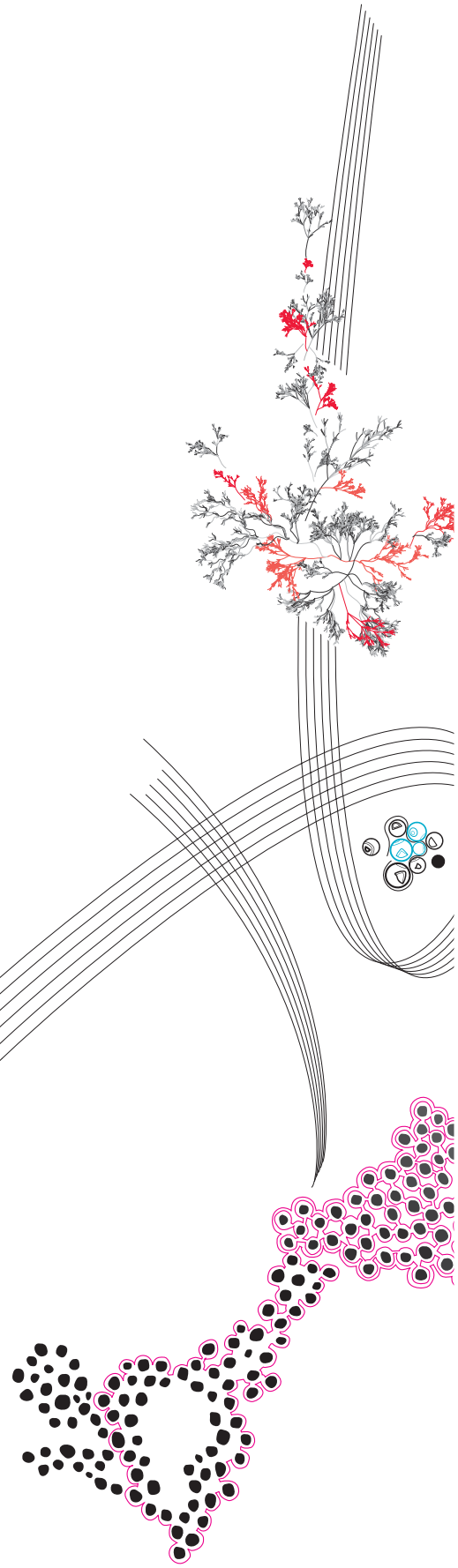


MSc Thesis Applied Mathematics

# Reduced-order surrogate modeling for linear-nonlinear coupled problems

Paul Stuiver

Daily supervisor: dr. M. Guo
Committee chair: prof. dr. C. Brune
Committee member: dr. S.M. Glas

October, 2023

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science
Chair of Mathematics of Imaging & AI

UNIVERSITY OF TWENTE.

## Acknowledgements

I'm grateful of the chance to have been involved in this thesis project. First and foremost, I would like to thank Mengwu Guo for all the help over the past year. You have given me the opportunity grow both on academic and professional level. I have always felt extremely welcome and heard during our meetings. You have been very flexible and shown to be closely involved in this thesis. Next, I would like to thank Christoph Brune for the valuable feedback during this thesis. I would like to thank Dongwei Ye for providing me feedback on my draft version. I would like to thank Andrea Manzoni for his suggestions.

Being invested in a single topic alone for a longer time was quite a new experience. I would like to thank Feike and Sjon for keeping me company during the writing of this thesis.

# Reduced-order surrogate modeling for linear-nonlinear coupled problems

P. Stuiver *

October, 2023

**Abstract**

**Introduction**: Parametrized linear-nonlinear coupled problems consist of a linear and non-linear sub-domain that are coupled via a non-overlapping interface. Solving these problems with a conventional full-order approach such as the finite element method (FEM) is usually paired with prohibitive computational cost. Reduced-order models are required to address these limitations.

**Methods**: We developed reduced-order surrogate models to solve parametrized linear-nonlinear coupled problems with improved efficiency. These models utilize domain decomposition to apply both intrusive and non-intrusive reduced-order methods with POD-ROM and POD-NN on sub-domains separately. The choices of the reduction techniques on the sub-domains are based on the parametric complexity and the underlying equations (linear or non-linear) in order to implement reduction effectively. The designed ROMs termed hybrid-LL and surrogate-LL act as proof of concepts to solve linear-linear coupled problems, while the hybrid-NL and surrogate-NL are designed to solve linear-nonlinear coupled problems.

**Results**: We report the performance of the ROMs on three steady-state PDEs with increased parametric complexity. While both the hybrid-NL and surrogate-NL guarantee solutions, the hybrid-NL is inaccurate for particular parameter locations and is usually costly. In contrast, the surrogate-NL is successfully tested and shows minimal errors with extremely low computational costs. Equally, in the field of uncertainty quantification, the surrogate-NL shows promising results using Monte Carlo simulations.

**Conclusion**: While the hybrid-NL does not perform as desired due to lacking robustness and computational gain, the surrogate-NL has shown to perform remarkably well with low errors and computational cost.

*Email: p.stuiver@student.utwente.nl

# Table of Contents

## Table of Symbols

| Abbreviation | Description |
|---|---|
| POD | Proper orthogonal decomposition |
| FOM | Full-order model |
| ROM | Reduced-order model |
| FEM | Finite element method |
| PDE | Partial differential equation |
| POD-ROM | Reduced-order model based on POD |
| RB | Reduced basis |
| POD-NN | Reduced-order model based on POD and neural networks |
| SVD | Singular value decomposition |
| BFGS | Broyden-Fletcher-Goldbarf-Shanno algorithm |
| Hybrid-LL | hybrid ROM to solve linear-linear coupled problems |
| Surrogate-LL | surrogate ROM to solve linear-linear coupled problems |
| Hybrid-NL | hybrid ROM to solve linear-nonlinear coupled problems |
| Surrogate-NL | surrogate ROM to solve linear-nonlinear coupled problems |
| OoI | Output of interest |
| QoI | Quantity of interest |
| PDF | Probability density function |

| Symbol | Description |
|---|---|
| $\boldsymbol{\mu}$ | Parameter vector |
| $\boldsymbol{\mu}^i$ | $i$-th parameter vector |
| $\mu^{(i)}$ | $i$-th element of the parameter vector |
| $\mathcal{P}$ | Parameter space |
| $\mathbb{1}_\Omega$ | Indicator function w.r.t. $\Omega$ |
| $\Delta$ | Laplace operator |
| $\mathcal{N}_D$ | Non-linear diffusion operator |
| $\mathcal{N}_R$ | Non-linear reaction operator |
| $\mathcal{C}^c$ | Space of continuous functions with first $c$ derivatives |
| $L^p$ | Space of $p$-integrable functions |
| $H^n$ | $n$-th order Sobolev space |
| $\mathcal{M}$ | Solution manifold |
| $\mathbb{S}$ | Snapshot matrix |
| $\pi$ | Mapping |
| $\mathbb{V}$ | Transformation matrix |
| $\gamma$ | Step size |
| $R_k$ | Residual in iteration $k$ |
| $\mathbb{J}_F$ | Jacobian w.r.t. $F$ |
| $\boldsymbol{U}$ | Solution vector in full-order space |
| $\boldsymbol{U}^{rb}$ | Solution vector in full-order space from the ROM |
| $\boldsymbol{u}$ | Solution vector in reduced-order space |
| $\boldsymbol{u}^{rb}$ | Solution vector in reduced-order space from the ROM |
| $\boldsymbol{T}_\Gamma$ | Traction force |
| $\boldsymbol{t}_\Gamma$ | Reduced-order traction force |

# 1 Introduction

Partial differential equations (PDEs) have been extensively investigated due to their prominent role in various scientific fields such as physics and engineering. They can model quantities of interest in for example heat transfer, electrostatics and fluid dynamics [1–3]. Yet studied thoroughly, solving PDEs remains challenging along with heavy computational demands. Starting in the 1940s, the Finite Element Method (FEM) has developed into a general numerical method to solve PDEs and is still used today thanks to its accuracy [4, 5]. As a full-order numerical solver, FEM involves a large amount of degrees of freedom (DoFs) that in turn can demand a large computation time and memory cost for complex problems. In problems requiring a single solution once and for all, FEM is a feasible approach.

In applied sciences, parameters can model material characteristics, input forces or boundary conditions that are manifested in the problem [6–10]. In addition, in uncertainty quantification one is aiming to determine a statistical quantity of interest depending on errors in the input parameters [11, 12]. Another example is the increased popularity in digital twins defined as 'a model that brings together the technology to map, monitor and control real-world entities by continually receiving and integrating data from the physical twin to provide an up-to-date digital representation of the physical entity' [13, 14]. For digital twins to operate effectively, computations need to be carried out within a short amount of time. These many-query problems involving parameters require a set of solutions depending on the parameters rather than just a single solution [12, 15, 16]. The construction of the ensemble of solutions using FEM requires the corresponding problem to be solved time and again for each parameter location, which can quickly turn out to become prohibitive due to the paired high computational cost.

To overcome the issue of the computational requirements in the parametrized problems, the study of reduced order modeling has gained substantial ground over the past decades [17, 18]. The aim of reduced order modeling is to decrease the computational cost required from full-order methods (FOMs) so that solutions can be obtained in a much smaller time frame. Although designed to be computationally cheap, reduced order models (ROMs) exhibit a decrease in accuracy when compared to their corresponding FOMs. The challenge lies in finding a balance between an acceptable computational cost and a satisfactory accuracy.

Typically, ROMs consist of an offline and online stage. In the offline stage, precomputations are performed once and for all to construct a simplified model. In the online stage, the simplified model is utilized to obtain the solution for an unseen parameter location. Usually the offline stage entails constructing a set of solutions to the FOM for different parameter locations. With this information one tries to construct a relation between the parameter locations and the respective solutions. Such construction can either be intrusive in which we use the underlying mathematical formulation of the FOM, or non-intrusive in which the construction is also based on observed data. The project-based reduced-basis (RB) method is a well known approach in which an RB is constructed using proper orthogonal decomposition (POD) and the singular value decomposition (SVD) [19]. The RB construction may also be performed by exploiting the Greedy Algorithm [20]. The RB maps the full-order problem onto a reduced-order space on which the problem is solved intrusively. In turn, the reduced solution is recovered to the full-order space. The strength

of the RB methods is motivated by various flow problems in [21, 22] complemented with a posteriori error bounds [20, 23, 24]. In non-linear problems or problems not satisfying the affine parametric dependence however, the projection-based RB method may perform inadequately as the simplified model is yet computationally demanding.

Recent advances in data-driven reduced-order modeling have overcome the issue of the still demanding computational cost to solve the problem in the reduced space. Being non-intrusive, a mapping is constructed between the parameters and the reduced-order solution. For example, in POD neural networks (POD-NN) the construction of an RB is still performed but the reduced solution is determined directly using neural networks [25]. Similarly, such mapping can be constructed using a Gaussian process regression model (GPR) [26] as is done in [27]. These mappings overcome the computational cost arising in non-affine parametric problems.

However, this does not take into account problems for which an accurate reduced basis is hard to construct. For that matter, in deep learning ROMs (DL-ROMs), one tries to construct a mapping from the parameters directly to the full-order solution using neural networks that anticipate well for non-linearities [28].

Alternative methods to reduce the computational cost in full-order models can be achieved by utilizing domain decomposition. Here, the domain is decomposed into smaller sub-regions that form a partition of the complete domain allowing for the solving of sub-problems. The sub-problems are coupled to form the full problem with coupling conditions forcing a smooth solution at the regions where the sub-problems connect. Restricting to non-overlapping domain decomposition methods, Schwarz iterative methods [29] as well as Schur complement methods [30] are well-known full-order techniques to solve coupled problems. In presence of parameters, linear-linear coupled problems have shown to be effectively solved in partitioned schemes while combining ROMs [31]. Domain decomposition methods can be extended to multi-scale physics problems where different physical models are used in different sub-regions that behave interactively. For instance, in fluid-structure interaction, one concurrently models both the fluid and the structure, allowing them to interact [32]. Similarly, in Micro Electro Mechanical Systems, the modeling involves capturing the interaction between the electrostatic field and the structural components [33].

In practical applications, coupled problems consist of a linear and non-linear sub-domain that are coupled via a non-overlapping interface paired with parameters, for example in modeling air flow around an obstacle [34]. In full-order models, the parametrized linear-nonlinear coupled problems must be solved using non-linear solvers such as Newton's method due to the presence of non-linearity, even though part of the problem is linear. Generally, non-linear solvers are computationally demanding and prone to instability. In order to tackle these drawbacks, ROMs that combine RB methods as well as the employment of domain decomposition have been thoroughly investigated [33–37]. As a result of domain decomposition, effective reduction can be applied on the linear and non-linear domain respectively. While RB methods may not guarantee satisfactory cost reduction, data-driven modeling overcomes this issue. In this thesis we propose new reduced-order surrogate models by not only combining RB methods and domain decomposition but also data-driven ROMs to solve parametrized linear-nonlinear coupled problems.

## 1.1 Motivation

As an example consider a steady-state heat conduction problem in a unit square domain $\Omega$. The domain is comprised of nine equal-sized non-overlapping blocks such that on each block we define a different constant thermal conductivity. This gives

$$\overline{\Omega} = \bigcup_{i=1}^{9} \overline{R}_i, \tag{1.1}$$

where $R_i$ is a subregion with thermal conductivity $\mu_i > 0$. The corresponding boundary $\partial\Omega$ is decomposed into four parts such that $\overline{\partial\Omega} = \overline{\Gamma_T \cup \Gamma_B \cup \Gamma_L \cup \Gamma_R}$. An illustration is given in Figure 1.1.

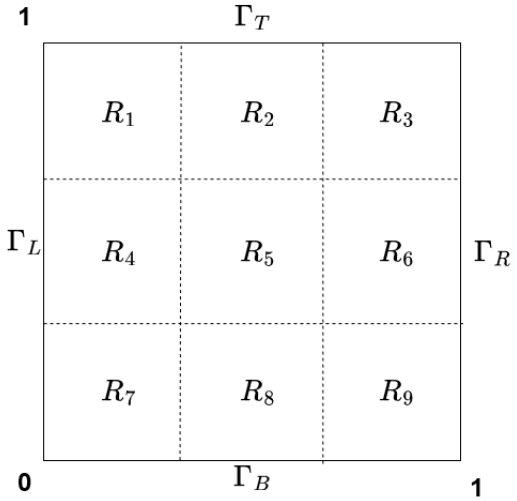



FIGURE 1.1: Domain of interest $\Omega$ with boundaries and nine subregions.

Let $\mathcal{N}_D$ be a non-linear diffusion operator given by

$$\mathcal{N}_D u = -\nabla \cdot (k(\boldsymbol{x};\boldsymbol{\mu}) D(\boldsymbol{x};\boldsymbol{\mu}) \nabla u) \tag{1.2}$$

with $D(\boldsymbol{x};\boldsymbol{\mu}) = 1 + u\mathbb{1}_{R_5}(\boldsymbol{x})$ where $\mathbb{1}_{R_5}(\boldsymbol{x})$ denotes the indicator function with respect to $R_5$ and the conductivity field

$$k(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{i=1}^{9} \mu_i \mathbb{1}_{R_i}(\boldsymbol{x}) \quad \text{with} \quad \boldsymbol{\mu} = [\mu_1 \ \mu_2 \ ... \ \mu_9]^T \in \mathcal{P} = [0.1 \ 1]^9. \tag{1.3}$$

The problem of interest is a steady-state Poisson problem given by

$$\begin{cases} -\mathcal{N}_D u = -6 & \text{in} \quad \Omega, \\ u = 1 + x^2 + 2y^2 & \text{on} \quad \Gamma_L \cup \Gamma_R, \\ \frac{\partial u}{\partial \boldsymbol{n}} = 4y & \text{on} \quad \Gamma_T \cup \Gamma_B, \end{cases} \tag{1.4}$$

to solve for the temperature $u(\boldsymbol{x};\boldsymbol{\mu})$. This means that linear Poisson equations are defined on the outer blocks of the domain with on the middle region $R_5$ a non-linear Poisson equation. The goal is to solve the problem for all parameter locations in the parameter space $\mathcal{P}$.

Treating (1.4) the conventional way as a non-linear problem leads to the requirement of non-linear PDE solvers such as Newton's method for every parameter location. Sustainable for a single parameter location, the exploration of the whole parameter space using traditional PDE solvers quickly becomes problematic due to the computational demands. While RB methods work well for linear problems, the reduction of computational cost here is unsatisfactory due the non-linearity in the middle region. As such, it is beneficial to decompose the domain such that the linear and non-linear domains are separated. In turn, RB methods can be applied on the linear domain while data-driven reduced-order modeling can be used on the non-linear domain.

## 1.2 Outline of thesis

The aim of this thesis is to construct alternative reduced-order surrogate models for parametrized linear-nonlinear coupled problems. These ROMs are intended to address the limitations of conventional full-order techniques that demand prohibitive computational costs. Before discussing the ROMs, we consider general background that is required throughout this thesis in Section 2. In Section 3 we explain two ROMs for parametrized linear-linear coupled problems that establish two ROMs for linear-nonlinear case. As a proof-of-concept, the results of the numerical experiments for a linear-linear coupled problem using the ROMs is compared with FEM in Section 4. In Section 5 and 6 we discuss results of numerical experiments for linear-nonlinear coupled problems and investigate the influence of slight modifications in the ROMs. To further motivate the use of reduced-order modeling, we report numerical results on uncertainty quantification using Monte Carlo simulation in Section 7. Regarding the numerical experiments we refer to Appendix A for the implementation of the code using FEniCS, an open-source computing platform for solving PDEs [38, 39].

# 2 Preliminaries

This section introduces the concepts of the finite element method, domain decomposition, reduced-order modeling, iterative solvers and Monte Carlo simulation for PDEs. We refer to [25, 40–49].

## 2.1 Finite element method

The finite element method (FEM) is a widely used full-order numerical method to solve PDEs that approximates an unknown function on a domain. Given a PDE, the general idea is to construct the corresponding discrete problem by means of a space discretization and divide the domain of interest into smaller discrete cells, also known as the finite elements. In the end, a linear system of equations is left to be solved. In this section we formulate the finite element approximation of the Poisson equation. Appearing in numerous applications such as in electrostatics, heat transfer and fluid dynamics, the Poisson equation forms an important equation of interest [49].

