stage, we generate training parameters and project them onto the reduced basis space to obtain the training outputs for the ANN. We then train the ANN model for predicting solutions on the reduced space. More information on the ANN model will be covered in the next section.

The second stage is an online stage where we can make use of the trained ANN-POD model and obtain the high-fidelity solution by passing the input parameter $\boldsymbol{\mu}$ into the ANN followed by reconstructing the full-order solution with the reduced basis functions obtained from stage one. This approach would significantly reduce our reliance on the costly FEM. This online process can be integrated into a computer system with restricted computational resources and memory capacity, enabling quick real-time access to the response of a complex system [**?**].

Consider that we now have the reduced basis space $\mathbf{V}_{rb}$ that is used to represent the original function space $\mathbf{V}$. $\mathbf{V}_{rb}$ is formed by the set of L basis functions $\mathbf{u}_{rb} = [\mathbf{u}_1, ..., \mathbf{u}_L]$, which represent the N solutions from the snapshot matrix $\mathbf{S}$. Usually, we have $L \ll M$, where M is the original degrees of freedom on the full-order solution so that MOR works well in dimensionality reduction. Each original solution $\mathbf{s}_n$ from $\mathbf{S}$ can be represented on the reduced basis space with equation 13.

$$\sum_{l=1}^{L} \langle \mathbf{s}_n, \mathbf{u}_l \rangle_{\mathbb{R}^M} \mathbf{u}_l \tag{13}$$

Where $\langle \cdot, \cdot \rangle$ defines the inner production action. Hence, to obtain the best set of the reduced basis functions $\mathbf{u_{rb}}$, our goal would be to minimise the error in equation 14:

$$\epsilon = \sum_{n=1}^{N} \left\| \mathbf{s}_n - \sum_{l=1}^{L} \langle \mathbf{s}_n, \mathbf{u}_l \rangle_{\mathbb{R}^M} \mathbf{u}_l \right\|_{\mathbb{R}^M}^2 \tag{14}$$

4.1 Proper Orthogonal Decomposition

The appeal of POD lies in its ability to identify the dominant modes of variation in data through singular value decomposition (SVD). The Schmidt-Eckart-Young theorem for matrices states that the optimal rank-l approximation of a matrix in the Euclidean topology is obtained by retaining the first l terms from the SVD of that matrix [**?**, **?**]. These corresponding left eigenvectors form the reduced basis space, where the full-order solutions are projected.

Consider a snapshot matrix $\mathbf{S}$ of dimension $N \times M$, where N is the number of snapshots and M is the degrees of freedom for each high-fidelity FEM solution. By performing singular value decomposition (SVD) on $\mathbf{S}$, we can express $\mathbf{S}$ as:

$$\mathbf{S} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{15}$$

Where $\mathbf{U}$ is an $N \times N$ matrix containing the left eigenvectors (also known as POD modes or basis functions), $\mathbf{\Sigma}$ is the $N \times M$ diagonal matrix containing the singular values $\{\sigma_1, \sigma_2, ..., \sigma_{min(N,M)}\}$ in decreasing order, and $\mathbf{V}^T$ is the transpose of an $M \times M$ matrix containing the right eigenvectors.

If we want to obtain the L basis functions $\mathbf{u_{rb}} = \{\mathbf{u}_1, ..., \mathbf{u}_L\}$ that best represent the snapshot matrix $\mathbf{S}$, we would pick the left eigenvectors in $\mathbf{U}$ corresponding to the most significant L singular values from $\mathbf{\Sigma}$ (i.e. $\{\sigma_1, \sigma_2, ..., \sigma_L\}$).

We can reinforce two stopping criteria to determine the value of L (i.e., the number of modes that form the reduced basis space). The first one is to set an arbitrary maximum number of modes N_{max} that we are willing to accept. The second criterion is the tolerance on the retained energy ϵ. This is defined as the ratio between the smallest singular value to the largest singular value (i.e. pick L such that $\epsilon \geq \frac{\sigma_L}{\sigma_1}$). The smaller ϵ is, the more information about the full-order model we wish to retain in $\mathbf{V}_{rb}$. In practice, we will stop when either condition is met.