Let $\Omega \subset \mathbb{R}^2$ be an open bounded and connected set with Lipschitz-continuous boundary $\partial\Omega$. Consider the strong form of the Poisson equation given by

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{2.1}$$

with $g = 0$ and where $\Delta$ denotes the laplace operator and the second equation a homogeneous Dirichlet boundary condition. The equations model the displacement $u$ of a membrane due to a force with intensity $f$ with no displacement on the boundary. Other boundary conditions such as Neumann or Robin conditions are possible but not treated here. Due to the laplace operator, the solution $u$ is required to be twice differentiable and so $u \in \mathcal{C}^2(\Omega) \cap \mathcal{C}(\overline{\Omega})$ where $\overline{\Omega}$ denotes the closure of $\Omega$. Moreover, $f \in \mathcal{C}(\Omega)$. In physical cases, the solution $u$ does not necessarily satisfy the differentiability conditions imposed by the strong form. An alternative formulation of the problem is required to allow for less smooth solutions. This is called the weak form. It is obtained by multiplying (2.1) by a so-called arbitrary test function $v$ and integrating over the domain $\Omega$ and applying the Green formula for the divergence operator. This gives the following weak formulation:

$$\text{find} \quad u \in H_0^1(\Omega) : \int_\Omega \nabla u \cdot \nabla v \; d\Omega = \int_\Omega f v \; d\Omega \quad \forall v \in H_0^1(\Omega), \tag{2.2}$$

with $f \in L^2(\Omega)$, which is the space of square integrable functions

$$L^2(\Omega) := \{f : \Omega \longrightarrow \mathbb{R} \quad \text{such that} \quad \int_\Omega |f|^2 \; d\Omega < \infty\}$$

and consequently we introduce the Sobolev spaces of order 1, that is

$$H^1(\Omega) := \{u : \Omega \longrightarrow \mathbb{R} \quad \text{such that} \quad u, \; \frac{\partial u}{\partial x_i} \in L^2(\Omega), \; i = 1, 2\},$$
$$H_0^1(\Omega) := \{u \in H^1(\Omega) : u = 0 \quad \text{on} \quad \partial\Omega\}.$$

Using functional analysis, it can be proven that the solutions to the weak form are also solutions to the strong form in an almost everywhere setting. By letting $V = H_0^1(\Omega)$, the weak form (2.2) can be formulated in a more compact form with bilinear form

$$a : V \times V \longrightarrow \mathbb{R}, \; a(u, v) = \int_\Omega \nabla u \cdot \nabla v \; d\Omega$$

and functional

$$F : V \longrightarrow \mathbb{R},\ F(v) = \int_\Omega f v d\Omega.$$

This yields

$$\text{find} \quad u \in V : a(u, v) = F(v) \quad \forall v \in V \tag{2.3}$$

Thanks to the coercivity of and continuity of the bilinear form and continuity of the functional in (2.3), the existence and uniqueness of the weak form is guaranteed thanks to the Lax-Milgram Theorem, a well-known result from functional analysis [48].

### 2.1.1 Galerkin method

Having the weak formulation (2.3) at hand, we now aim to construct a discrete formulation. We start with a finite dimensional subspace depending on a parameter $h$:

$$V_h \subset V \quad \text{such that } \dim(V_h) = N_h < \infty \quad \forall h > 0.$$

Substituting the above in the weak form in (2.3) gives the so-called Galerkin Problem

$$\text{find} \quad u_h \in V_h : a(u_h, v_h) = F(v_h) \quad \forall v_h \in V_h. \tag{2.4}$$

It can be proven that this form still retains the useful properties such as existence, uniqueness, stability and convergence of the solution when $h \longrightarrow 0$. They are based on results from functional analysis. For a chosen basis $\{\varphi_1, \varphi_2, ..., \varphi_{N_h}\}$, we express the solution in terms of the basis functions:

$$u_h(\boldsymbol{x}) = \sum_{j=1}^{N_h} u_j \varphi_j(\boldsymbol{x})$$

and $v_h$ similarly. In this way, the Galerkin Problem can be reformulated such that

$$a(u_h, \varphi_i) = F(\varphi_i), \qquad i = 1, 2, ..., N_h \tag{2.5}$$

giving rise to the linear system

$$\mathbb{A}\boldsymbol{u} = \boldsymbol{f} \tag{2.6}$$

with stiffness matrix $\mathbb{A} \in \mathbb{R}^{N_h \times N_h}$ and $\boldsymbol{f} \in \mathbb{R}^{N_h}$ whose elements are given by

$$a_{ij} = \int_\Omega \nabla\varphi_j \nabla\varphi_i \, d\Omega \quad \text{and} \quad f_i = \int_\Omega f \varphi_i \, d\Omega.$$

### 2.1.2 Finite elements

With the Galerking Problem in (2.4) at hand, the remaining question is what kind of basis to choose. Ideally, a basis that promotes sparsity in matrix $\mathbb{A}$ in (2.6) ensures a low computational cost. Before constructing a suitable basis, we will first discretize the domain with a cover of triangular elements such that

$$\Omega_h = \text{int}\left( \bigcup_{K \in \mathcal{T}_h} K \right),$$

where $\mathcal{T}_h$ is a partition consisting of elements $K$ that make up $\Omega$ and int denotes the interior part of the set. $\mathcal{T}_h$ is often referred to as the mesh. The parameter $h$ refers to the spacing of the mesh.

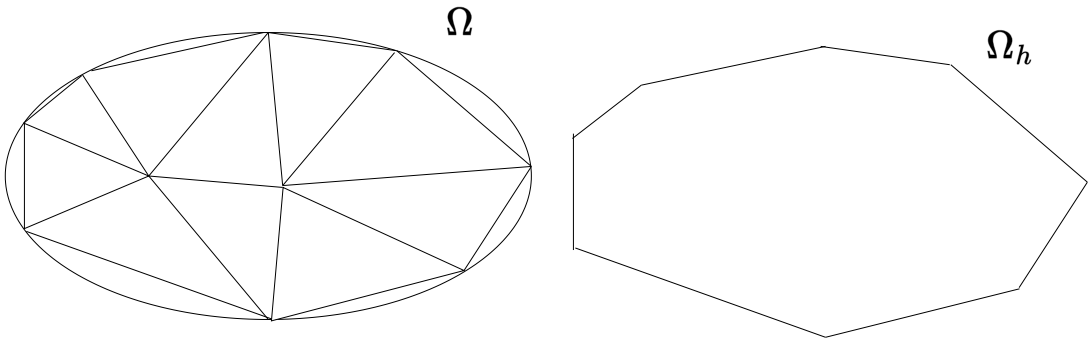



FIGURE 2.1: Possible mesh $\Omega_h$ to approximate the original domain $\Omega$.

Let $\mathbb{P}_1$ denote the space of polynomials up to degree 1, that is

$$\mathbb{P}_1 = \{p(x_1, x_2) = a + bx_1 + cx_2, \text{ with } a, b, c \in \mathbb{R}\}.$$

We introduce the finite element space

$$X_h = \{v_h \in \mathcal{C}^0(\overline{\Omega}) : v_h\big|_K \in \mathbb{P}_1 \ \forall K \in \mathcal{T}_h\}, \tag{2.7}$$

which is the space of continuous polynomials of degree 1 on $\overline{\Omega}$ on single triangles of the triangulation $\mathcal{T}_h$. Next, we define

$$\overset{\circ}{X}_h = \{v_h \in X_h : v_h\big|_{\partial\Omega} = 0\}. \tag{2.8}$$

The latter forms an approximation of $H_0^1(\Omega)$ and therefore by setting $V_h = \overset{\circ}{X}_h$, the finite element problem becomes

$$\text{find} \quad u_h \in V_h : \int_\Omega \nabla u_h \cdot \nabla v_h \, d\Omega = \int_\Omega f v_h \, d\Omega \quad \forall v_h \in V_h. \tag{2.9}$$

As a basis for $V_h$ we choose the characteristic Lagrangian functions $\varphi_j \in V_h$, $j = 1, ..., N_h$ such that

$$\varphi_j(\boldsymbol{N_i}) = \delta_{ij} = \begin{cases} 0 & \text{if} \quad i \neq j, \\ 1 & \text{if} \quad i = j, \end{cases} \qquad i, j = 1, ..., N_h \tag{2.10}$$

In turn, the functions $u_h$ and $v_h$ can be expressed by a linear combination of the basis vectors:

$$v_h(\boldsymbol{x}) = \sum_{i=1}^{N_h} v_i \varphi_i(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \Omega \text{ with } v_i = v_h(\boldsymbol{N}_i) \tag{2.11}$$

$$u_h(\boldsymbol{x}) = \sum_{j=1}^{N_h} u_j \varphi_j(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \Omega \text{ with } u_j = u_h(\boldsymbol{N}_j) \tag{2.12}$$

Inserting (2.11) and (2.12) into the finite element problem in (2.9) yields the linear system in (2.6). By choice of the basis functions, the stiffness matrix $\mathbb{A}$ is sparse.

## 2.2 Multidomain formulation of the finite element approximation

In multi-physics, there exist multiple partial differential equations (PDEs) defined in different subdomains of the computational domain [32, 33]. For these problems, numerical methods such as the finite element method (FEM) allow for a discretization of the complete domain. As such, it may be desirable to divide the computational domain into subregions on which each subregion a single PDE is defined, while taking care of the boundaries. This allows for solving smaller subregions and thereby possibly saving the total computational cost. This concept is known as domain decomposition. Decomposing a domain can be done using overlapping and non-overlapping subdomains [29, 30]. Here we concentrate on the latter. We will start with a finite element approximation of the Poisson equation in (2.1) on a single domain and then split the domain in two subregions to construct an equivalent formulation [41].

Let $\Omega \in \mathbb{R}^2$ be the region that is subdivided into two non-overlapping subdomains such that

$$\overline{\Omega} = \overline{\Omega_1 \cup \Omega_2}, \qquad \overline{\Omega_1 \cap \Omega_2} = \emptyset, \qquad \Gamma = \Omega_1 \cap \Omega_2. \tag{2.13}$$

The boundary $\Gamma$ that connects the two domains is often referred to as the interface. See Figure 2.2 below.

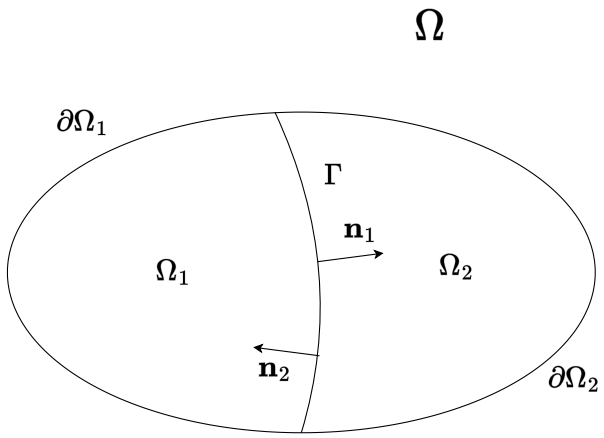



FIGURE 2.2: A non-overlapping domain decomposition of $\Omega$.

The multidomain formulation in which $u_i = u\big|_{\Omega_i}$, $i = 1, 2$ is as follows:

$$\begin{cases} -\Delta u_1 = f & \text{in } \Omega_1, \\ u_1 = 0 & \text{on } \partial\Omega_1 \backslash \Gamma, \\ -\Delta u_2 = f & \text{in } \Omega_2, \\ u_2 = 0 & \text{on } \partial\Omega_2 \backslash \Gamma, \\ u_1 = u_2 & \text{on } \Gamma, \quad \text{(continuity of the interface)} \\ \frac{\partial u_1}{\partial n} = \frac{\partial u_2}{\partial n} & \text{on } \Gamma. \quad \text{(continuity of normal derivatives)} \end{cases} \tag{2.14}$$

The last two equations in (2.14) are the so-called interface conditions and are imposed to have continuity on the boundary. The nodes on domain $\Omega$ are partitioned into sets such

that $\{x_j^{(1)}, 1 \le j \le N_1\}$, $\{x_j^{(2)}, 1 \le j \le N_2\}$, $\{x_j^{(\Gamma)}, 1 \le j \le N_\Gamma\}$ contain the nodes on domain $\Omega_1$, $\Omega_2$ and the interface respectively. Similarly, we split the basis functions such that $\varphi_j^{(1)}$, $\varphi_j^{(2)}$ and $\varphi_j^{(\Gamma)}$ correspond to the nodes on the respective domain. Consequently, the finite element problem in (2.9) can be reformulated: find $u_h \in V_h$ such that

$$\begin{cases} a(u_h, \varphi_i^{(1)}) = F(\varphi_i^{(1)}) & \forall i = 1, ..., N_1, \\ a(u_h, \varphi_i^{(2)}) = F(\varphi_i^{(2)}) & \forall i = 1, ..., N_2, \\ a(u_h, \varphi_i^{(\Gamma)}) = F(\varphi_i^{(\Gamma)}) & \forall i = 1, ..., N_\Gamma. \end{cases} \tag{2.15}$$

Let

$$a_i(u_h, v_h) = \int_{\Omega_i} \nabla u_h \nabla v_h d\Omega \qquad \forall u_h, v_h \in V_{i,h},\ i = 1, 2 \tag{2.16}$$

be the linear forms respective to domain $i$ with suitable function space $V_{i,h} = \{v \in H^1(\Omega_i) \big| v = 0 \text{ on } \partial\Omega_i \backslash \Gamma\}$ $(i = 1, 2)$. The problem is reformulated again: find $u_h^{(1)} \in V_{1,h}$, $u_h^{(2)} \in V_{2,h}$ such that

$$\begin{cases} a_1(u_h^{(1)}, \varphi_i^{(1)}) = F_1(\varphi_i^{(1)}) & \forall i = 1, ..., N_1, \\ a_2(u_h^{(2)}, \varphi_j^{(2)}) = F_2(\varphi_j^{(2)}) & \forall j = 1, ..., N_2, \\ a_1(u_h^{(1)}, \varphi_k^{(\Gamma)}) + a_2(u_h^{(2)}, \varphi_k^{(\Gamma)}) = F_1(\varphi_k^{(\Gamma)}) + F_2(\varphi_k^{(\Gamma)}) & \forall k = 1, ..., N_\Gamma. \end{cases} \tag{2.17}$$

Next, we rewrite the solution vector $u_h$ in terms of the basis functions on their respective domain such that

$$u_h(x) = \sum_{j=1}^{N_1} u_h(x_j^{(1)}) \varphi_j^{(1)}(x) + \sum_{j=1}^{N_2} u_h(x_j^{(2)}) \varphi_j^{(2)}(x) + \sum_{j=1}^{N_\Gamma} u_h(x_j^{(\Gamma)}) \varphi_j^{(\Gamma)}(x). \tag{2.18}$$

To ease notation, we let $u_h(x_j^{(\alpha)}) = u_j^{(\alpha)}$ for $\alpha = 1, 2, \Gamma$ and $j = 1, ..., N_\alpha$. Substituting the rewritten solution vector into (2.17) gives

$$\begin{cases} \sum_{j=1}^{N_1} u_j^{(1)} a_1(\varphi_j^{(1)}, \varphi_i^{(1)}) + \sum_{j=1}^{N_\Gamma} u_j^{(\Gamma)} a_1(\varphi_j^{(\Gamma)}, \varphi_i^{(1)}) = F_1(\varphi_i^{(1)}) & \forall i = 1, ..., N_1, \\ \sum_{j=1}^{N_2} u_j^{(2)} a_2(\varphi_j^{(2)}, \varphi_i^{(2)}) + \sum_{j=1}^{N_\Gamma} u_j^{(\Gamma)} a_2(\varphi_j^{(\Gamma)}, \varphi_i^{(2)}) = F_2(\varphi_i^{(2)}) & \forall i = 1, ..., N_2, \\ \sum_{j=1}^{N_\Gamma} u_j^{(\Gamma)} \left( a_1(\varphi_j^{(\Gamma)}, \varphi_i^{(\Gamma)}) + a_2(\varphi_j^{(\Gamma)}, \varphi_i^{(\Gamma)}) \right) & \\ + \sum_{j=1}^{N_1} u_j^{(1)} a_1(\varphi_j^{(1)}, \varphi_i^{(\Gamma)}) + \sum_{j=1}^{N_2} u_j^{(2)} a_2(\varphi_j^{(2)}, \varphi_i^{(\Gamma)}) & \\ = F_1(\varphi_i^{(\Gamma)}\big|_{\Omega_1}) + F_2(\varphi_i^{(\Gamma)}\big|_{\Omega_2}) & \forall i = 1, ..., N_\Gamma. \end{cases} \tag{2.19}$$

We will now formulate the algebraic formulation by introducing the following arrays:

$$
\begin{aligned}
&(\mathbb{A}_{11})_{ij} = a_1(\varphi_j^{(1)}, \varphi_i^{(1)}), && (\mathbb{A}_{1\Gamma})_{ij} = a_1(\varphi_j^{(\Gamma)}, \varphi_i^{(1)}),\\
&(\mathbb{A}_{22})_{ij} = a_2(\varphi_j^{(2)}, \varphi_i^{(2)}), && (\mathbb{A}_{2\Gamma})_{ij} = a_2(\varphi_j^{(\Gamma)}, \varphi_i^{(2)}),\\
&(\mathbb{A}_{\Gamma\Gamma}^{(1)})_{ij} = a_1(\varphi_j^{(\Gamma)}, \varphi_i^{(\Gamma)}), && (\mathbb{A}_{\Gamma\Gamma}^{(2)})_{ij} = a_2(\varphi_j^{(\Gamma)}, \varphi_i^{(\Gamma)}),\\
&(\mathbb{A}_{\Gamma 1})_{ij} = a_1(\varphi_j^{(1)}, \varphi_i^{(\Gamma)}), && (\mathbb{A}_{\Gamma 2})_{ij} = a_2(\varphi_j^{(2)}, \varphi_i^{(\Gamma)}),\\
&(\boldsymbol{f}_1)_i = F_1(\varphi_i^{(1)}), && (\boldsymbol{f}_2)_i = F_2(\varphi_i^{(2)}),\\
&(\boldsymbol{f}_\Gamma^{(1)})_i = F_1(\varphi_i^{(\Gamma)}), && (\boldsymbol{f}_\Gamma^{(2)})_i = F_2(\varphi_i^{(\Gamma)}).
\end{aligned}
$$

Moreover, let

$$
\boldsymbol{u}_1 = u_j^{(1)},\ \boldsymbol{u}_2 = u_j^{(2)} \text{ and } \boldsymbol{u}_\Gamma = u_j^{(\Gamma)} \text{such that } \boldsymbol{u} = (\boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{u}_\Gamma)^T \tag{2.20}
$$

giving the linear system

$$
\mathbb{A}\boldsymbol{u} = \begin{bmatrix} \mathbb{A}_{11} & 0 & \mathbb{A}_{1\Gamma} \\ 0 & \mathbb{A}_{22} & \mathbb{A}_{2\Gamma} \\ \mathbb{A}_{\Gamma 1} & \mathbb{A}_{\Gamma 2} & \mathbb{A}_{\Gamma\Gamma}^{(1)} + \mathbb{A}_{\Gamma\Gamma}^{(2)} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{u}_2 \\ \boldsymbol{u}_\Gamma \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_\Gamma^{(1)} + \boldsymbol{f}_\Gamma^{(2)} \end{bmatrix} \tag{2.21}
$$

This linear system is in fact equivalent to the linear system in (2.6) with characteristic Lagrangian functions of first order. It gives a rearrangement of the linear system such that all DoFs belonging to each domain are stacked together. This means that $\boldsymbol{u}_1$ contains all DoFs that belong strictly to domain 1, $\boldsymbol{u}_2$ the DoFs that belong strictly to domain 2 and $\boldsymbol{u}_\Gamma$ the DoFs that belong to the interface.