4.2 Solution Projection

Once we have the modes, we have formed the reduced basis space $\mathbf{V}_{rb}$ where each FEM solution can then be projected. To project a high-fidelity solution $\mathbf{s}$ onto a reduced basis space represented by $\mathbf{u_{rb}} = \{\mathbf{u}_1, ..., \mathbf{u}_L\}$, we can use the Galerkin projection method. This involves computing the inner product of the high-fidelity solution with each basis function and multiplying it by the corresponding basis function. Mathematically, this is given by:

$$\mathbf{s}_{rb} = \sum_{i=1}^{L} \langle \mathbf{s}, \mathbf{u}_i \rangle \mathbf{u}_i$$

Where the operation $\langle \mathbf{s}, \mathbf{u}_i \rangle$ defines the inner production action between the high-fidelity solution and the i^{th} basis function. This is similar to what we had in equation 13 but for

different purposes, where equation 13 was used for estimating the best set of $\mathbf{u}_{rb}$ during the offline stage, and over here we are trying to utilise the information during the online stage.

4.3 Solution Reconstruction

Later in the report, we will discuss the use of ANN in obtaining the predicted solutions on the reduced basis space. However, we are still interested in the full-order solution, which requires a way of reconstructing the high-fidelity solution and measuring the error between the original FEM solution and the reconstructed solution. For a set of basis functions $\mathbf{u_{rb}} = \{\mathbf{u}_1, ..., \mathbf{u}_L\}$ and a reduced basis solution $\mathbf{s}_{rb} = \{s_1, ..., s_L\}$, the reconstructed solution $\mathbf{v}$ can be simply represented by multiplying the basis functions by their corresponding coefficients and summing over all modes:

$$\mathbf{v} = \sum_{i=1}^{L} s_i \cdot \mathbf{w}_i \tag{16}$$

To measure the error between the original solution and the reconstructed solution, the relative error was calculated as the ratio of the L_2-norm error to the L_2-norm of the original solution $\mathbf{u}$, which would give us a percentage that represents the amount of deviation from the FEM solution:

$$\epsilon = \frac{\|\mathbf{u} - \mathbf{v}\|}{\|\mathbf{u}\|} \tag{17}$$

The specific norm operation is defined by the specific definition of the inner product action used in the function. The L_2-norm is commonly used in functional analysis and finite element methods to measure the magnitude of functions, which is defined in equation 18.

$$\|f\|_{L_2(\Omega)} = \left(\int_{\Omega} |f(x)||q(x)|\, dx \right)^{1/2} \tag{18}$$

The other possible choice is the H_1-norm that incorporates the first derivative of the function given by equation 19. The H_1-norm would incur additional computational cost in computing the gradient term. In this report, we used the L_2-norm inner product action for u because it provides a measure of magnitude without considering directional information.

$$\|f\|_{H_1(\Omega)} = \left(\|f\|^2_{L_2(\Omega)} + \|\nabla f\|^2_{L_2(\Omega)} \right)^{1/2} \tag{19}$$

For the case of σ, we included the inner product of the divergence term in the L_2-norm. This is similar to equation 19, but changed the grad operation to divergence due to σ being a vector field, as shown in equation 20. This also helps to capture the spatial variation (i.e. the smoothness) of the fields.

$$\|f\| = \left(\|f\|^2_{L_2(\Omega)} + \|\nabla \cdot f\|^2_{L_2(\Omega)} \right)^{1/2} \tag{20}$$

4.4 Choice of Parameters Based on POD Result

In Section 3, an examination was conducted regarding the selection of parameters ($\boldsymbol{\mu} = [\mu_1, \mu_2]$ versus $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_3, \mu_4, \mu_5]$), and we explained our rationale in using only 2 parameters by showing the large variability of the solution manifold imposed by changing 5 parameters altogether. The motivation for choosing 2 parameters is further proved when we examine the POD outcomes obtained using different parametrisations. The choices are checked through a comparison of the POD reconstruction errors.