Rather than solving the complete linear system in (2.21) at once, we can use the block structure to solve multiple smaller problems to save computational cost. The first two lines of (2.21) can be interpreted as two separate discretizations on domain $\Omega_1$ and $\Omega_2$ respectively, with common value $\boldsymbol{u}_\Gamma$ as a Dirichlet boundary condition on the interface. On the other hand we can obtain the so-called traction force (or flux) [41] relation from the third line in (2.21):

$$
\boldsymbol{t}_\Gamma := \mathbb{A}_{\Gamma 1}\boldsymbol{u}_1 + \mathbb{A}_{11}^{(1)}\boldsymbol{u}_\Gamma - \boldsymbol{f}_\Gamma^{(1)} = \boldsymbol{f}_\Gamma^{(2)} - \mathbb{A}_{\Gamma 2}\boldsymbol{u}_2 - \mathbb{A}_{\Gamma\Gamma}^{(2)}\boldsymbol{u}_\Gamma. \tag{2.22}
$$

The traction force relates the interaction between the two domains. The first line of (2.21) is a discretization for domain $\Omega_1$ with a Neumann boundary condition of $\boldsymbol{t}_\Gamma$ on the interface. Similarly the second line is a discretization for domain $\Omega_2$ with Neumann data $\boldsymbol{t}_\Gamma$ on the interface. Schur complement methods use these different perspectives to solve smaller subsystems and thereby saving computational cost.

## 2.3 Singular value decomposition

In linear algebra, the singular value decomposition (SVD) is a matrix factorization process allowing for low-rank approximations. This is useful in the applications of image and data compression [42].

**Definition 2.1** (Singular Value Decomposition)**.** *If $\mathbb{A} \in \mathbb{R}^{m\times n}$ is a real matrix, there exist two orthogonal matrices*

$$
\mathbb{U} = [\boldsymbol{\zeta}_1| \ ... \ |\boldsymbol{\zeta}_m] \in \mathbb{R}^{m\times m}, \quad \mathbb{Z} = [\boldsymbol{\psi}_1| \ ... \ |\boldsymbol{\psi}_n] \in \mathbb{R}^{n\times n}
$$

*such that*

$$\mathbb{A} = \mathbb{U}\Sigma\mathbb{Z}^T, \quad \textit{with} \quad \Sigma = \textit{diag}(\sigma_1, ..., \sigma_p) \in \mathbb{R}^{m\times n} \tag{2.23}$$

*and* $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0$*, for* $p = \min(m, n)$*.*

*Proof.* See [42]. □

With the following core result, the best low-rank approximation of a matrix can be guaranteed.

**Definition 2.2** (Schmidt-Eckart-Young)**.** *Given a matrix* $\mathbb{A} \in \mathbb{R}^{m\times n}$ *of rank* $r$*, the matrix*

$$\mathbb{A}_k = \sum_{i=1}^{k} \sigma_i \boldsymbol{\zeta}_i \boldsymbol{\psi}_i^T, \quad 0 \leq k \leq r,$$

*satisfies the optimality property*

$$\|\mathbb{A} - \mathbb{A}_k\|_F = \min_{\substack{\mathbb{B}\in\mathbb{R}^{m\times n} \\ rank(\mathbb{B}\leq k)}} \|\mathbb{A} - \mathbb{B}\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}$$

*Proof.* See [42]. □

## 2.4 Reduced-order modeling for parametrized PDEs

Consider a well-posed parametrized PDE that is solved using the numerical scheme of FEM. Let $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$ be the parameter vector in parameter space $\mathcal{P}$. For a given parameter location $\boldsymbol{\mu}$, the finite element approximation is given by

$$\mathbb{A}_h(\boldsymbol{\mu})\boldsymbol{u}_h(\boldsymbol{\mu}) = \boldsymbol{f}_h(\boldsymbol{\mu}) \tag{2.24}$$

with $\mathbb{A}_h(\boldsymbol{\mu}) \in \mathbb{R}^{N_h\times N_h}$ and $\boldsymbol{f}_h(\boldsymbol{\mu}) \in \mathbb{R}^{N_h}$. When aiming to solve (2.24) over the complete parameter space, we are required to solve the equation repeatedly for every parameter location. For large parameter spaces this can quickly become unsustainable. Over the past decades, ROMs have been studied to overcome this issue. The main idea of reduced-order modeling for parametrized PDEs is to construct a relation between the parameters and their solutions, often referred to as the parametric map. This translates to discovering the collection of all solutions to the parametrized PDE under variation of the parameters, also known as the solution manifold given by (in algebraic form):

$$\mathcal{M}_h = \{\boldsymbol{u}_h(\boldsymbol{\mu})\}_{\boldsymbol{\mu}\in\mathcal{P}} \tag{2.25}$$

The manifold holds information on how the parameters influence their corresponding solution. Typically, reduced order modeling methods consist of two stages:

1. The offline stage
   This is the stage in which we perform precomputations usually related to an approximation of the solution manifold that are needed to construct the ROM. These computations are done once and for all.

2. The online stage
   For a new parameter input we exploit the precomputations from the offline stage to construct and solve the ROM

Two projection-based ROMs will be explained that revolve around a proper orthogonal decomposition (POD).

### 2.4.1 Intrusive reduced-order modeling with POD

Consider the problem described in (2.24). Typically in full-order models, the size $N_h$ is large. In POD reduced order modeling (POD-ROM) the full-order problem is projected onto a lower-order space using a constructed POD basis using an approximated solution manifold. The resulting reduced-order problem of size $N$ such that $N \ll N_h$. In turn the reduced-order problem is solved and its solution is projected onto the full-order space.

**Offline stage**
The solution manifold introduced in (2.25) is unattainable because the problem would be solved otherwise. Instead we generate a collection of solutions (also called snapshots) for different parameter locations that acts as an approximation to the solution manifold. For $n_s$ different parameter locations that are stored in $\Xi = [\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}] \in \mathbb{R}^{n_s \times p}$, we construct the corresponding snapshot matrix $\mathbb{S} \in \mathbb{R}^{N_h \times n_s}$ such that

$$\mathbb{S} = [\boldsymbol{u}_h(\boldsymbol{\mu}^1) \mid \boldsymbol{u}_h(\boldsymbol{\mu}^2) \mid ... \mid \boldsymbol{u}_h(\boldsymbol{\mu}^{n_s}) ] \tag{2.26}$$

contains the high-fidelity solutions for each parameter vector corresponding in $\Xi$. The snapshot matrix acts as an approximation to the solution manifold. We wish to extract as much information as possible of $\mathbb{S}$ while using only a few basis vectors. This translates to finding a low-rank approximation and is done by performing the SVD, allowing to represent the snapshot matrix as

$$\mathbb{S} = \mathbb{U}\Sigma\mathbb{Z}^T \tag{2.27}$$

with $\mathbb{U} = [\boldsymbol{\zeta}_1 \mid \boldsymbol{\zeta}_2 \mid ... \mid \boldsymbol{\zeta}_{N_h}]$ containing the left singular vectors, $\Sigma = \text{diag}(\sigma_1, \sigma_2, ..., \sigma_r)$ with $r = \min\{N_h, n_s\}$ containing the singular values in decreasing order and $\mathbb{Z}^T$ containing the right singular vectors. The POD basis $\mathbb{V} \in \mathbb{R}^{N_h \times N}$ is given by taking the first $N$ columns of $\mathbb{U}$.

**Online stage**
Given a new parameter location $\boldsymbol{\mu}$ we construct the FOM giving (2.24) (without solving it) and approximate the high-fidelity solution vector by

$$\boldsymbol{u}_h(\boldsymbol{\mu}) \approx \mathbb{V}\boldsymbol{u}_N(\boldsymbol{\mu}) \tag{2.28}$$

with $\boldsymbol{u}_N(\boldsymbol{\mu}) \in \mathbb{R}^N$ the so-called reduced-basis (RB) coefficients or modes. We left-multiply (2.24) by $\mathbb{V}^T$ and substitute (2.28). This gives

$$\mathbb{A}_N(\boldsymbol{\mu})\boldsymbol{u}_N(\boldsymbol{\mu}) = \boldsymbol{f}_N(\boldsymbol{\mu}) \tag{2.29}$$

with $\mathbb{A}_N(\boldsymbol{\mu}) = \mathbb{V}^T\mathbb{A}_h(\boldsymbol{\mu})\mathbb{V}\boldsymbol{u}_N(\mu) \in \mathbb{R}^{N \times N}$ and $\boldsymbol{f}_N(\boldsymbol{\mu}) = \mathbb{V}^T\boldsymbol{f}_h(\boldsymbol{\mu}) \in \mathbb{R}^N$. Notice that (2.29) is a system of only $N \times N$ equations whereas (2.24) is $N_h \times N_h$. We solve (2.29) to obtain the RB coefficients and use (2.28) to project to the higher order space.

The POD-ROM procedure is summarized in the Algorithm 2.1.

**Algorithm 2.1** POD-ROM

1: **Offline Stage**

2: Create snapshot matrix $\mathbb{S} = [\boldsymbol{u}_h(\boldsymbol{\mu}^1) \mid \boldsymbol{u}_h(\boldsymbol{\mu}^2) \mid ... \mid \boldsymbol{u}_h(\boldsymbol{\mu}^{n_s}) ]$ corresponding to random parameter locations $\Xi = [\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}]$

3: Perform an SVD on $\mathbb{S}$: $\mathbb{U}$ and $\Sigma$ obtained

4: Take $N$ columns of $\mathbb{U}$: $\mathbb{V}$ obtained

5: **Online Stage**

6: For the new parameter location $\boldsymbol{\mu}$, construct the FOM: $\mathbb{A}_h(\boldsymbol{\mu})\boldsymbol{u}_h(\boldsymbol{\mu}) = \boldsymbol{f}_h(\boldsymbol{\mu})$

7: Let $\boldsymbol{u}_h(\boldsymbol{\mu}) \approx \mathbb{V}\boldsymbol{u}_N(\boldsymbol{\mu})$ and left-multiply the FOM by $\mathbb{V}^T$ to get

$$\mathbb{A}_N(\boldsymbol{\mu})\boldsymbol{u}_N(\boldsymbol{\mu}) = \boldsymbol{f}_N(\boldsymbol{\mu})$$

and solve for $\boldsymbol{u}_N(\boldsymbol{\mu})$.

8: Project $\boldsymbol{u}_N(\boldsymbol{\mu})$ to the full-order space.

Left unanswered in the construction of the POD basis is to choose how many vectors should be included. Choosing too few basis vectors will result in relatively large errors compared to taking more basis vectors, but this comes with the price of a larger computational cost. Closely related to the Eckart-Young Theorem, consider the following result:

**Proposition 2.1.** *Let* $\mathcal{V}_N = \{\mathbb{W} \in \mathbb{R}^{N_h \times N} : \ \mathbb{W}^T\mathbb{W} = \mathbb{I}_N\}$ *be the set of all* $N$*-dimensional orthonormal bases. Then,*

$$\sum_{i=1}^{n_s} \|\boldsymbol{u}_i - \mathbb{V}\mathbb{V}^T\boldsymbol{u}_i\|_2^2 = \min_{\mathbb{W}\in\mathcal{V}_N} \sum_{i=1}^{n_s} \|\boldsymbol{u}_i - \mathbb{W}\mathbb{W}^T\boldsymbol{u}_i\|_2^2 = \sum_{i=N+1}^{r} \sigma_i^2.$$

*Proof.* See [42]. □

In other words, the error in the POD basis compared to the snapshot matrix is equal to the sum of the squares of the singular values corresponding to the left-singular vectors that are not included in the construction of $\mathbb{V}$. Given a chosen tolerance $\epsilon_{\text{POD}}$, the relative information content of the POD basis is given by

$$I(N) = \frac{\sum_{i=1}^{N} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \geq 1 - \epsilon_{\text{POD}}^2. \tag{2.30}$$

The higher the relative information content, the more information from the snapshot matrix is included in the POD basis. This allows us to choose a suitable $N$ in the construction of the POD basis. In order to get desired reduction in computational cost while maintaining good accuracy, we need the solution manifold $\mathcal{M}_h$ to be low-rank approximable. This translates to a fast decay of the singular values in the SVD. If this property is not satisfied the POD-ROM remains computationally expensive.

The construction of (2.29) has a complexity depending on $N_h$. In particular problems, we can exploit the so-called affine parametric dependence property. This means we can remove the dependency of $\boldsymbol{\mu}$ on $\mathbb{A}_N(\boldsymbol{\mu})$ and $\boldsymbol{f}_N(\boldsymbol{\mu})$ and write them as a linear combination. The low-fidelity problem becomes

$$\sum_{q=1}^{Q_a} \left(\theta_a^q(\boldsymbol{\mu})\mathbb{A}_N^q\right) \boldsymbol{u}_N(\boldsymbol{\mu}) = \sum_{q'=1}^{Q_f} \theta_f^{q'}(\boldsymbol{\mu})\boldsymbol{f}_N^{q'} \tag{2.31}$$

where $\{\theta_a^q(\boldsymbol{\mu})\}_{q=1}^{Q_a}$ and $\{\theta_f^{q'}(\boldsymbol{\mu})\}_{q'=1}^{Q_f}$ are two sets of $Q_a$ (and $Q_f$ respectively) scalar functions $\theta_a^q(\boldsymbol{\mu})$, $\theta_f^{q'}(\boldsymbol{\mu}) : \mathcal{P} \longrightarrow \mathbb{R}$. In this way, the matrices and vectors $\mathbb{A}_N^q$ (for $q = 1, ..., Q_a$) and $\boldsymbol{f}_N^{q'}$ (for $q' = 1, ..., Q_f$) are computed once and for all in the offline stage which saves a substantial amount of computational cost for every calculation in the online stage. When this property is not satisfied, the online stage remains computationally expensive. Using POD and neural networks, we can overcome this issue.

### 2.4.2 Non-intrusive reduced-order modeling with POD and neural networks

Before explaining the reduced-order method using neural networks, we give a brief explanation of neural networks.

#### Neural Networks

A neural network consists of multiple layers that allow for a mapping between two state spaces. Consider $V_1 \in \mathbb{R}^{n_1}$ and $V_2 \in \mathbb{R}^{n_2}$ and an activation function $\rho : \mathbb{R} \longrightarrow \mathbb{R}$. A layer $L$ is a mapping $L : V_1 \longrightarrow V_2$ such that

$$L(\boldsymbol{v}) = \rho(\mathbb{W}\boldsymbol{v} + \boldsymbol{b}) \tag{2.32}$$

with weights $\mathbb{W} \in \mathbb{R}^{n_1 \times n_2}$ and biases $\boldsymbol{b} \in \mathbb{R}^{n_2}$. A neural network with $l \geq 0$ hidden layers is a mapping of the form

$$\Phi := L_{l+1} \circ L_l \circ ... \circ L_1 \tag{2.33}$$

with $L_i : \mathbb{R}^{n_i} \longrightarrow \mathbb{R}^{n_{i+1}}$ for $i = 1, 2, ..., l$. The depth of a neural network is defined as $l$.

#### Proper orthogonal decomposition and neural networks

Instead of solving the reduced-order linear system in (2.29) for POD-ROM in the online stage, we directly approximate the mapping from the parameters to the RB coefficients. Consider the mapping

$$\pi : \mathbb{R}^p \longrightarrow \mathbb{R}^N \quad \text{such that} \quad \pi : \boldsymbol{\mu} \longrightarrow \mathbb{V}^T \boldsymbol{u}_h(\boldsymbol{\mu}). \tag{2.34}$$

The high-fidelity solution is then recovered by left-multiplying by $\mathbb{V}$:

$$\boldsymbol{u}_h(\boldsymbol{\mu}) = \mathbb{V}\pi(\boldsymbol{\mu}) \tag{2.35}$$

We approximate the mapping in (2.34) by $\hat{\pi}(\boldsymbol{\mu})$ which is trained using a neural network. In this way, the offline stage consists of the construction of $\mathbb{V}$ as well as training the neural network for $\hat{\pi}(\boldsymbol{\mu})$. In the online stage, we only have to use (2.35) to recover the solution. The method is abbreviated with POD-NN. See Algorithm 2.2.

**Algorithm 2.2** POD-NN

1: **Offline Stage**
2: Do POD algorithm (see Offline Stage in Algorithm 2.1): $\mathbb{S}$, $\Xi$ and $\mathbb{V}$ obtained
3: Create a database $\{\mathbb{V}^T \boldsymbol{u}_h(\boldsymbol{\mu})\}_{i=1}^{n_s}$ for the outputs of the neural network
4: Train the mapping $\pi(\boldsymbol{\mu})$ using a neural network: $\hat{\pi}(\boldsymbol{\mu})$ obtained
5: **Online Stage**
6: For a new parameter location $\boldsymbol{\mu}$, we approximate the high-fidelity solution by $\mathbb{V}\hat{\pi}(\boldsymbol{\mu})$.