Each parameter is adjusted along a Numpy *linspace* scale, ensuring equal intervals between values. Combining each value creates a collection of input parameters. The full-order model was solved for each $\boldsymbol{\mu}$, followed by performing POD on the snapshot matrix, which creates a reduced basis space. In the first case, we only varied the two parameters μ_1 and μ_2, each taking 10 values (a total of 100 solutions for POD). In the second case, we varied all 5 parameters taking $4 \times 4 \times 3 \times 3 \times 3$ values (a total of 432 solutions for POD).

To measure the accuracy and reliability of POD, we initialised 100 random samples of input parameters and solved for the high-fidelity solutions using FEM. The solutions were then projected onto $\mathbf{V}_{rb}$ with $\mathbf{u}_{rb}$ to form the reduced basis solution $\mathbf{s}_{rb}$. The quality of POD can then be quantified by calculating the average relative error of the solution reconstructed using equation 16 and 17. Getting the POD error as small as possible is an essential first step in the project. Only with a reasonably accurate POD process can we safely proceed to the next step of ANN training, which requires reliable training and validation data.

In addition to the relative error, we can compare eigenvalues obtained from SVD for different snapshot matrices. If the eigenvalue decays faster, we can obtain a lower intrinsic dimensional space representing the parametric dependence on high-fidelity solutions. This would enhance the computational efficiency and increase the accuracy of POD.

Multiple results have shown that the 5-parameter case would provide a very unstable and inaccurate result in POD, making it harder to proceed with the project. In contrast, changing only 2 parameters would lead to a more desirable POD result. While employing a lot more solution snapshots and modes may make the 5-parameter work slightly better, this was beyond the computational capacity of this project. In addition, POD only creates a linear subspace (i.e. only represents linear combinations of the basis vectors). However, the parameters may interact in complex ways that cannot be adequately represented by linear combinations alone, resulting in a very slow eigenvalue decay rate. These non-linearities of the feature space were particularly relevant for μ_3, μ_4, and μ_5. The detailed results for both the errors and the eigenvalue decay will be presented in Section 6.

5 ANN-POD Based Model

Artificial Neural Network (ANN) is a computational model inspired by the human brain's neural networks. These networks consist of interconnected nodes, called neurons, which process information. ANNs learn from examples, adjusting their internal parameters through training. They are organized into layers, including an input layer where data is fed, one or more hidden layers where computation occurs, and an output layer that produces the network's final output. An example of a neural network is shown in Figure 5.

With the ANN-POD model that we trained, the solution matrix of any set of N input parameters can be determined offline as $\mathbf{S} = \mathbf{B}\mathbf{u}_{rb}$, where $\mathbf{B}$ is a $N \times L$ matrix including coefficients corresponding to each reduced basis function for every parameter, which is obtained as outputs of the ANN model. $\mathbf{u}_{rb}$ is the set reduced basis functions obtained previously in section 4 during the POD process.

In this section, a feed-forward ANN was built with TensorFlow separately for the scalar field u and then flux term σ. In both models, the input layer has the dimension of input data (in this case, input $\boldsymbol{\mu}$ consisted of two parameters), and the output shapes are the number of reduced basis functions we obtained from POD. Different numbers of hidden layers H were used between the inputs and outputs with n neurons for each layer and the optimal number was determined through trial and error.

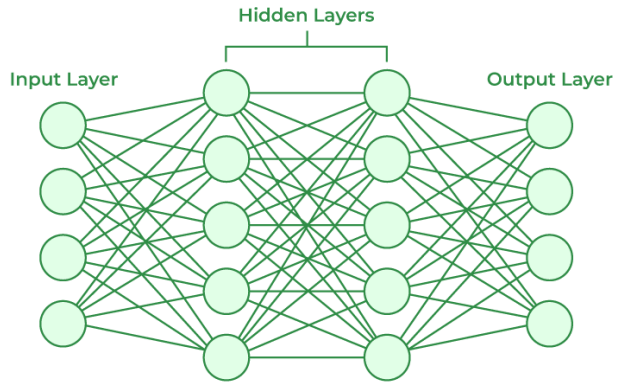


Figure 5: An example neural network with input shape 3, $H = 2$ hidden layers and $n = 5$ neurons

5.1 Data Generation

The data samples are generated through a technique called Latin Hypercube Sampling (LHS) [**?**], which is a statistical method for generating a near-random sample of parameter values from multiple-dimensional space. The process ensures that each sample represents a unique combination of values across all dimensions.

Consider the example of N samples with M dimensions [1], each parameter dimension is divided into N equal intervals. Within each interval along each dimension, one point is

[1] In our case, we chose $M = 2$, the number of parameters in μ

randomly selected, and the samples within each dimension are randomly shuffled. Finally, the shuffled samples from each dimension are randomly combined to obtain the final set of N Latin hypercube samples. LHS reduces systematic bias due to the sequential generation of samples through shuffling, while preserving the generality by considering samples across the whole interval within each dimension.

The LHS has provided us with a random set of input parameters. We could then calculate the corresponding outputs of shape L by solving for the high-fidelity FEM solutions and project them onto the reduced basis space $\mathbf{V}_{rb}$ using the method described in Section 4.

5.2 Feature Scaling

Feature scaling is a method in data preprocessing that aims to normalise the values of variables within a dataset. This approach makes the data comparable on a similar scale and prevents any single feature from overshadowing others due to its larger values. In this project, the simple Min-Max Scaling scheme is used, where the data are scaled linearly based on a predefined range, usually determined by the activation function used [2]. For a data set where the minimum and maximum values are μ_{min} and μ_{max}, and the scaled range is $[a, b]$, each input μ_i can be scaled as:

$$\mu_i^{norm} = (b - a) \times \frac{\mu_i - \mu_{min}}{\mu_{max} - \mu_{min}} + a \tag{21}$$

Note that the input $\boldsymbol{\mu}$ in our case has two dimensions, so each dimension is scaled independently using Equation 21. In addition, Equation 21 also defines the way of reconstructing the original μ_i using the scaled value μ_i^{norm}, as well as the scaling range defined by $[\mu_{min}, \mu_{max}]$.

5.3 Training of ANN

In this project, we used a feed-forward ANN with H hidden layers, each of n neurons. The N samples of data points are randomly split into an 80% group for training and a 20% group for validation purposes. The training dataset is also grouped into batches to speed up the training of the ANN, which means that the losses are summed up within each batch before performing backward propagation for gradient optimisation.

In training the ANN, we used the Adam optimiser with different learning rates, with the optimal choice tuned through trials and errors. For the loss function, we firstly tried the mean absolute error(MAE), which measures the mean of the sum of the absolute difference of the coefficients between the predicted results and the original reduced basis coefficients for each mode:

$$\text{MAE} = \frac{1}{N} \sum_{n=1}^{N} \sum_{l=1}^{L} |s_l^{(n)} - \hat{s}_l{}^{(n)}| \tag{22}$$

Another common choice for the error function was the mean squared error (MSE), which is defined as follows:

[2] In this project, we mostly used the scaling range [0,1] with the sigmoid activation function or [-1, 1] with the tanh-hyperbolic activation function

$$\text{MSE} = \frac{1}{N}\sum_{n=1}^{N}\sum_{l=1}^{L}\left(s_l^{(n)} - \hat{s}_l{}^{(n)}\right)^2 \tag{23}$$

Both choices of loss functions were implemented and compared, and eventually, the MSE was preferred. In our case, we decided to place more tolerance on smaller localised errors. Since MSE squares the errors before averaging them, it penalises large errors more heavily than smaller ones. This property makes MSE particularly useful in scenarios where it is important to minimise the impact of outliers or large errors, as it prioritizes reducing these errors to improve overall performance. Additionally, the mathematical properties of MSE, such as differentiability, make it well-suited for optimisation. A more quantitative comparison will also be provided in Section 6.

To terminate the training of the ANN, we introduced two stopping criteria. The first one is simply the maximum number of training epochs, and the training stops if the number of training iterations exceeds the limit. The second one is the early-stopping criterion with a tolerance of 3 training epochs. This means that if there are no improvements in the validation data loss for 3 consecutive iterations, we would stop the training process and the model parameters at the point of early stopping are retained as the final model. The early-stopping criterion is a regularisation technique that prevents overfitting of the training data and improves its generalisation ability to unseen data., while the incorporated tolerance also gives a chance for the optimisation process to escape from the local optimum.