## 2.5 Iterative solvers for minimization problems

Consider $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ and $f \in \mathcal{C}^1$ and its minimization problem of the form

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \tag{2.36}$$

The gradient descent algorithm is a first-order method to solve the minimization problem above:

**Algorithm 2.3** Gradient Descent

**Inputs** : $f$, $\boldsymbol{x}_0$, $\gamma$, $\epsilon$
**Output** : $\boldsymbol{x}_k$
Set $\delta := \epsilon$
Repeat until $\delta < \epsilon$:
    $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \gamma \nabla f(\boldsymbol{x}_k)$
    $\delta = \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2$
    $\boldsymbol{x}_k = \boldsymbol{x}_{k+1}$

In every iteration $k$ at the given point $x_k$, one moves into the oppositve direction of the gradient with a fixed step size $\gamma$. In practice, finding a suitable step size is an important problem. A step size too small would lead to slow convergence, while a step size too large might lead to divergence. Given a descent direction $\boldsymbol{p}$, line search algorithms find a suitable step size to move into that direction. A common line search algorithm is the backtracking line search algorithm:

**Algorithm 2.4** Backtracking Line Search

**Inputs** : $f$, $\boldsymbol{x}$, $\boldsymbol{p}$, $\gamma_0 > 0$ and control parameters $\tau, c \in (0, 1)$
**Output** : $\gamma_k$
Set $m = \nabla f(\boldsymbol{x})^T \boldsymbol{p}$ and $t = -cm$
Repeat until $f(\boldsymbol{x}) - f(\boldsymbol{x} + \gamma_k \boldsymbol{p}) \geq \gamma_k t$:
    $\gamma_{k+1} = \tau \gamma_k$
    $\gamma_k = \gamma_{k+1}$

Another method approximate (2.36) is the Broyden-Fletcher-Goldbarf-Shanno algorithm (BFGS) in which we approximate the inverse of the Hessian iteratively:

**Algorithm 2.5** BFGS

**Inputs** : $f$, $\boldsymbol{x}_0$, $\mathbb{H}_0$, $\epsilon$
**Output** : $\boldsymbol{x}_k$
Set $\boldsymbol{y}_k = \epsilon$
Repeat until $\|\boldsymbol{y}_k\|_2 < \epsilon$:
    Solve $\boldsymbol{p}_k = -\mathbb{H}_k \nabla f(\boldsymbol{x}_k)$
    Find a suitable step size $\gamma_k$ using a line-search algorithm
    Set $\boldsymbol{s}_k = \gamma_k \boldsymbol{p}_k$ and update $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$
    Set $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$
    $\mathbb{H}_{k+1} = \mathbb{H}_k + \frac{(\boldsymbol{s}_k^T \boldsymbol{y}_k + \boldsymbol{y}_k^T \mathbb{H}_k \boldsymbol{y}_k)(\boldsymbol{s}_k \boldsymbol{s}_k^T)}{(\boldsymbol{s}_k^T \boldsymbol{s}_k)^2} - \frac{\mathbb{H}_k \boldsymbol{y}_k \boldsymbol{s}_k^T + \boldsymbol{s}_k \boldsymbol{y}_k^T \mathbb{H}_k}{\boldsymbol{s}_k^T \boldsymbol{y}_k}$.
    $\mathbb{H}_k = \mathbb{H}_{k+1}$
    $\boldsymbol{x}_k = \boldsymbol{x}_{k+1}$

## 2.6 Newton's method for finding roots

Consider $F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$ and $F \in \mathcal{C}^1$ and its rootfinding problem

$$F(\boldsymbol{x}) = \mathbf{0} \tag{2.37}$$

This problem can be solved using Newton's Method for finding roots:

**Algorithm 2.6** Newton's Method for finding roots

1: **Inputs** : $F$, $\boldsymbol{x}_0$, $\epsilon$
**Output** : $\boldsymbol{x}_k$
2: Set $\delta = \epsilon$
3: Repeat until $\delta < \epsilon$:
4: Solve $\mathbb{J}_F(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = -F(\boldsymbol{x}_k)$ for $(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$
5: $\delta = \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2$
6: $\boldsymbol{x}_k = \boldsymbol{x}_{k+1}$

## 2.7 Forward uncertainty quantification of PDEs using Monte Carlo simulation

In the application of PDEs in fields such as physics and biology, the inputs required for the real physical model are usually measured and therefore contain uncertainty. The inaccurate inputs propagate through the model and may lead to large errors. In uncertainty quantification, the task is to obtain statistical information of the output of the model given the uncertainties in the inputs. This can be of great importance in high-impact decisions such as in hurricane forecasting. Usually, it is not just the solution to the PDE that is of interest but rather another output of interest using the solution. This is called the output of interest (OoI).

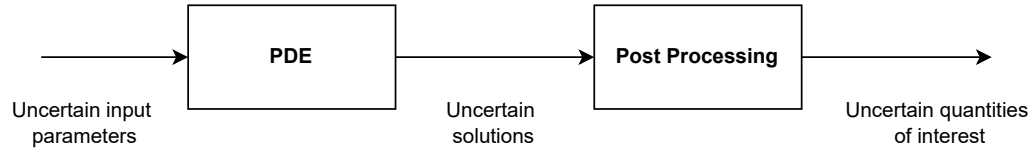



FIGURE 2.3: Uncertain inputs propagate through the model leading to uncertain quantities of interest.

Consider the non-linear problem in (1.4) and suppose the OoI is given by a function of the solution, that is $F(u)$. Let the quantity of interest (QoI) be the statistical information of interest denoted by $G$ such that

$$\text{QoI} = \int_{\mathcal{P}} G(u(\boldsymbol{x};\boldsymbol{\mu});\boldsymbol{x},\boldsymbol{\mu})\rho(\boldsymbol{\mu})\ d\boldsymbol{\mu}, \tag{2.38}$$

where $\rho(\boldsymbol{\mu})$ denotes the probability density function (PDF) of the input parameters. Most prominent examples of desired statistical information are to compute the expectation (so that $G = F$) or the variance (in which $G = (F - \mathbb{E}[F])^2$). Generally, the integrals above cannot be evaluated exactly and must therefore be approximated. A common method is to use Monte Carlo simulation. The first step is to construct a set of random parameter locations $\{\boldsymbol{\mu}^q\}_{q=1}^{Q}$ whose are independent and identically distributed (i.i.d.). Second, using the

random parameter locations, we generate a set of approximated solutions $\{\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q)\}_{q=1}^Q$ such that

$$u(\boldsymbol{x};\boldsymbol{\mu}^q) \approx \boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q), \qquad q = 1, 2, ..., Q. \tag{2.39}$$

These approximations can be achieved by performing numerical schemes to solve PDEs such as finite element method explained in Section 2.1. Third, we compute the corresponding approximated statistical information:

$$G(u(\boldsymbol{x};\boldsymbol{\mu}^q);\boldsymbol{x},\boldsymbol{\mu}^q) \approx G(\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q);\boldsymbol{x},\boldsymbol{\mu}^q), \qquad q = 1, 2, ..., Q. \tag{2.40}$$

This gives the approximation

$$\text{QoI} \approx \text{QoI}_{MC} = \frac{1}{Q}\sum_{q=1}^{Q} G(\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q);\boldsymbol{x},\boldsymbol{\mu}^q). \tag{2.41}$$

The Monte Carlo simulation for PDEs is summarized in the algorithm below.

**Algorithm 2.7** Monte Carlo simulation for PDEs

1: **Inputs**: PDE, $F$, $G$
**Output** : $\text{QoI}_{MC}$
2: Generate a set of i.i.d. random parameter locations $\{\boldsymbol{\mu}^q\}_{q=1}^Q$
3: Compute the corresponding set of approximate solutions $\{\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q)\}_{q=1}^Q$ and in turn the set of approximate integrands $\{G(\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q);\boldsymbol{x},\boldsymbol{\mu}^q)\}_{q=1}^Q$
4: Compute the quantity of interest:

$$\text{QoI}(\boldsymbol{x},t) \approx \text{QoI}_{MC} = \frac{1}{Q}\sum_{q=1}^{Q} G(\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q);\boldsymbol{x},\boldsymbol{\mu}^q)$$

The ubiquitous use of Monte Carlo simulation in practice is mainly motivated by the simplicity of the algorithm. A bottleneck however is the slow convergence behaviour. As we have seen before, the computations to solve a PDE numerous times can be an expensive task and we might need a lot of sample points. For that matter, reduced-order modeling (see Section 2.4) can be a great application in the field of uncertainty quantification. In that case, the set full-order solutions $\{\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^q)\}_{q=1}^Q$ can be replaced by a set of reduced-order solutions $\{\boldsymbol{u}_h^{rb}(\boldsymbol{x};\boldsymbol{\mu}^q)\}_{q=1}^Q$ which can be computed much faster.

### 2.7.1 Importance sampling

In light of failure analysis, rare event simulations are an important tool to get better insight in the behaviour of the corresponding model when failures are drastic. Suppose a failure occurs when

$$F(u(x,y;\boldsymbol{\mu});\boldsymbol{\mu}) > F_0,$$

with $F_0$ given. This means the QoI is now the failure probability given by (2.38) with

$$G(u(\boldsymbol{x};\boldsymbol{\mu});\boldsymbol{x},\boldsymbol{\mu}) = \begin{cases} 1 & \text{if} \quad F(u(x,y;\boldsymbol{\mu});\boldsymbol{\mu}) > F_0, \\ 0 & \text{else}, \end{cases} \tag{2.42}$$

This means we assign a 1 (failure) for when the OoI is larger than a threshold $F_0$ and 0 (success) else. Failure probabilities are often very small. If we were to approximate the failure probability using regular Monte Carlo simulation, the failure region is often not reached for random sampling. For that reason we use a method called importance sampling. Importance sampling is a technique that pushes the simulations to favor the rare events of interest so that they will occur more frequently than they would otherwise. In other words, we will change the probability distributions of the input parameters so as to make the rare events happen more often. We consider a biasing distribution $q(\boldsymbol{\mu})$ that favours events in the failure region.

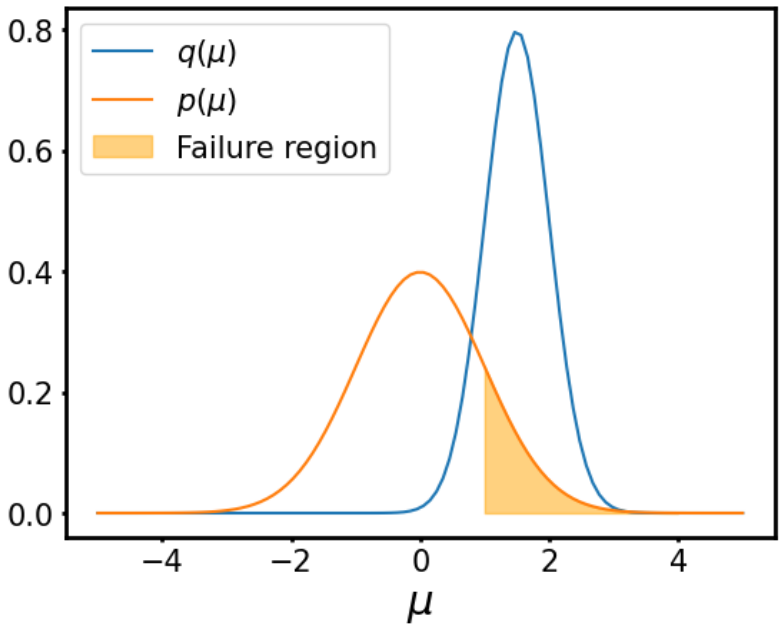



FIGURE 2.4: Importance sampling in one dimension: the biasing distribution $q(\mu)$ favours inputs $\mu$ that lie in the failure region.

Given the new distribution $q(\boldsymbol{\mu})$, we get that

$$\begin{aligned}
\text{QoI} &= \int_{\mathcal{P}} G(u(\boldsymbol{x};\boldsymbol{\mu});\boldsymbol{x},\boldsymbol{\mu})\rho(\boldsymbol{\mu})\ d\boldsymbol{\mu} \\
&= \int_{\mathcal{P}} \left( G(u(\boldsymbol{x};\boldsymbol{\mu});\boldsymbol{x},\boldsymbol{\mu}) \frac{\rho(\boldsymbol{\mu})}{q(\boldsymbol{\mu})} \right) q(\boldsymbol{\mu})\ d\boldsymbol{\mu} \\
&\approx \frac{1}{Q} \sum_{i=1}^{Q} G(\boldsymbol{u}_h(\boldsymbol{x};\boldsymbol{\mu}^i);\boldsymbol{x},\boldsymbol{\mu}^i) \frac{\rho(\boldsymbol{\mu}^i)}{q(\boldsymbol{\mu}^i)}.
\end{aligned}$$

Notice that we can now sample from the biasing distribution to favour the failures.

# 3 Reduced-order surrogate models for coupled problems

This section covers our main contribution: reduced-order surrogate models for coupled problems. Being proof of concept, the first two ROMs can only be applied to linear-linear coupled problems. As we will see later, the first two ROMs are useful in situations in which one part of the problem is paired with a high parametric complexity while the other part is rather simple. On the contrary, the other two ROMs can be used to solve both linear-linear and linear-nonlinear coupled problems. To explain the ROMs, we will use a linear and non-linear Poisson problem described in the next section.

## 3.1 Linear and non-linear Poisson problem

Consider the unit square domain $\Omega = (0,1)\times(0,1)$ with $\overline{\Omega} = \overline{\Omega_1 \cup \Omega_2}$ such that the domain is horizontally split into two equal parts, that is

$$\Omega_1 = \{(x,y) \mid x \in \Omega \ \wedge \ 0.5 < y < 1\}, \quad \Omega_2 = \{(x,y) \mid x \in \Omega \ \wedge \ 0 < y < 0.5\}.$$

The boundary is given by $\overline{\partial\Omega} = \overline{\Gamma_T \cup \Gamma_B \cup \Gamma_L \cup \Gamma_R}$, that is top, bottom, left and right boundary respectively. Let $\mathcal{N}_L$ be a linear operator resembling linear diffusion so that

$$\mathcal{N}_L u = -\nabla \cdot (\tilde{k}(\boldsymbol{x})\nabla u) \quad \text{with} \quad \tilde{k}(\boldsymbol{x}) = \sum_{i=1}^{2} \mathbb{1}_{\Omega_i}(\boldsymbol{x}). \tag{3.1}$$

The linear parametrized Poisson problem is given by

$$\begin{cases} -\mathcal{N}_L u = f(\boldsymbol{x};\boldsymbol{\mu}) & \text{in} \quad \Omega, \\ u = 1 & \text{on} \quad \Gamma_L \cup \Gamma_R, \\ \frac{\partial u}{\partial \boldsymbol{n}} = 1 & \text{on} \quad \Gamma_T, \\ \frac{\partial u}{\partial \boldsymbol{n}} = -1 & \text{on} \quad \Gamma_B, \end{cases} \tag{3.2}$$

with outward unit normal $\boldsymbol{n}$ and

$$f(\boldsymbol{x},\boldsymbol{\mu}) = \mathbb{1}_{\Omega_2}(\boldsymbol{x}) \ 100 \ \exp\left(0.5\left(\frac{(x-\mu_1)^2 + (y-\mu_2)^2}{\mu_3}\right)^2\right) \tag{3.3}$$

with $\boldsymbol{\mu} \in \mathcal{P} = [0,1] \times [0,\ 0.5] \times [0.001,\ 0.1]$. The source term describes a Gaussian-like function that varies in position and intensity on domain $\Omega_2$. The goal is to solve the problem over the whole parameter space.

The non-linear Poisson equation is similar to (3.2) but with non-linear diffusion operator

$$\tilde{\mathcal{N}}_D = -\nabla \cdot (\tilde{k}(\boldsymbol{x})\tilde{D}(\boldsymbol{x};\boldsymbol{\mu})\nabla u) \quad \text{with} \quad \tilde{D}(\boldsymbol{x};\boldsymbol{\mu}) = (1 + u\mathbb{1}_{\Omega_2}(\boldsymbol{x})), \tag{3.4}$$

so that the equation on domain $\Omega_2$ is non-linear. For these problems, the parametric complexity is solely present on domain $\Omega_2$. We anticipate that the solution within domain $\Omega_2$ will exhibit greater variation for distinct parameter placements when contrasted with the solution within domain $\Omega_1$. In context of reduced-order modeling, this means that solution manifold on domain $\Omega_1$ can be approximated by only a few basis functions. On the other hand, on domain $\Omega_2$ more basis functions are required to effectively approximate the solution manifold. This leads to the utilization of domain decomposition so that reduction

can be applied separately.

Using domain decomposition, the problem can be rewritten such that the DoFs are ordered in the solution vector per domain by following the procedure in Section 2.2. See Figure 3.1. The multidomain formulation in which $u_i = u\big|_{\Omega_i}$, $i = 1, 2$ is given by

$$\begin{cases} -\Delta u_1 = 0 & \text{in } \Omega_1, \\ u_1 = 1 & \text{on } \Gamma_{L1} \cup \Gamma_{R1}, \\ \frac{\partial u_1}{\partial \boldsymbol{n}} = 1 & \text{on } \Gamma_T, \\ -\Delta u_2 = f(\boldsymbol{x}, \boldsymbol{\mu}) & \text{in } \Omega_2, \\ u_2 = 1 & \text{on } \Gamma_{L2} \cup \Gamma_{R2}, \\ \frac{\partial u_2}{\partial \boldsymbol{n}} = -1 & \text{on } \Gamma_B, \\ u_1 = u_2 & \text{on } \Gamma, \\ \frac{\partial u_1}{\partial \boldsymbol{n}} = \frac{\partial u_2}{\partial \boldsymbol{n}} & \text{on } \Gamma. \end{cases} \tag{3.5}$$

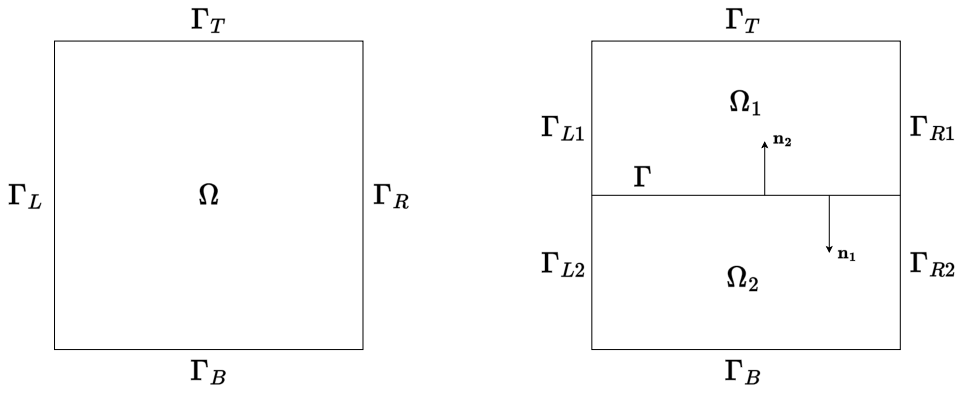



FIGURE 3.1: Original domain (left) and decomposed domain (right) for the Poisson problem in (3.2).