Batch normalisation was also used in the training process. It normalises the inputs of each layer in a neural network to have a mean of zero and a standard deviation of one. This is done by adjusting the activations using the mean and variance calculated over the current mini-batch. For each batch during training, before passing the inputs to the next layer, batch normalization computes the mean (μ_B) and variance (σ_B^2) of the activations along each feature dimension. The activations for each feature are normalised using the mean and variance obtained from the mini-batch using $\hat{x}_i = \frac{x_i - \mu_B}{\sigma_B}$.

The batch normalisation helps stabilise and speed up the training of deep neural networks by reducing internal covariate shifts. In addition, it allows for the use of higher learning rates, which can accelerate convergence during training.

With the help of the ANN, the original PDE can now be solved in a data-driven approach, and knowledge of the original problem (i.e. the governing equations, weak forms etc) is no longer needed. This greatly simplifies the computation, with some sacrifice on the solution accuracy.

The effectiveness of ANN depends heavily on the choices of hyperparameters including the learning rate, the number of neurons per layer n and the number of hidden layers H. To evaluate the performance of the ANN, an approach similar to the one used for evaluating POD was used. New testing samples of inputs were randomly initialised and solved with ANN for the predicted solution on the reduced space, as well as the FEM method for the high-fidelity solution. The ANN solutions were then reconstructed using $\mathbf{u}_{rb}$, and the relative error could be calculated against the high-fidelity FEM solution. This allows us to examine the performance of the trained ANN model. Optimal choices of these hyperparameters were discovered through trials and errors. Details of the results can be found in the next section.

5.4 Parameter Range

Another important aspect of concern is the range of each parameter. In all cases, the input parameters [3] are generated within the input range we define. These ranges also determine the scaling in equation 21. Initially, we kept an arbitrarily general range for both μ_1 and μ_2. This was during the later part of the project refined to $\mu_1 \in [2\pi, 2.5\pi]$ and $\mu_2 \in [12, 13]$, where an acceptable level of accuracy could be achieved from the trained model with a reasonably low number of training samples needed. Ideally, we would like as wide a range as possible (and within a physical sense) to improve the generality of the ANN-POD model. However, the actual performance might be limited, and a wider range of parameters would require a more complex model, more training data and training time, which was not further discovered due to the limited computational resources and time of this project.

[3] These include the input parameters generated for POD to form the reduced basis, training and validation data for the ANN, as well as the testing data for the final model.

6 Numerical Results

Currently, the Python package DLRBniCSx facilitates the creation of ANN-POD-based models for solving different types of partial differential equations. While the package currently integrates PyTorch for ANN functionality, a significant aspect of this project involves extending support for TensorFlow within DLRBniCSx. This expansion encompasses various aspects such as model construction, training-validation, data preprocessing, and auxiliary functionalities like error analysis. Further information regarding these enhancements can be found on the project's GitHub page.

6.1 Effectiveness of POD

Recall that in Section 4, we compared the choices of parameterisation through the POD results in both cases. In the first scenario, we varied only two parameters (μ_1 and μ_2), with each parameter taking on 10 values. This configuration yielded a total of 100 solutions for POD. In the second scenario, we explored all five parameters, each taking $4 \times 4 \times 3 \times 3 \times 3$ values. This resulted in a larger dataset consisting of 432 solutions for POD. In both cases, we set $N_{max} = 20$ and $\epsilon = 10^{-4}$ as the stopping criterion [4].