## 3.2 Surrogate models for linear-linear coupled problems

Consider the linear Poisson problem in (3.2). The ROMs will be explained in the offline and online setting.

**Offline stage**

By performing domain decomposition, we obtain the rearranged algebraic formulation as in (2.21) but now with dependency of the parameters, that is the linear system

$$\mathbb{A}\boldsymbol{U} = \boldsymbol{F}(\boldsymbol{\mu}), \tag{3.6}$$

with $\mathbb{A} \in \mathbb{R}^{N \times N}$ and $\boldsymbol{F}(\boldsymbol{\mu}) \in \mathbb{R}^N$. The offline stage begins by constructing a set of sample solutions. Specifically, we generate $n_s$ full-order solutions to (3.6) for random parameter locations $\boldsymbol{\mu}^i \in \mathcal{P}$ $(i = 1, 2, ..., n_s)$. The parameter locations are stored in a set $\Xi_{n_s} = \{\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}\}$. Moreover, we store all the matrices and vectors for each parameter location in databases

$$\mathcal{A}_{n_s} = \{\mathbb{A}_{11},\ \mathbb{A}_{1\Gamma},\ \mathbb{A}_{22},\ \mathbb{A}_{2\Gamma},\ \mathbb{A}_{\Gamma 1},\ \mathbb{A}_{\Gamma 2},\ \mathbb{A}^{(1)}_{\Gamma\Gamma},\ \mathbb{A}^{(2)}_{\Gamma\Gamma}\} \tag{3.7}$$

and

$$\mathcal{F}_{n_s} = \left\{ \boldsymbol{F}_1,\ \boldsymbol{F}_\Gamma^{(1)},\ \boldsymbol{F}_2(\boldsymbol{\mu}^i),\ \boldsymbol{F}_\Gamma^{(2)}(\boldsymbol{\mu}^i) \right\}_{i=1}^{n_s} \tag{3.8}$$

Note that for this particular problem, the dependence of $\boldsymbol{\mu}$ is only present in the set $\mathcal{F}_{n_s}$. For more general problems, we may need to save the matrices and vectors for every parameter location in $\mathcal{A}_{n_s}$ too. Every solution vector is composed of three vectors that correspond to the solutions on domain $\Omega_1$, $\Omega_2$ and the interface $\Gamma$. Therefore, each solution vector can be represented as

$$\boldsymbol{U}(\boldsymbol{\mu}^i) = \begin{bmatrix} \boldsymbol{U}_1(\boldsymbol{\mu}^i) \\ \boldsymbol{U}_2(\boldsymbol{\mu}^i) \\ \boldsymbol{U}_\Gamma(\boldsymbol{\mu}^i) \end{bmatrix} \in \mathbb{R}^N, \qquad i = 1, 2, ..., n_s. \tag{3.9}$$

The dimension of the solution vector is a sum of three integers corresponding to dimensions of the respective domains such that $N = N_1 + N_2 + N_\Gamma$. The full-order solutions are collected in the snapshot matrix $\mathbb{S}$, that is

$$\mathbb{S} = \left[ \boldsymbol{U}(\boldsymbol{\mu}^1) \mid \boldsymbol{U}(\boldsymbol{\mu}^2) \mid ... \mid \boldsymbol{U}(\boldsymbol{\mu}^{n_s}) \right] \in \mathbb{R}^{N \times n_s}. \tag{3.10}$$

The rearranged solution vector allows us to perform reduced basis methods on the decomposed domains separately. On domain $\Omega_1$ and the interface $\Gamma$ we perform a POD-ROM as in Section 2.4.1 to obtain the transformation matrices $\mathbb{V}_1 \in \mathbb{R}^{N \times n_1}$ and $\mathbb{V}_\Gamma \in \mathbb{R}^{N \times n_\Gamma}$ with $n_1 \ll N_1$ and $n_\Gamma \ll N_\Gamma$, so that

$$\boldsymbol{U}_1(\boldsymbol{\mu}) \approx \mathbb{V}_1 \boldsymbol{u}_1(\boldsymbol{\mu}) \quad \text{and} \quad \boldsymbol{U}_\Gamma(\boldsymbol{\mu}) \approx \mathbb{V}_\Gamma \boldsymbol{u}_\Gamma(\boldsymbol{\mu}). \tag{3.11}$$

Next, consider the transformation matrix $\hat{\mathbb{V}}^T$ given by

$$\hat{\mathbb{V}}^T = \begin{bmatrix} \mathbb{V}_1^T & 0 & 0 \\ 0 & \mathbb{I}_{N_2} & 0 \\ 0 & 0 & \mathbb{V}_\Gamma^T \end{bmatrix} \in \mathbb{R}^{(n_1 + N_2 + n_\Gamma) \times N_h}. \tag{3.12}$$

We left-multiply $\hat{\mathbb{V}}^T$ on both sides of (3.6) and substitute the approximations from (3.11). This gives the so-called hybrid model, that is

$$a\hat{\boldsymbol{u}} = \begin{bmatrix} a_{11} & 0 & a_{1\Gamma} \\ 0 & \mathbb{A}_{22} & a_{2\Gamma} \\ a_{\Gamma 1} & a_{\Gamma 2} & a_{\Gamma\Gamma}^{(1)} + a_{\Gamma\Gamma}^{(2)} \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{u}}_1 \\ \hat{\boldsymbol{U}}_2 \\ \hat{\boldsymbol{u}}_\Gamma \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{F}_2 \\ \boldsymbol{f}_\Gamma^{(1)} + \boldsymbol{f}_\Gamma^{(2)} \end{bmatrix} = \hat{\boldsymbol{f}}(\boldsymbol{\mu}), \tag{3.13}$$

with

$$\begin{aligned} &a_{11} = \mathbb{V}_1^T \mathbb{A}_{11} \mathbb{V}_1, \quad a_{1\Gamma} = \mathbb{V}_1^T \mathbb{A}_{11} \mathbb{V}_\Gamma, \quad a_{2\Gamma} = \mathbb{A}_{2\Gamma} \mathbb{V}_\Gamma, \quad a_{\Gamma 1} = \mathbb{V}_\Gamma^T \mathbb{A}_{\Gamma 1} \mathbb{V}_1, \\ &a_{\Gamma 2} = \mathbb{V}_\Gamma^T \mathbb{A}_{\Gamma 2}, \quad a_{\Gamma\Gamma}^{(1)} = \mathbb{V}_\Gamma^T \mathbb{A}_{\Gamma\Gamma}^{(1)} \mathbb{V}_\Gamma, \quad a_{\Gamma\Gamma}^{(2)} = \mathbb{V}_\Gamma^T \mathbb{A}_{\Gamma\Gamma}^{(2)} \mathbb{V}_\Gamma, \quad \boldsymbol{f}_1 = \mathbb{V}_1^T \boldsymbol{F}_1, \\ &\boldsymbol{f}_\Gamma^{(1)} = \mathbb{V}_\Gamma^T \boldsymbol{F}_\Gamma^{(1)}, \quad \boldsymbol{f}_\Gamma^{(2)} = \mathbb{V}_\Gamma^T \boldsymbol{F}_\Gamma^{(2)}. \end{aligned}$$

Note that this is similar to the first step of the online stage in the POD-ROM procedure but now only applied to domain $\Omega_1$ and the interface. To ease notation, we omit the dependence of the parameter $\boldsymbol{\mu}$ but it is important to remember that the matrices and vectors may still depend on $\boldsymbol{\mu}$. With the reduction applied, the hybrid model is a system of $(n_1 + N_2 + n_\Gamma)$ equations. Using the hybrid model we generate another snapshot matrix

$\hat{\mathbb{S}} \in \mathbb{R}^{(n_1+N_2+n_\Gamma)\times n_s}$ containing the solutions to the hybrid model with the same parameter locations as in $\Xi$. In addition, we generate a matrix and vector database

$$\tilde{\mathcal{T}}_{n_s} = \{a_{11}^{(1)},\ a_{\Gamma 1},\ a_{11},\ a_{1\Gamma},\ \boldsymbol{f}_1,\ \boldsymbol{f}_\Gamma^{(1)}\} \tag{3.14}$$

which will be needed later. Note again that for more general problems, we may need to compute these matrices and vectors for every parameter location.

Now that we have applied reduction on $\Omega_1$ and the interface, we will apply reduction on domain $\Omega_2$ too. The reason for not applying this reduction at the same time as on domain $\Omega_1$ and the interface is due to the possible error increase. With the hybrid model, we aim to minimize the error on domain $\Omega_2$. This is especially effective in this particular problem in which the parametric complexity is only present on domain $\Omega_2$. Similar to the POD-NN procedure in Algorithm 2.2, we apply a POD-NN using the hybrid model to obtain $\mathbb{V}_2$ so that $\hat{\boldsymbol{u}}_2 \approx \mathbb{V}^T\hat{\boldsymbol{U}}_2$. This time, we train separate neural networks for each mode:

$$\hat{\pi}_2^{(i)} \ :\ \{\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2\} \longrightarrow \hat{\boldsymbol{u}}_2^{(i)}, \qquad i = 1, 2, ..., n_2. \tag{3.15}$$

Note that this mapping inputs both the parameters $\boldsymbol{\mu}_2$ on domain $\Omega_2$ and the interface values $\hat{\boldsymbol{u}}_\Gamma$ of the hybrid model. The reason for training separate neural networks for each mode is as follows. In a regular POD-NN, one maps to all modes at once in a single neural network. In that case, the loss is the error of all modes together. Recall that the first modes correspond to the highest singular values in the singular value decomposition (see Section 2.3). By the well-known results from Schmidt-Eckart-Young (Definition 2.2), we know that the first singular values contribute to the most information of the low-rank approximation. By training separate neural networks, each mode is optimized separately, instead of optimized together.

Similar to the traction force derived in the rearranged system of equations in (2.21), we obtain a reduced-order traction force deducted from the hybrid model, that is

$$\boldsymbol{t}_\Gamma = (a_{\Gamma\Gamma}^{(1)} - a_{\Gamma 1}a_{11}^{-1}a_{1\Gamma})\hat{\boldsymbol{u}}_\Gamma + a_{\Gamma 1}a_{11}^{-1}\boldsymbol{f}_1 - \boldsymbol{f}_\Gamma^{(1)}. \tag{3.16}$$

Note that this is similar to (2.22) and relates the interaction between domain $\Omega_1$ and $\Omega_2$. For all snapshots, we store the traction forces in the database $\{\boldsymbol{t}_\Gamma(\boldsymbol{\mu}^i)\}_{i=1}^{n_s}$. The final step of the offline stage is to construct the mapping for the traction force by training a neural network so that

$$\hat{\boldsymbol{t}}_\Gamma \ :\ \{\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2\} \longrightarrow \boldsymbol{t}_\Gamma. \tag{3.17}$$

**Online stage**

For an unseen parameter location, we construct the algebraic system as in (3.6) and multiply the corresponding parts of the system by the transformation matrices $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$ to obtain the hybrid model. Without solving it, the next step is to solve for the modes on the interface $\boldsymbol{u}_\Gamma$. From the algebraic system we obtain the traction force for the unseen parameter location in terms of the unknown $\hat{\boldsymbol{u}}_\Gamma$ similar to (3.16). By equating this to the mapping of the traction force $\hat{\boldsymbol{t}}_\Gamma$ we obtain a non-linear function (due to $\hat{\boldsymbol{t}}_\Gamma$) given by

$$\boldsymbol{F}(\hat{\boldsymbol{u}}_\Gamma) := (a_{\Gamma\Gamma}^{(1)} - a_{\Gamma 1}a_{11}^{-1}a_{1\Gamma})\hat{\boldsymbol{u}}_\Gamma + a_{\Gamma 1}a_{11}^{-1}\boldsymbol{f}_1 - \boldsymbol{f}_\Gamma^{(1)} - \hat{\boldsymbol{t}}_\Gamma(\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2) = \boldsymbol{0}. \tag{3.18}$$

There are various ways to solve (3.18) that are explained in more detail in Section 3.4. Given the reduced-order solution $\boldsymbol{u}_\Gamma^{rb}$, we advance by computing the reduced-order solution $\boldsymbol{u}_1^{rb}$ using the first line in (3.13) such that

$$\hat{\boldsymbol{u}}_1 = a_{11}^{-1}(\boldsymbol{f}_1 - a_{1\Gamma}\hat{\boldsymbol{u}}_\Gamma). \tag{3.19}$$

The reduced-order solution $\boldsymbol{u}_2^{rb}$ is determined through the neural networks $\hat{\pi}_2^{(i)}$, $i = 1, 2, ..., n_2$. Using the transformation matrices to the corresponding domain, we project the reduced-order solutions to the full-order space to recover the full-order solution per domain so that

$$\boldsymbol{U}_1^{rb} = \mathbb{V}_1\boldsymbol{u}_1^{rb}, \quad \boldsymbol{U}_2^{rb} = \mathbb{V}_2\boldsymbol{u}_2^{rb}, \quad \boldsymbol{U}_\Gamma^{rb} = \mathbb{V}_\Gamma\boldsymbol{u}_\Gamma^{rb}.$$

This ROM is a method to solve linear-linear coupled problems such that the reduced-interface vector is determined through a combination of data-driven and intrusive modeling. For that reason, we refer to this ROM as the hybrid-LL. The general algorithm is summarized below.

**Algorithm 3.1** Hybrid-LL

**Offline Stage**

Generate parameter set $\Xi_{n_s} = \{\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}\}$;

Generate full-order solution database $\{\boldsymbol{U}(\boldsymbol{\mu}^1),\ \boldsymbol{U}(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{U}(\boldsymbol{\mu}^{n_s})\}$ of $n_s$ solutions and store

$$\mathcal{A}_{n_s} = \left\{\mathbb{A}_{11},\ \mathbb{A}_{1\Gamma},\ \mathbb{A}_{22},\ \mathbb{A}_{2\Gamma},\ \mathbb{A}_{\Gamma 1},\ \mathbb{A}_{\Gamma 2},\ \mathbb{A}_{\Gamma\Gamma}^{(1)},\ \mathbb{A}_{\Gamma\Gamma}^{(2)}\right\}_{i=1}^{n_s}$$

$$\mathcal{F}_{n_s} = \left\{\boldsymbol{F}_1,\ \boldsymbol{F}_\Gamma^{(1)},\ \boldsymbol{F}_2,\ \boldsymbol{F}_\Gamma^{(2)}\right\}_{i=1}^{n_s}$$

possibly dependent on $\boldsymbol{\mu}^i$;

Apply POD-ROM on domain $\Omega_1$ and $\Gamma$ to obtain $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$;

Generate solution set $\{\hat{\boldsymbol{u}}(\boldsymbol{\mu}^1),\ \hat{\boldsymbol{u}}(\boldsymbol{\mu}^2),\ ...,\ \hat{\boldsymbol{u}}(\boldsymbol{\mu}^{n_s})\}$ containing $n_s$ solutions to the hybrid model and store

$$\tilde{\mathcal{T}}_{n_s} = \left\{a_{11}^{(1)},\ a_{\Gamma 1},\ a_{11},\ a_{1\Gamma},\ \boldsymbol{f}_1,\ \boldsymbol{f}_\Gamma^{(1)}\right\}_{i=1}^{n_s}$$

possibly dependent on $\boldsymbol{\mu}^i$;

Apply POD-NN on domain $\Omega_2$ to obtain $\mathbb{V}_2$ and train separate neural networks

$$\pi_2^{(i)} \ : \ \{\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2\} \longrightarrow \hat{\boldsymbol{u}}_2^{(i)}, \qquad i = 1, 2, ..., n_2;$$

Generate the training set of traction forces $\{\boldsymbol{t}_\Gamma(\boldsymbol{\mu}^i)\}_{i=1}^{n_s}$;

Use a neural network to train $\hat{\boldsymbol{t}}_\Gamma \ : \ \{\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2\} \longrightarrow \boldsymbol{t}_\Gamma$;

**Online Stage**

Construct the hybrid model;

Obtain $\boldsymbol{u}_\Gamma^{rb}$ using $\boldsymbol{F}(\hat{\boldsymbol{u}}_\Gamma) = \boldsymbol{0}$;

Obtain $\boldsymbol{u}_2^{rb}$ using $\hat{\pi}_2^{(i)}(\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2),\ i = 1, 2, ..., n_s$;

Obtain $\boldsymbol{u}_1^{rb}$ using $\hat{\boldsymbol{u}}_1 = a_{11}^{-1}\left(\boldsymbol{f}_1 - a_{1\Gamma}\hat{\boldsymbol{u}}_\Gamma\right)$;

Transform the reduced-order solutions to the full-order space:

$$\boldsymbol{U}_1^{rb} = \mathbb{V}_1\boldsymbol{u}_1^{rb}, \quad \boldsymbol{U}_2^{rb} = \mathbb{V}_2\boldsymbol{u}_2^{rb}, \quad \boldsymbol{U}_\Gamma^{rb} = \mathbb{V}_\Gamma\boldsymbol{u}_\Gamma^{rb}.$$

In the hybrid-LL, the neural network for the traction force is ultimately used to solve the non-linear equation in (3.18) to determine the modes on the interface $\Gamma$. Rather than this approach, we can also directly train a neural network for the modes on the interface. The matrices and vectors to compute the traction forces no longer need to be stored. This way, the reduced interface vector is determined completely through data-driven modeling. For that reason we refer to this ROM as surrogate-LL. The neural network is given by

$$\hat{\boldsymbol{\pi}}_\Gamma : \{\boldsymbol{\mu}\} \longrightarrow \hat{\boldsymbol{u}}_\Gamma, \tag{3.20}$$

so it is a mapping from the parameter vector to the modes on the interface. In the online stage, one can directly use this mapping to determine the modes on the interface instead of solving the non-linear equation in (3.18). The procedure of the surrogate-LL is given below in Algorithm 3.2.