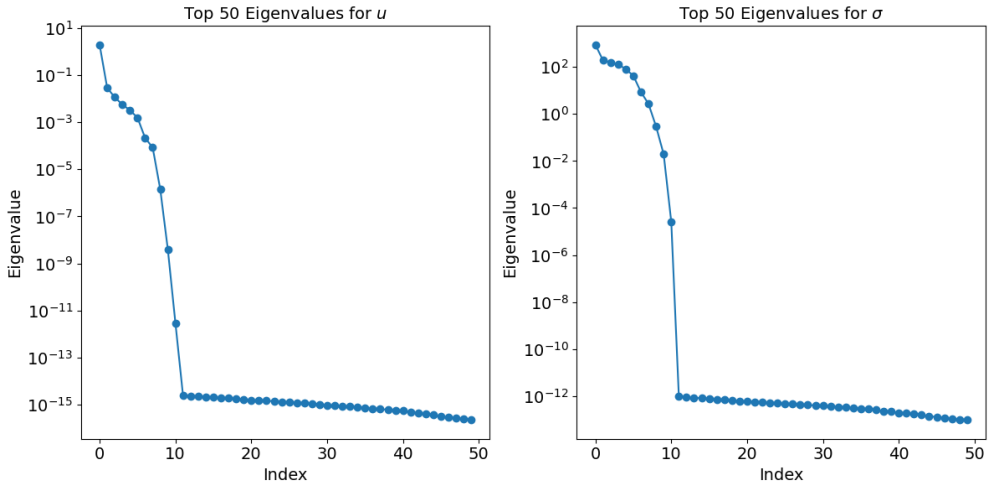


Figure 6: Top 50 Eigenvalues from POD with varying μ_1 and μ_2, each parameter spanning 10 discrete values

Table 1 compares the proposed two cases, where we can observe that the reconstruction error for both u and σ were greatly reduced with only 2 parameters and the size of the reduced basis space despite a much lower number of solutions used. Note that for σ in the 5-parameter case, the size of the reduced basis was even reaching the maximum threshold N_{max}, meaning that ideally we still need more modes to capture the features of the solutions.

Figure 6 shows the decay of eigenvalues when performing SVD on the snapshot matrix, varying μ_1 and μ_2, each parameter spanning 10 discrete values (total 100 combinations). Figure 7 shows the decay of eigenvalues when varying all parameters, with the number of

[4] see Section 4.1 for details

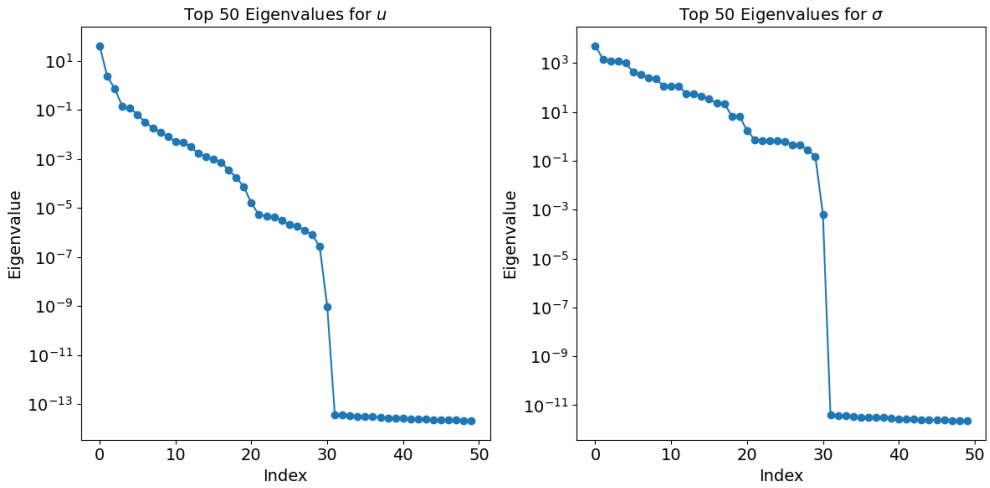


Figure 7: Top 50 Eigenvalues from POD with varying all parameters, with μ_1 and μ_2 spanning 4 discrete values, μ_3, μ_4 and μ_5 each spanning 3 discrete values

Table 1: Results of POD and reconstruction on different parameter sets

Number of Parameters	2	5
Sample Size	10×10	$4 \times 4 \times 3 \times 3 \times 3$
Size of $\mathbf{V}_{rb}$ (u, σ)	(7, 9)	(14, 20)
Mean Error (u, σ)	(0.012, 0.005)	(0.062, 0.246)
Max Error (u, σ)	(0.067, 0.015)	(0.293, 0.781)
POD Time in Seconds (u, σ)	(30, 240)	(2, 13)

values for $\boldsymbol{\mu}$ equals $[4, 4, 3, 3, 3]$ (total 432 combinations). The eigenvalues for both u and σ decay very quickly with only 2 varying parameters, while taking much longer for the 5-parameter case. This also corresponds to the fact that a higher number of reduced basis functions was needed.