**Algorithm 3.2** Surrogate-LL

**Offline Stage**

Generate parameter set $\Xi_{n_s} = \{\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}\}$;

Generate full-order solution database $\{\boldsymbol{U}(\boldsymbol{\mu}^1),\ \boldsymbol{U}(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{U}(\boldsymbol{\mu}^{n_s})\}$ of $n_s$ solutions and store

$$\mathcal{A}_{n_s} = \left\{\mathbb{A}_{11},\ \mathbb{A}_{1\Gamma},\ \mathbb{A}_{22},\ \mathbb{A}_{2\Gamma},\ \mathbb{A}_{\Gamma 1},\ \mathbb{A}_{\Gamma 2},\ \mathbb{A}^{(1)}_{\Gamma\Gamma},\ \mathbb{A}^{(2)}_{\Gamma\Gamma}\right\}_{i=1}^{n_s}$$

$$\mathcal{F}_{n_s} = \left\{\boldsymbol{F}_1,\ \boldsymbol{F}^{(1)}_\Gamma,\ \boldsymbol{F}_2,\ \boldsymbol{F}^{(2)}_\Gamma\right\}_{i=1}^{n_s}$$

possibly dependent on $\boldsymbol{\mu}^i$ for $n_s$ parameter locations;

Apply POD-ROM on domain $\Omega_1$ and $\Gamma$ to obtain $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$;

Create a database $\{\hat{\boldsymbol{u}}(\boldsymbol{\mu}^1),\ \hat{\boldsymbol{u}}(\boldsymbol{\mu}^2),\ ...,\ \hat{\boldsymbol{u}}(\boldsymbol{\mu}^{n_s})\}$ containing $n_s$ solutions to the hybrid model;

Apply POD-NN on domain $\Omega_2$ to obtain $\mathbb{V}_2$ and train separate neural networks

$$\pi_2^{(i)}\ :\ \{\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2\} \longrightarrow \hat{\boldsymbol{u}}_2^{(i)}, \qquad i = 1, 2, ..., n_2$$

Use a neural network to train $\hat{\boldsymbol{\pi}}_\Gamma : \{\boldsymbol{\mu}\} \longrightarrow \hat{\boldsymbol{u}}_\Gamma$;

**Online Stage**

Construct the hybrid model;

Obtain $\boldsymbol{u}^{rb}_\Gamma$ using $\hat{\boldsymbol{\pi}}_\Gamma(\boldsymbol{\mu})$;

Obtain $\boldsymbol{u}^{rb}_2$ using $\hat{\pi}_2^{(i)}(\hat{\boldsymbol{u}}_\Gamma, \boldsymbol{\mu}_2),\ i = 1, 2, ..., n_s$;

Obtain $\boldsymbol{u}^{rb}_1$ using $\hat{\boldsymbol{u}}_1 = a_{11}^{-1}\left(\boldsymbol{f}_1 - a_{1\Gamma}\hat{\boldsymbol{u}}_\Gamma\right)$;

Transform the reduced-order solutions to the full-order space:

$$\boldsymbol{U}^{rb}_1 = \mathbb{V}_1\boldsymbol{u}^{rb}_1, \quad \boldsymbol{U}^{rb}_2 = \mathbb{V}_2\boldsymbol{u}^{rb}_2, \quad \boldsymbol{U}^{rb}_\Gamma = \mathbb{V}_\Gamma\boldsymbol{u}^{rb}_\Gamma$$

## 3.3 Surrogate models for linear-nonlinear coupled problems

So far we have discussed two ROMs to solve linear-linear coupled problems. Now suppose that on the decomposed domain $\Omega_2$, we define a non-linear Poisson equation so that the problem becomes a linear-nonlinear coupled problem. We consider the Poisson problem in (3.2) but with (3.4).

In the full-order method, this forces the problem to be solved as a full non-linear problem even though part of the problem is linear. As a result, it is no longer possible to construct a linear system of equations to be solved such as in (2.21). The finite element problem can be represented as

$$\mathbb{A}\boldsymbol{U}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbb{A}_{11} & 0 & \mathbb{A}_{1\Gamma} \\ 0 & \mathbb{N}_{22} & \mathbb{N}_{2\Gamma} \\ \mathbb{A}_{\Gamma 1} & \mathbb{N}_{\Gamma 2} & \mathbb{A}^{(1)}_{\Gamma\Gamma} + \mathbb{N}^{(2)}_{\Gamma\Gamma} \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_1 \\ \boldsymbol{U}_2 \\ \boldsymbol{U}_\Gamma \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_1 \\ \boldsymbol{F}_2 \\ \boldsymbol{F}^{(1)}_\Gamma + \boldsymbol{F}^{(2)}_\Gamma \end{bmatrix} = F(\boldsymbol{\mu}),$$

for which $\mathbb{A}_{**}$ are linear operators, while $\mathbb{N}_{**}$ are non-linear. In the offline stages of the hybrid-LL and surrogate-LL, we multiply the linear system of equations by a transformation matrix to obtain the hybrid model in (3.13). This is now no longer possible since the non-linear operators are unknown.

Even for linear-linear coupled problems, the hybrid-LL and surrogate-LL may still require a substantial amount of computational and memory cost. This is because during the offline stage, next to the generation of a full-order solution database, we generate another solution database for the hybrid solutions. For every solution, this requires solving a system of equations for $n_1 + N_2 + n_\Gamma$ which can still be cost-inefficient. Rather than constructing this database, one can approximate the reduced-order solutions by applying the transformation matrices on the full-order solutions directly. This also allows us to solve linear-nonlinear coupled problems. The idea is worked out in the ROMs hybrid-NL and surrogate-NL that follow the procedure of the hybrid-LL and surrogate-LL respectively. We will now explain the surrogate-NL in further detail.

**Offline stage**
We construct a full-order solution database $\{\boldsymbol{U}(\boldsymbol{\mu}^1),\ \boldsymbol{U}(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{U}(\boldsymbol{\mu}^{n_s})\}$ such that each solution is again composed of three vectors corresponding to each domain. At the same time, on the linear domain $\Omega_1$ we construct the system of linear equations - without solving it - with no boundary conditions specified on the interface $\Gamma$. The matrices and vectors in

$$\mathcal{T}_{n_s} = \left\{ \mathbb{A}^{(1)}_{11},\ \mathbb{A}_{\Gamma 1},\ \mathbb{A}_{11},\ \mathbb{A}_{1\Gamma},\ \boldsymbol{F}_1,\ \boldsymbol{F}^{(1)}_\Gamma \right\}_{i=1}^{n_s} \tag{3.21}$$

possibly depending on $\boldsymbol{\mu}^i$ are stored and later needed in the computation of the traction force as in (3.16).

Using the full-order solution database, we apply POD-ROM on domains $\Omega_1$ and $\Gamma$ to obtain the transformation matrices $\mathbb{V}_1 \in \mathbb{R}^{N\times n_1}$ and $\mathbb{V}_\Gamma \in \mathbb{R}^{N\times n_\Gamma}$ with $n_1 \ll N_1$ and $n_\Gamma \ll N_\Gamma$. These are directly applied on the corresponding full-order solutions so that

$$\boldsymbol{u}_1(\boldsymbol{\mu}^i) = \mathbb{V}_1^T \boldsymbol{U}_1(\boldsymbol{\mu}^i) \quad \text{and} \quad \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^i) = \mathbb{V}_\Gamma^T \boldsymbol{U}_\Gamma(\boldsymbol{\mu}^i), \quad i = 1, 2, ..., n_s.$$

The reduced-order solutions are collected in the databases $\{\boldsymbol{u}_1(\boldsymbol{\mu}^1),\ \boldsymbol{u}_1(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_1(\boldsymbol{\mu}^{n_s})\}$ and $\{\boldsymbol{u}_\Gamma(\boldsymbol{\mu}^1),\ \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^{n_s})\}$. In this way, the reduced-order solutions are retrieved much faster than solving the hybrid model every time in the hybrid and surrogate-LL, albeit with some loss of accuracy.

Similar to the hybrid and surrogate-LL, we apply a POD-NN to obtain $\mathbb{V}_2 \in \mathbb{R}^{N\times n_2}$ with $n_2 \ll N_2$ and construct and store the reduced-order solutions in a database $\{\boldsymbol{u}_2(\boldsymbol{\mu}^1),\ \boldsymbol{u}_2(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_2(\boldsymbol{\mu}^{n_s})\}$. We train separate neural networks for the modes on

domain $\Omega_2$ but with inputs and outputs from the reduced-order solution database, that is

$$\hat{\pi}_2^{(i)} \ : \ \{\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}\} \longrightarrow \boldsymbol{u}_2^{(i)}, \qquad i = 1, 2, ..., n_2, \tag{3.22}$$

where $\boldsymbol{\mu}_{nlin}$ corresponds to the parameters on the non-linear domain $\Omega_2$.

In order to compute the reduced-order traction forces, we first determine the reduced-order matrices and vectors such that

$$\begin{aligned} &a_{11} = \mathbb{V}_1^T \mathbb{A}_{11} \mathbb{V}_1, \quad a_{1\Gamma} = \mathbb{V}_1^T \mathbb{A}_{11} \mathbb{V}_\Gamma, \quad a_{\Gamma 1} = \mathbb{V}_\Gamma^T \mathbb{A}_{\Gamma 1} \mathbb{V}_1, \\ &a_{\Gamma\Gamma}^{(1)} = \mathbb{V}_\Gamma^T \mathbb{A}_{\Gamma\Gamma}^{(1)} \mathbb{V}_\Gamma, \quad \boldsymbol{f}_1 = \mathbb{V}_1^T \boldsymbol{F}_1, \quad \boldsymbol{f}_\Gamma^{(1)} = \mathbb{V}_\Gamma^T \boldsymbol{F}_\Gamma^{(1)}, \end{aligned}$$

for every parameter location. The reduced-order matrices and vectors are stored in a database $\widetilde{\mathcal{T}}_{n_s}$ as in (3.14) possibly depending on $\boldsymbol{\mu}^i$. The above collection of matrices and vectors together with the transformation matrices $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$ and reduced-order solutions on the interface allow us to determine the collection of reduced traction forces such as in (3.16) to obtain $\{\boldsymbol{t}_\Gamma(\boldsymbol{\mu}^i)\}_{i=1}^{n_s}$. In this way, we train a neural network for the reduced-order traction force, that is

$$\hat{\boldsymbol{t}}_\Gamma \ : \ \{\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}\} \longrightarrow \boldsymbol{t}_\Gamma. \tag{3.23}$$

**Online Stage**
For an unseen parameter location, we construct the system of linear equations on domain $\Omega_1$ to extract the matrices as in (3.21). Using the transformation matrices $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$ we compute the reduced-order traction force in terms of $\boldsymbol{u}_\Gamma$. The rest of the procedure is similar to the online stage of the hybrid-LL in Algorithm 3.1. The hybrid-NL algorithm is given below.

**Algorithm 3.3** Hybrid-NL

**Offline Stage**

Generate parameter set $\Xi_{n_s} = \{\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}\}$;

Generate full-order solution database $\{\boldsymbol{U}(\boldsymbol{\mu}^1),\ \boldsymbol{U}(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{U}(\boldsymbol{\mu}^{n_s})\}$ of $n_s$ solutions and store

$$\mathcal{T}_{n_s} = \left\{\mathbb{A}_{11}^{(1)},\ \mathbb{A}_{\Gamma 1},\ \mathbb{A}_{11},\ \mathbb{A}_{1\Gamma},\ \boldsymbol{F}_1,\ \boldsymbol{F}_\Gamma^{(1)}\right\}_{i=1}^{n_s}$$

possibly dependent on $\boldsymbol{\mu}^i$ for $n_s$ parameter locations;

Apply POD-ROM on domains $\Omega_1$ and $\Gamma$ to obtain $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$ so that

$$\boldsymbol{U}_1(\boldsymbol{\mu}) \approx \mathbb{V}_1 \boldsymbol{u}_1(\boldsymbol{\mu}) \quad \text{and} \quad \boldsymbol{U}_\Gamma(\boldsymbol{\mu}) \approx \mathbb{V}_\Gamma \boldsymbol{u}_\Gamma(\boldsymbol{\mu})$$

to create the reduced-order solution database $\{\boldsymbol{u}_1(\boldsymbol{\mu}^1),\ \boldsymbol{u}_1(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_1(\boldsymbol{\mu}^{n_s})\}$ and $\{\boldsymbol{u}_\Gamma(\boldsymbol{\mu}^1),\ \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^{n_s})\}$;

Apply POD-NN on domain $\Omega_2$ to obtain $\mathbb{V}_2$ and construct reduced-order solution database $\{\boldsymbol{u}_1(\boldsymbol{\mu}^1),\ \boldsymbol{u}_1(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_1(\boldsymbol{\mu}^{n_s})\}$ and train separate neural networks

$$\hat{\pi}_2^{(i)}\ :\ \{\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}\} \longrightarrow \boldsymbol{u}_2^{(i)}, \qquad i = 1, 2, ..., n_2;$$

Generate reduced-order database

$$\tilde{\mathcal{T}}_{n_s} = \left\{a_{11}^{(1)},\ a_{\Gamma 1},\ a_{11},\ a_{1\Gamma},\ \boldsymbol{f}_1,\ \boldsymbol{f}_\Gamma^{(1)}\right\}_{i=1}^{n_s},$$

possibly depending on $\boldsymbol{\mu}^i$;

Generate the training set for the traction force with the reduced-order traction forces $\{\boldsymbol{t}_\Gamma(\boldsymbol{\mu}^i)\}_{i=1}^{n_s}$;

Train a neural network for the traction force

$$\hat{\boldsymbol{t}}_\Gamma\ :\ \{\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}\} \longrightarrow \boldsymbol{t}_\Gamma;$$

**Online Stage**

Construct the reduced-order system of linear equations on domain $\Omega_1$ using $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$;

Obtain $\boldsymbol{u}_\Gamma^{rb}$ using $\boldsymbol{F}(\boldsymbol{u}_\Gamma) = \boldsymbol{0}$;

Obtain $\boldsymbol{u}_2^{rb}$ using $\hat{\pi}_2^{(i)}(\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}),\ i = 1, 2, ..., n_s$;

Obtain $\boldsymbol{u}_1^{rb}$ using $\hat{\boldsymbol{u}}_1 = a_{11}^{-1}\left(\boldsymbol{f}_1 - a_{1\Gamma}\boldsymbol{u}_\Gamma\right)$;

Transform the reduced-order solutions to the full-order space:

$$\boldsymbol{U}_1^{rb} = \mathbb{V}_1 \boldsymbol{u}_1^{rb}, \quad \boldsymbol{U}_2^{rb} = \mathbb{V}_2 \boldsymbol{u}_2^{rb}, \quad \boldsymbol{U}_\Gamma^{rb} = \mathbb{V}_\Gamma \boldsymbol{u}_\Gamma^{rb}.$$

The last method to discuss is the surrogate-NL. Here, we follow the procedure of the surrogate-LL but construct the reduced-order solution directly as explained earlier. See Algorithm 3.4.

**Algorithm 3.4** Surrogate-NL

**Offline Stage**

Generate parameter set $\Xi_{n_s} = \{\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^{n_s}\}$;

Generate full-order solution database $\{\boldsymbol{U}(\boldsymbol{\mu}^1),\ \boldsymbol{U}(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{U}(\boldsymbol{\mu}^{n_s})\}$ of $n_s$ solutions

Apply POD-ROM on domains $\Omega_1$ and $\Gamma$ to obtain $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$ so that

$$\boldsymbol{U}_1(\boldsymbol{\mu}) \approx \mathbb{V}_1 \boldsymbol{u}_1(\boldsymbol{\mu}) \quad \text{and} \quad \boldsymbol{U}_\Gamma(\boldsymbol{\mu}) \approx \mathbb{V}_\Gamma \boldsymbol{u}_\Gamma(\boldsymbol{\mu})$$

to create the reduced-order solution database $\{\boldsymbol{u}_1(\boldsymbol{\mu}^1),\ \boldsymbol{u}_1(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_1(\boldsymbol{\mu}^{n_s})\}$ and $\{\boldsymbol{u}_\Gamma(\boldsymbol{\mu}^1),\ \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^{n_s})\}$;

Apply POD-NN on domain $\Omega_2$ to obtain $\mathbb{V}_2$ and construct reduced-order solution database $\{\boldsymbol{u}_1(\boldsymbol{\mu}^1),\ \boldsymbol{u}_1(\boldsymbol{\mu}^2),\ ...,\ \boldsymbol{u}_1(\boldsymbol{\mu}^{n_s})\}$ and train separate neural networks

$$\hat{\pi}_2^{(i)} \ : \ \{\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}\}; \longrightarrow \boldsymbol{u}_2^{(i)}, \qquad i = 1, 2, ..., n_2;$$

Use a neural network to train $\hat{\boldsymbol{\pi}}_\Gamma : \boldsymbol{\mu} \longrightarrow \boldsymbol{u}_\Gamma$;

**Online Stage**

Obtain $\boldsymbol{u}_\Gamma^{rb}$ using $\hat{\boldsymbol{\pi}}_\Gamma(\boldsymbol{\mu})$;

Obtain $\boldsymbol{u}_2^{rb}$ using $\hat{\pi}_2^{(i)}(\boldsymbol{u}_\Gamma, \boldsymbol{\mu}_{nlin}),\ i = 1, 2, ..., n_s$;

Construct the reduced-order system of linear equations on domain $\Omega_1$ using $\mathbb{V}_1$ and $\mathbb{V}_\Gamma$;

Obtain $\boldsymbol{u}_1^{rb}$ using $\boldsymbol{u}_1 = a_{11}^{-1}\left(\boldsymbol{f}_1 - a_{1\Gamma}\boldsymbol{u}_\Gamma\right)$;

Transform the reduced-order solutions to the full-order space:

$$\boldsymbol{U}_1^{rb} = \mathbb{V}_1 \boldsymbol{u}_1^{rb}, \quad \boldsymbol{U}_2^{rb} = \mathbb{V}_2 \boldsymbol{u}_2^{rb}, \quad \boldsymbol{U}_\Gamma^{rb} = \mathbb{V}_\Gamma \boldsymbol{u}_\Gamma^{rb}$$

## 3.4 Solving techniques for a system of non-linear equations

For the hybrid-LL and hybrid-NL, it remains to solve the non-linear equations in (3.18). In the next subsections, we will discuss a few methods.

### 3.4.1 Newton's method

One way is to view (3.18) as a root-finding problem that can be solved using Newton's method. Since the Jacobian cannot be determined exactly we have to approximate it. The Jacobian with respect to $\boldsymbol{u}_\Gamma$ is given by

$$\mathbb{J}_F(\boldsymbol{u}_\Gamma) = \begin{bmatrix} \frac{\partial F_1}{\partial u_{\Gamma,1}} & \cdots & \frac{\partial F_1}{\partial u_{\Gamma,n_\Gamma}} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_{n_\Gamma}}{\partial u_{\Gamma,1}} & \cdots & \frac{\partial F_{n_\Gamma}}{\partial u_{\Gamma,n_\Gamma}} \end{bmatrix}. \tag{3.24}$$

The entries of the Jacobian are approximated using forward finite differences. Let $\mathbb{H} = \text{diag}(h) \in \mathbb{R}^{n_\Gamma \times n_\Gamma}$ with $h > 0$ but small. Column-wise, this gives

$$\mathbb{J}_F(\boldsymbol{u}_\Gamma) \approx \frac{1}{h}\Big[\boldsymbol{F}(\boldsymbol{u}_\Gamma + \mathbb{H}\boldsymbol{e}^{(1)}) - \boldsymbol{F}(\boldsymbol{u}_\Gamma) \mid \ \ldots \ \mid \boldsymbol{F}(\boldsymbol{u}_\Gamma + \mathbb{H}\boldsymbol{e}^{(n_\Gamma)}) - \boldsymbol{F}(\boldsymbol{u}_\Gamma)\Big], \tag{3.25}$$

where $\boldsymbol{e}^{(i)} \in \mathbb{R}^{n_\Gamma},\ i = 1, 2, ..., n_\Gamma$ is the $i$th unit vector. Algorithm 2.6 can readily be applied.

### 3.4.2 Iterative solvers

Another option is to view (3.18) as a minimization problem. This means that we construct the corresponding least squares problem

$$G(\boldsymbol{u}_\Gamma) = \frac{1}{2}\|\boldsymbol{F}(\boldsymbol{u}_\Gamma)\|_2^2 \tag{3.26}$$

and find its minimum using iterative solvers explained in Section 2.5. For gradient descent, this gives the following iterative scheme

$$\boldsymbol{u}_\Gamma^{(k+1)} = \boldsymbol{u}_\Gamma^{(k)} - \gamma_k \mathbb{J}_{\boldsymbol{F}}(\boldsymbol{u}_\Gamma^{(k)})^T \boldsymbol{F}(\boldsymbol{u}_\Gamma^{(k)}), \qquad k = 0, 1, ..., K. \tag{3.27}$$

with suitable step-size $\gamma_k$ and $K$ iterations. Note that $\nabla G(\boldsymbol{u}_\Gamma^{(k)}) = \mathbb{J}_{\boldsymbol{F}}(\boldsymbol{u}_\Gamma^{(k)})^T \boldsymbol{F}(\boldsymbol{u}_\Gamma^{(k)})$ and its Jacobian can be approximated using finite differences just as in (3.25). In each iteration, the step-size is optimized using Backtracking Line Search in Algorithm 2.4.

Another option to solve the minimization problem is to use the BFGS algorithm (see Algorithm 2.5). As initial condition for the Hessian, we choose the identity matrix.

### 3.4.3 Remarks

In each of the methods to solve the non-linear equations in (3.18), we require an initial condition. This is of great importance as it may occur that a bad initial condition finds the wrong solution or even fails to find a solution. It may also influence the rate of convergence. To ensure we start with a suitable initial condition, we have constructed a method aiming to find a good solution. We explain how to find a suitable initial condition for the hybrid-NL.

In the online stage, we solve the coupled problem for an unseen parameter location $\boldsymbol{\mu}^*$. During the offline stage, the reduced-order solutions $\{\boldsymbol{u}_\Gamma(\boldsymbol{\mu}^i)\}_{i=1}^{n_s}$ are given. We choose the initial condition $\boldsymbol{u}_\Gamma^*$ by taking the reduced interface vector that corresponds to the parameter location that is closest to $\boldsymbol{\mu}^*$, that is

$$\boldsymbol{u}_\Gamma^* := \boldsymbol{u}_\Gamma(\boldsymbol{\mu}^i) \quad \text{such that} \quad \|\boldsymbol{\mu}^* - \boldsymbol{\mu}^i\|_2 \leq \|\boldsymbol{\mu}^* - \boldsymbol{\mu}^j\|_2 \qquad \forall j,\ i, j = 1, 2, ..., n_s. \tag{3.28}$$

A similar approach holds for the hybrid-LL but instead we take the reduced-order solutions from the hybrid model.

# 4 Numerical Experiments part 1: linear-linear coupled Poisson problem

In this section, we will show performance results of the surrogate models explained in Section 3 compared with the full-order FEM applied on the linear-linear coupled Poisson problem given in (3.2). We start by computing a sample solution and later work out the surrogate models and give an overview in terms of accuracy and computation time.

## 4.1 Sample solution

Regarding the discretization in the FEM we used first-order Lagrangian elements (see Section 2.1) that are consistent with the domain decomposition such that the total DoFs is $N = 10201$ with $N_1 = N_2 = 5050$ and $N_\Gamma = 101$ denoting the DoFs corresponding to respective domains. A sample solution of the FOM is depicted in Figure 4.1 below.

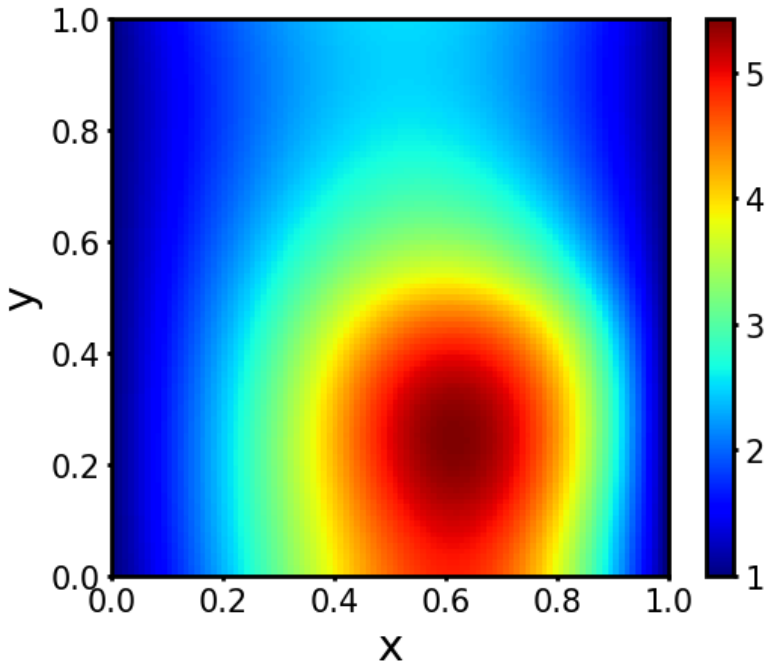



FIGURE 4.1: Sample solution with $\boldsymbol{\mu} = [0.682\ 0.314\ 0.0530]$. The first two elements of $\boldsymbol{\mu}$ correspond to the center location of the source term while the last element corresponds to the intensity.

## 4.2 Numerical results of reduced-order modeling methods

**Offline stage**

We generate $n_s = 500$ snapshots of the FOM with uniformly randomized parameter locations. This gives the snapshot matrix $\mathbb{S} \in \mathbb{R}^{N \times n_s}$ of which we extract $\mathbb{S}_1$, $\mathbb{S}_2$ and $\mathbb{S}_\Gamma$ containing the DoFs corresponding to domains respectively. On each of the snapshot matrices we apply a POD analysis and plot the singular values. See Figure 4.2 below.

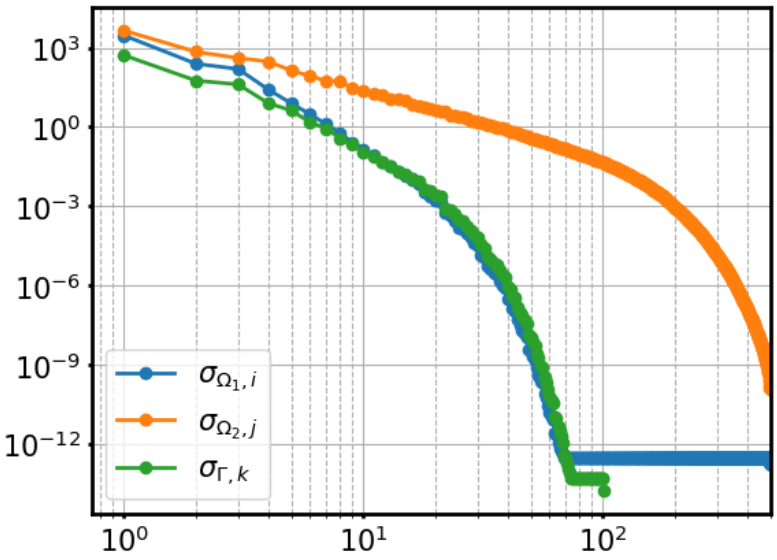



FIGURE 4.2: The singular values are denoted by $\sigma_{\Omega_1,i}$ for $i = 1, 2, ..., N_1$, $\sigma_{\Omega_2,j}$ for $j = 1, 2, ..., N_2$ and $\sigma_{\Omega_\Gamma,k}$ for $k = 1, 2, ..., N_\Gamma$ for each domain respectively. The singular values are plotted in blue, orange and green respectively. We see that the decay on domain $\Omega_2$ is the slowest due to the parametric complexity.

We choose $n_1 = n_\Gamma = n_2 = 5$ modes for each domain, resulting in relative information contents greater than 0.93 on all domains (see equation (2.30)). Regarding the neural networks, the data from the snapshot matrix is split into 80%, 10% and 10% training, validation and testing data respectively and use *LeakyReLU* activation functions. For the optimization we use the Adam optimizer and mini-batches of size 16. The loss function is given by the mean squared error. Each neural network is trained for 1000 epochs and utilizes the early stopping algorithm. The loss curves can be found in Appendix B. The neural network architectures are given in Tables 4.1, 4.2 and 4.3 below.

| Layer | Input | Output |
|---|---|---|
| Dense | 8 | 20 |
| Dense | 20 | 20 |
| Dense | 20 | 1 |

TABLE 4.1: Neural Network architectures for $\hat{\pi}_2^{(i)}$, $i = 1, 2, ..., n_2$.

| Layer | Input | Output |
|---|---|---|
| Dense | 8 | 50 |
| Dense | 50 | 50 |
| Dense | 50 | 50 |
| Dense | 50 | 50 |
| Dense | 50 | 5 |

TABLE 4.2: Neural Network architecture of $\hat{\boldsymbol{t}}_\Gamma$.

| Layer | Input | Output |
|---|---|---|
| Dense | 3 | 50 |
| Dense | 50 | 50 |
| Dense | 50 | 50 |
| Dense | 50 | 50 |
| Dense | 50 | 5 |

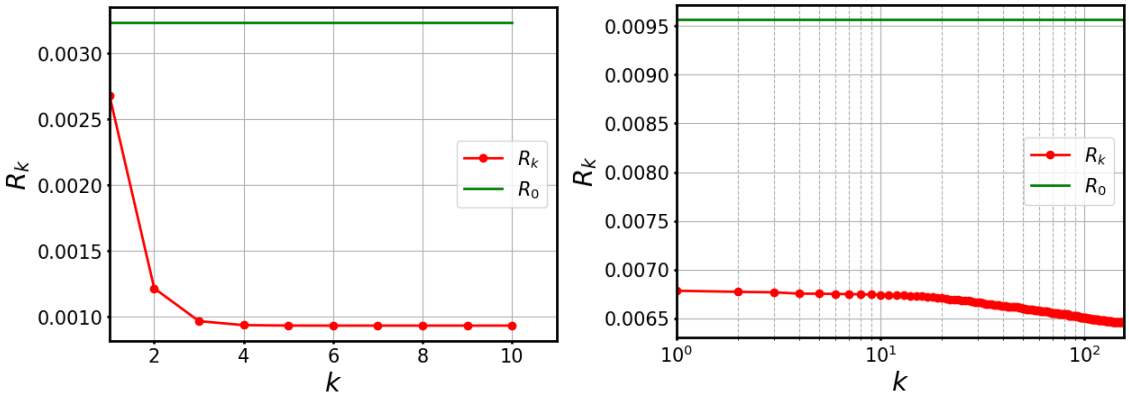
TABLE 4.3: Neural Network architecture of $\hat{\boldsymbol{\pi}}_\Gamma$.

**Online Stage**

We solve a new Poisson problem for the unseen parameter locations

$$\boldsymbol{\mu}^1_{\text{test}} = [0.550\ 0.386\ 0.0271] \quad \text{and} \quad \boldsymbol{\mu}^2_{\text{test}} = [0.920\ 0.169\ 0.0120].$$

For the hybrid-LL and hybrid-NL, we solve the non-linear equation in (3.18) using gradient descent with backtracking line search. We took an initial stepsize $\gamma = 10$, threshold $\epsilon = 1\text{e}{-7}$ and approximated the jacobian using forward finite differences with $h = 1\text{e}{-5}$. The control parameters for the backtracking line search are $\tau = c = 0.5$. See Figure 4.4 below.



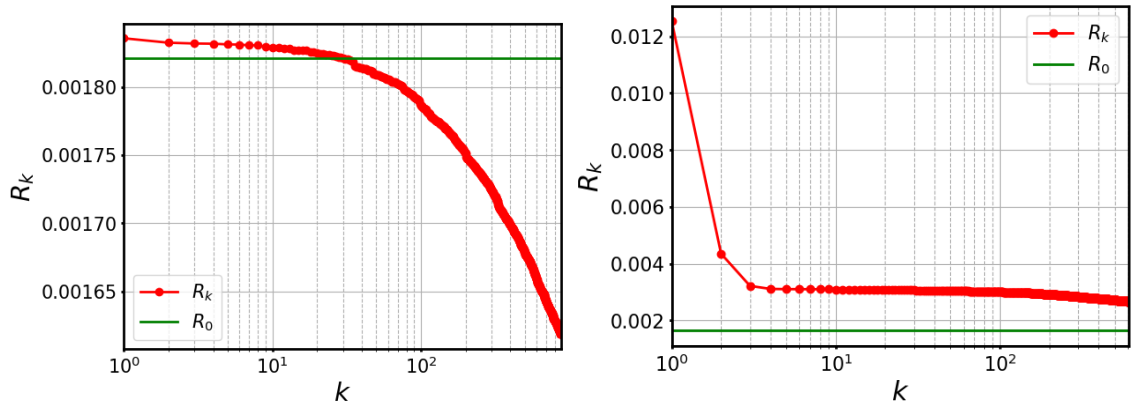



FIGURE 4.4: For $\boldsymbol{\mu}^1_{\text{test}}$ (top) and $\boldsymbol{\mu}^2_{\text{test}}$ (bottom) we show a residual plot for the hybrid-LL (left) and hybrid-NL (right) respectively. The residuals in the $k$-th iteration are given by $R_k = \|F(\boldsymbol{u}^{(k)}_\Gamma)\|_2$ (see equation (3.18)). The target residual is denoted by $R_0 := \|F(\boldsymbol{u}^{\text{exact}}_\Gamma)\|_2$ which is based on the FOM solution on the interface. Due to the errors of the mapping $\hat{\boldsymbol{t}}_\Gamma$ from the training stage, this target value is not exactly 0.

For $\boldsymbol{\mu}^1_{\text{test}}$ we see that the residuals start already below the target value. The choice of the initial condition is designed to start close to the optimal value. As the traction mapping $\hat{\boldsymbol{t}}_\Gamma$ minimizes the error over the complete training set, it may occur that the residuals start under the target value and in turn finds even smaller residuals as the gradient descent algorithm optimizes over just one data point. It should be noted that smaller residuals do not necessarily lead to better results. After all, the output of the gradient descent algorithm should be close to $\boldsymbol{u}^{\text{exact}}_\Gamma$. We report the relative errors compared to the exact reduced-order solutions $\boldsymbol{u}^{\text{exact}}_\Gamma$ in Table 4.4 below.

| | Reduced relative error on $\Gamma$ | | | |
|---|---|---|---|---|
| | Hybrid-LL | Surrogate-LL | Hybrid-NL | Surrogate-NL |
| $\boldsymbol{\mu}^1_{\text{test}}$ | 0.0670 | 0.0106 | 0.0671 | 0.0184 |
| $\boldsymbol{\mu}^2_{\text{test}}$ | 0.345 | 0.0164 | 0.345 | 0.0214 |

TABLE 4.4: Relative errors compared to their exact reduced-order solutions $\boldsymbol{u}^{\text{exact}}_\Gamma$ for $\boldsymbol{\mu}^1_{\text{test}}$ and $\boldsymbol{\mu}^2_{\text{test}}$. We note high errors for the hybrid-LL and hybrid-NL for $\boldsymbol{\mu}^2_{\text{test}}$.

Next, we plot the solutions for all methods in Figures 4.5 and 4.6 below.

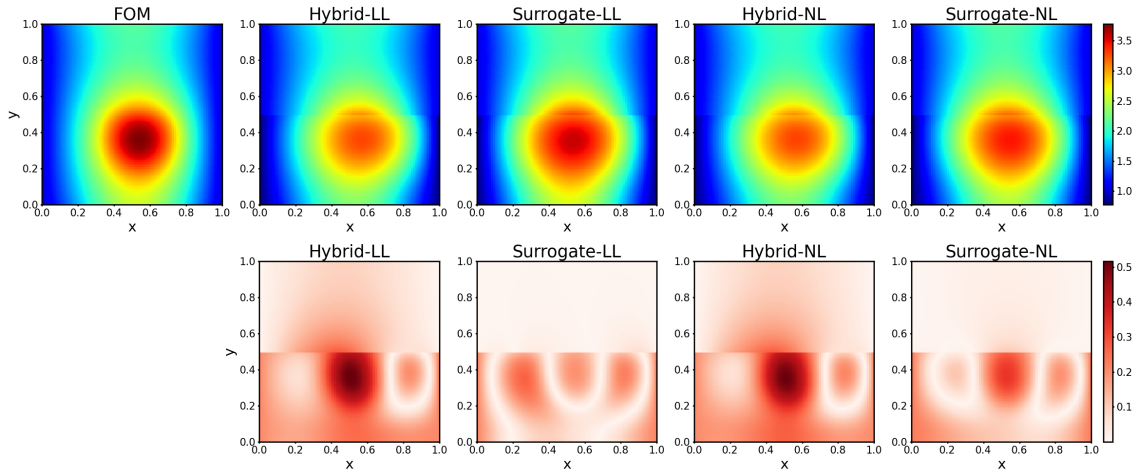



FIGURE 4.5: Solutions for $\boldsymbol{\mu}^1_{\text{test}}$ (top) using FOM and the ROMs from left to right. The absolute errors (bottom) are compared with the FOM. All ROMs show decent correspondence.

| | $\epsilon_1$ | $\epsilon_2$ | $\epsilon_\Gamma$ | $\epsilon$ |
|---|---|---|---|---|
| Hybrid-LL | 0.0493 | 0.0932 | 0.0672 | 0.0790 |
| Surrogate-LL | 0.00470 | 0.0527 | 0.0116 | 0.0412 |
| Hybrid-NL | 0.0492 | 0.0943 | 0.0672 | 0.0798 |
| Surrogate-NL | 0.0115 | 0.0594 | 0.0189 | 0.0468 |

TABLE 4.5: Relative errors for each domains for all ROMs for $\boldsymbol{\mu}^1_{\text{test}}$. The surrogate-LL performs best, followed by the surrogate-NL, hybrid-LL and the hybrid-NL.