Further to the accuracy of POD, the computational efficiency of the POD process is also greatly hindered by the high number of training snapshots needed with more parameters. Due to the nature of the snapshot matrix being a dense matrix, it was impossible to parallelise the process across multiple CPU cores, so the POD process would take a long time in serial. Tabel 1 proved this point as the POD time for 2 parameters was less than 10% than that for 5 parameters, and this was even excluding the time for generating the solution snapshot matrix. In addition, a higher number of reduced basis sizes would also incur additional costs in projecting the high-fidelity solution onto the reduced basis space. This proved to be particularly costly as we later tried to initialise the training inputs for the ANN, which required a large amount of data points.

These results have justified our choices of parameterisations on $[\mu_1, \mu_2]$ only, which ensured the numerical stability of the results. A further enhancement was made by reducing the tolerance for retained energy of POD to $\epsilon = 10^{-6}$. This further brought the POD error down by an order of 10, while only introducing an additional reduced basis size for both u and σ,

which we believe is worth the trade-off, as shown by the results presented in table 2

Table 2: Results of POD reconstruction with different ϵ, sample size $= 10 \times 10$

ϵ	Size of $\mathbf{V}_{rb}$ $(u,\ \sigma)$	Mean Error $(u,\ \sigma)$	Max Error $(u,\ \sigma)$
10^{-4}	(7, 9)	(0.012, 0.005)	(0.067, 0.015)
10^{-6}	(8, 10)	$(8.6 \times 10^{-4},\ 3.5 \times 10^{-4})$	(0.0028, 0.0023)

All the results above were recorded based on the case where we employed a general parameter range where $\mu_1 \in [5, 20]$ and $\mu_2 \in [10, 20]$. Although we, later on, chose a different range because of the ANN accuracy, they still served the purpose of selecting the suitable number of parameters and ϵ. Tuning the parameter range of interest to $\mu_1 \in [2\pi, 2.5\pi]$ and $\mu_2 \in [12, 13]$ after we studied the results for the ANN model, the POD results are presented below in table 3:

Table 3: Results of POD reconstruction with $\mu_1 \in [2\pi, 2.5\pi]$, $\mu_2 \in [12, 13]$ and $\epsilon = 10^{-6}$

Sample Size	Size of $\mathbf{V}_{rb}$ $(u,\ \sigma)$	Mean Error $(u,\ \sigma)$	Max Error $(u,\ \sigma)$
10×10	(4, 4)	$(3.3 \times 10^{-4},\ 8.3 \times 10^{-4})$	$(7.3 \times 10^{-4},\ 2.20 \times 10^{-3})$

6.2 Effectiveness of ANN

##

Serial: Include training time and data generation time, snapshot computation time for POD, POD process time, eigenvalue decay, loss wrt to epochs for ANN, projection error and ANN error Snapshot computation time in parallel, with different solvers (not main focus of work)

Add a ANN sample figure with input and output shape represented

Model Description, discussions on layers and number of neurons, activation functions and maybe stopping criteria.

Final results for a sigmoid activation function with 500 training and validation samples: Mean error for U is: 0.398826; Mean Error for SIGMA is 0.862105 Maximum error for U and SIGMA are: 0.771895, 0.999538

Final results for a sigmoid activation function with 1000 training and validation samples: (mu1 and mu2 only) Mean error for U is: 0.147512; Mean Error for SIGMA is 0.252696 Maximum error for U and SIGMA are: 0.582387, 0.605089

Final results for a sigmoid activation function with 1000 training and validation samples: Mean error for U is: 0.102683; Mean Error for SIGMA is 0.248321 Maximum error for U and SIGMA are: 0.553261, 0.813254

Final results for a sigmoid activation function with 500 training and validation samples: Learing rate = 0.002, ANN structure = [20, 20] Mean error for U is: 0.055362; Mean Error for SIGMA is 0.152036 Maximum error for U and SIGMA are: 0.163079, 0.442326

Activation: sigmoid Learing rate = 0.003, ANN structure = [25, 25] Mean error for U is: 0.066273; Mean Error for SIGMA is 0.140492

(MSE for u, MAE for sigma) Final results for a sigmoid activation function with 500 training and validation samples: Learing rate = 0.002, ANN structure = [20, 20] Mean error for U is: 0.081644; Mean Error for SIGMA is 0.167260 Maximum error for U and SIGMA are: 0.240615, 0.388974 Maximum error for U and SIGMA are: 0.212036, 0.228014

(MSE for both) Final results for a sigmoid activation function with 500 training and validation samples: Learning rate = 0.0025, ANN structure = [20, 20] Mean error for U is: 0.090378; Mean Error for SIGMA is 0.304891 Maximum error for U and SIGMA are: 0.278097, 0.787646;

MSE for both Final results for a tanh activation function with 400 training and validation samples: Learing rate = 0.0005, ANN structure sigma:[15, 15], u: [25, 25] Mean error for U is: 0.025661; Mean Error for SIGMA is 0.026224 Maximum error for U and SIGMA are: 0.043447, 0.069763

Final results for a tanh activation function with 200 training and validation samples: Learing rate = 0.0005, ANN structure sigma:[15, 15], u: [25, 25] Mean error for U is: 0.026253; Mean Error for SIGMA is 0.075967 Maximum error for U and SIGMA are: 0.048862, 0.110272 ##

final loss from each model effect of the number of training parameters effect of learning rate and optimisers final prediction norm error

7 Parallelisation

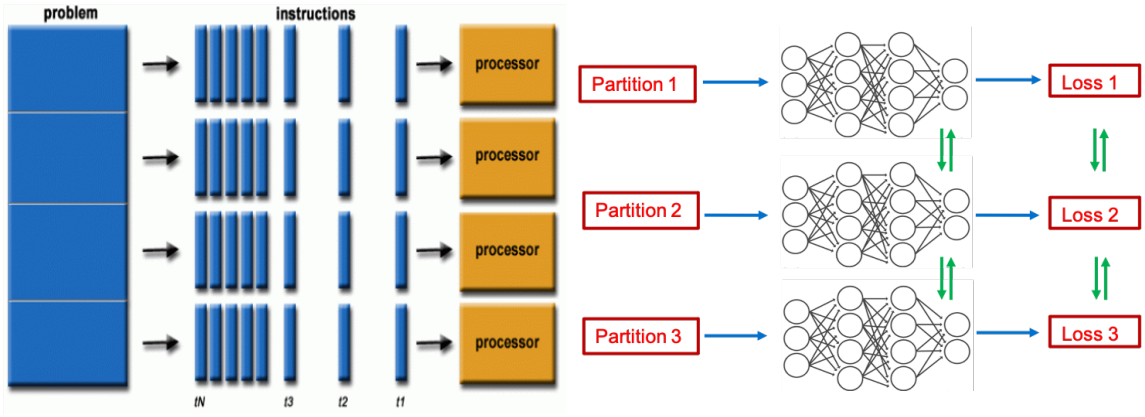


Figure 8: Parallel computing and its application in training an NN

Parallelisation refers to the technique of breaking down a task into smaller subtasks that can be executed simultaneously or in parallel by multiple processing units. These processing units could be CPU cores within a single processor, multiple processors within a computer, or even distributed computing resources across a network. The goal of parallelisation is to speed up the execution of a task by distributing the workload across multiple processing units, thereby reducing the overall execution time. This is particularly useful for computationally

intensive tasks divided into independent or loosely coupled subtasks. In this project, we utilised both GPU and CPU parallelisation to speed up the process of training the ANN and the process of performing POD and generating training data.

7.1 CPU-parallelisation

7.2 GPU-parallelisation