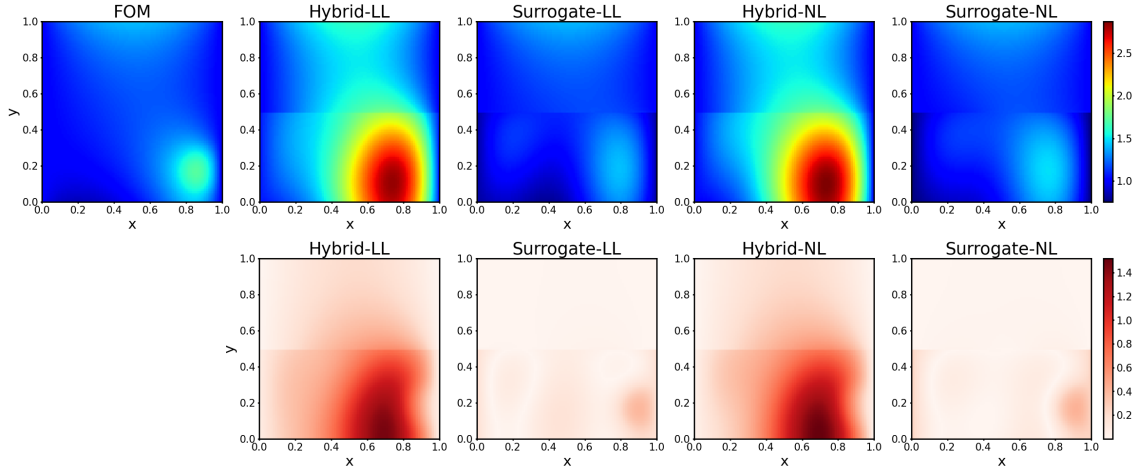



FIGURE 4.6: Solutions for $\boldsymbol{\mu}^2_{\text{test}}$ (top) using FOM and the ROMs from left to right. The absolute errors (bottom) are compared with the FOM. We remark high errors in the hybrid-LL and hybrid-NL.

| | $\epsilon_1$ | $\epsilon_2$ | $\epsilon_\Gamma$ | $\epsilon$ |
|---|---|---|---|---|
| Hybrid-LL | 0.192 | 0.627 | 0.345 | 0.463 |
| Surrogate-LL | 0.00778 | 0.104 | 0.0168 | 0.0732 |
| Hybrid-NL | 0.192 | 0.633 | 0.345 | 0.467 |
| Surrogate-NL | 0.00997 | 0.106 | 0.0215 | 0.0747 |

TABLE 4.6: Relative errors for each domains for all ROMs for $\boldsymbol{\mu}^2_{\text{test}}$. Clearly, the surrogate-LL and surrogate-NL perform better than the hybrid-LL and hybrid-NL.

For each ROM, the values on domain $\Omega_2$ and the $\Gamma$ are determined independently, which results in a discontinuity on the interface. The values on the interface are used to intrusively determine the values on domain $\Omega_1$. From the figures, it can immediately be seen that the surrogate methods perform better than the hybrid methods. In the computation of the reduced interface vector, we already reported large errors in the hybrid methods (see Table 5.1). To determine $\boldsymbol{u}_2$, we use the reduced interface vector as input. This means that if the error is already large on the reduced interface vector, the error is propagated in the computation of $\boldsymbol{u}_2$. This explains the large errors for the hybrid methods. On the other hand, the solution on the interface acts as a Dirichlet boundary condition. This means that the errors on the interface are propagated in the computation of the solution on domain 1.

The offline computation times can be seen in Table 4.7 below.

| | Hybrid-LL | Surrogate-LL | Hybrid-NL | Surrogate-NL |
|---|---|---|---|---|
| Offline CPU ($s$) | 220 | 221 | 207 | 209 |

TABLE 4.7: Offline computation times per ROM. We note that the hybrid-LL and surrogate-LL take longer due to the fact that we need to solve a hybrid model for each parameter location.

For 100 test samples, we computed the average relative errors for each domain and for each method. See Figure 4.7. In Figure 4.8 we reported the online computation times with their relative errors.

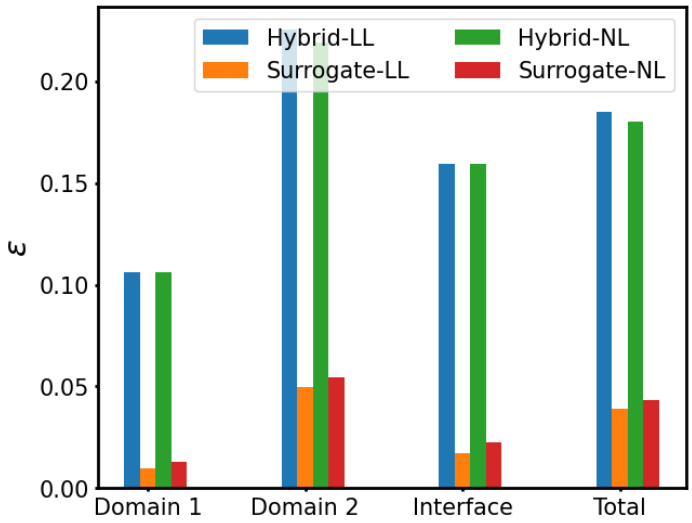



FIGURE 4.7: Average relative errors over 100 test samples for each method. We note high relative errors on domain $\Gamma$ and $\Omega_2$ for the hybrid methods. Overall, the surrogate-LL performs best.

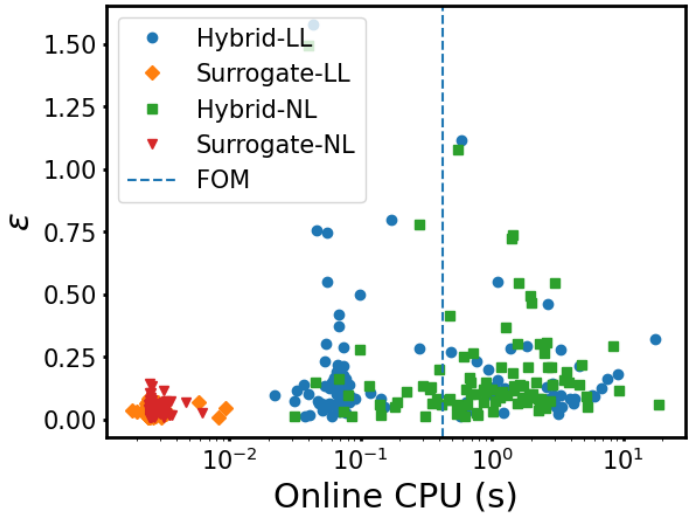



FIGURE 4.8: For the 100 test samples, we report the online computation time against the total relative error for each method. The vertical blue dashed line represents the computation time using the FOM. We see that the solutions for the hybrid methods take substantially longer to compute than for the surrogate methods. Regularly, the online computation time is even longer than that of the FOM. The computation times of the surrogate models is much faster than the FOM. Moreover, they show small variance of the relative error indicating more robustness in contrast to the hybrid models.

To conclude, we have seen that all ROMs can provide tolerable solutions closely related to the FOM. For the hybrid ROMs however, we note that in some cases the reduced-interface vector is determined poorly, resulting in errors propagating in the computation of domain $\Omega_2$. Moreover, the online computation times frequently exceed that of the FOM. On the other hand, the surrogate ROMs indicate better performance in terms of accuracy with low computational cost and a high level of robustness. The surrogate-LL takes longer in the offline stage and performs slightly better than the surrogate-NL in terms of accuracy.

# 5 Numerical Experiments Part 2: linear-nonlinear coupled problem with non-linear diffusion

In this section we report numerical results of the linear-nonlinear coupled problem with non-linear diffusion described in Section 1.1. We compare the hybrid-NL and surrogate-NL with the full-order FEM. Next, we investigate the influence of the number of modes in the reduced-order spaces and the performance of different algorithms with various initial conditions to solve the non-linear equation in (3.18) for the hybrid-NL.

## 5.1 Domain decomposition

Consider the domain described by (1.1) with the steady-state Poisson problem in (1.3). We decompose the domain of the problem in order to separate the linear and non-linear equations on respective domains. Specifically, we split the domain into two non-overlapping regions $\Omega_1$ and $\Omega_2$ such that $\Omega_1$ is the linear domain and $\Omega_2$ the non-linear domain with

$$\overline{\Omega} = \overline{\Omega_1 \cup \Omega_2}, \quad \Omega_1 \cap \Omega_2 = \emptyset, \quad \Omega_1 = \bigcup_{i \in I} \Omega_{1,i}, \quad \bigcap_{i \in I} \overline{\Omega_{1,i}} = \emptyset, \quad I = \{1, 2, 3, 4, 6, 7, 8\}.$$

Let the boundary that connects domains $\Omega_1$ and $\Omega_2$ be called the interface $\Gamma$ such that

$$\Gamma = \partial\Omega_1 \cap \partial\Omega_2,$$

where $\partial\Omega_1$ and $\partial\Omega_2$ denote the boundaries on domain $\Omega_1$ and $\Omega_2$ respectively. See Figure 5.1 on the right.

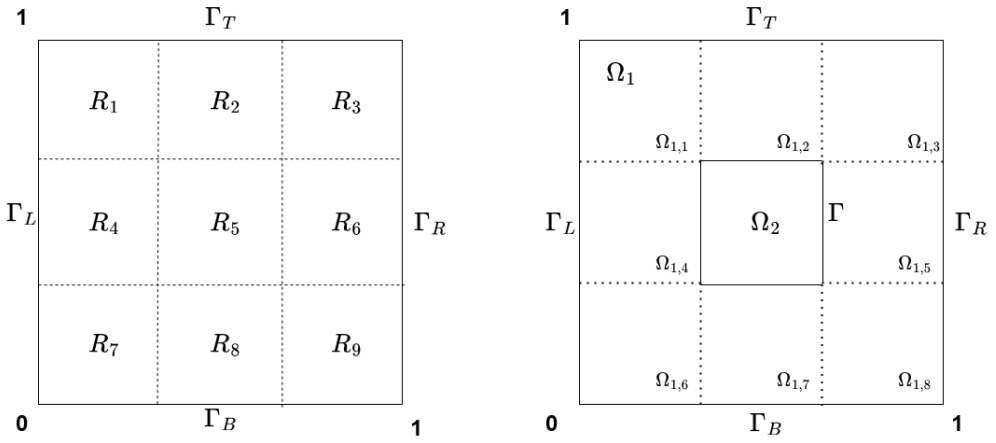



FIGURE 5.1: Original problem (left) and its decomposed problem (right). The decomposed domain allows for a distinction of the linear and non-linear effects on the domains respectively.

The domain decomposition approach leads to the following equations

$$\begin{cases} -\nabla \cdot (k_D(\boldsymbol{x}, \boldsymbol{\mu}) \nabla u_1) = -6 & \text{in} \quad \Omega_1, \\ -\nabla \cdot \mu_5 D(\boldsymbol{x}; \boldsymbol{\mu}) \nabla u_2 = -6 & \text{in} \quad \Omega_2, \\ u_1 = 1 + x^2 + 2y^2 & \text{on} \quad \Gamma_L \cup \Gamma_R, \\ \frac{\partial u_1}{\partial \boldsymbol{n}} = 4y & \text{on} \quad \Gamma_T \cup \Gamma_B, \end{cases} \tag{5.1}$$

with $D(\boldsymbol{x};\boldsymbol{\mu}) = (1 + u\mathbb{1}_{R_5}(\boldsymbol{x}))$ and restricted solutions $u_1$ and $u_2$ to domain $\Omega_1$ and $\Omega_2$ respectively, and

$$k_D(\boldsymbol{x};\boldsymbol{\mu}) = \sum_{i\in I} \mu_i \mathbf{1}_{\Omega_{1,i}} \quad \text{with} \quad \boldsymbol{\mu} = [\mu_1\ \mu_2\ ...\ \mu_9]^T \in \mathcal{P} = [0.1\ 1]^9. \tag{5.2}$$

On the interface we impose two coupling conditions forcing a smooth solution:

$$\begin{cases} \frac{\partial u_1}{\partial \boldsymbol{n}_1} = -\frac{\partial u_2}{\partial \boldsymbol{n}_2} & \text{on } \Gamma, \quad \text{(continuity of normal derivatives)} \\ u_1 = u_2 & \text{on } \Gamma. \quad \text{(continuity of the interface)} \end{cases} \tag{5.3}$$

The domain decomposition has lead to the presence of the non-linearity only on domain $\Omega_2$. The equations (5.1) and (5.3) form the same problem described in (1.3) but with decomposed domains.

## 5.2 Sample solution

Starting with a sample solution of the FOM, the discretization of FEM is performed with first-order Lagrangian elements with $N = 10\ 000$ DoFs such that $N_1 = 8844$, $N_2 = 1024$ and $N_\Gamma = 132$ with $N = N_1 + N_2 + N_\Gamma$.

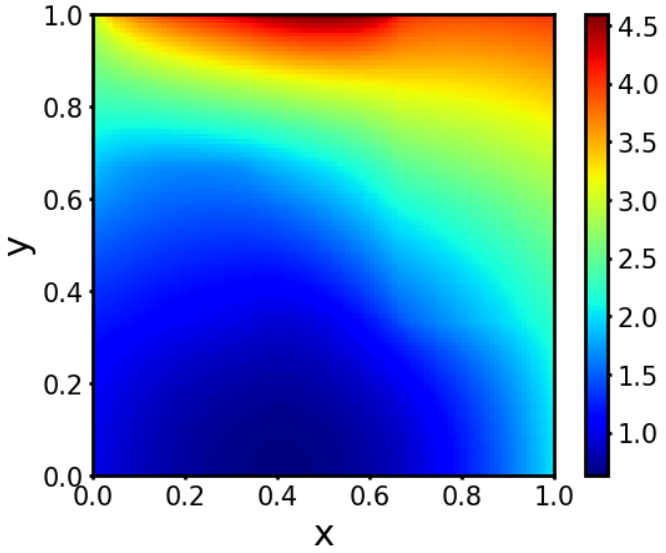



FIGURE 5.2: Sample solution to (1.3) with
$\boldsymbol{\mu} = [0.395\ 0.325\ 0.958\ 0.997\ 0.140\ 0.874\ 0.643\ 0.443\ 0.355]$.

## 5.3 Numerical results of reduced-order modeling methods

**Offline stage**

Using the FOM we generate $n_s = 500$ snapshots and perform a POD analysis on each of the domains. See Figure 5.3.

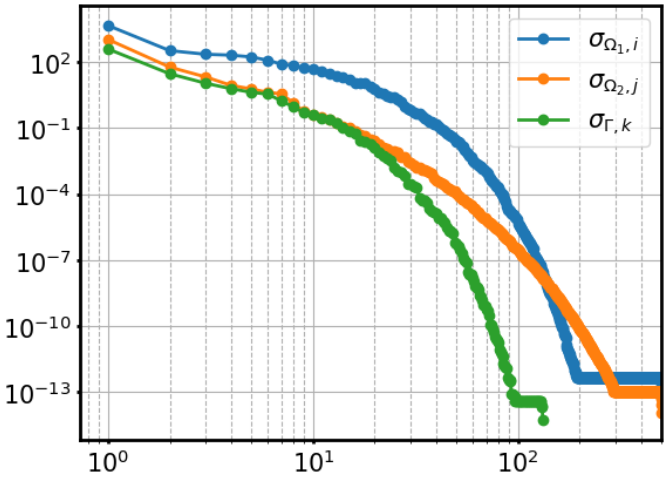



FIGURE 5.3: The singular values are denoted by $\sigma_{\Omega_1,i}$ for $i = 1, 2, ..., N_1$, $\sigma_{\Omega_2,j}$ for $j = 1, 2, ..., N_2$ and $\sigma_{\Omega_\Gamma,k}$ for $k = 1, 2, ..., N_\Gamma$ for each domain respectively. The singular values are plotted in blue, orange and green respectively. We see that the decay on domain $\Omega_1$ is the slowest due to the parametric complexity.

Since the singular value decay is the slowest on domain $\Omega_1$, we choose $n_1 = 10$, $n_2 = n_\Gamma = 5$ for the modes on each domain respectively, resulting in relative information contents greater than 0.96 on all domains. Regarding the neural networks, we used the same hidden layers described in Section 4.2 with Tables 4.1, 4.2 and 4.3.

**Online Stage**

For unseen parameter locations

$$\boldsymbol{\mu}^3_{\text{test}} = [0.280\ 0.142\ 0.558\ 0.549\ 0.926\ 0.951\ 0.748\ 0.229\ 0.409], \tag{5.4}$$

$$\boldsymbol{\mu}^4_{\text{test}} = [0.901\ 0.274\ 0.981\ 0.429\ 0.278\ 0.131\ 0.345\ 0.592\ 0.203] \tag{5.5}$$

we evaluate the performance of the hybrid-NL and surrogate-NL. Note that the fifth element of the parameters indicate the level of non-linearity: $\boldsymbol{\mu}^3_{\text{test}}$ has more non-linearity as opposed to $\boldsymbol{\mu}^4_{\text{test}}$. To solve the non-linear equation in (3.18) for the hybrid-NL, we used gradient descent with backtracking line search using similar hyperparameters described in Section 4.2.

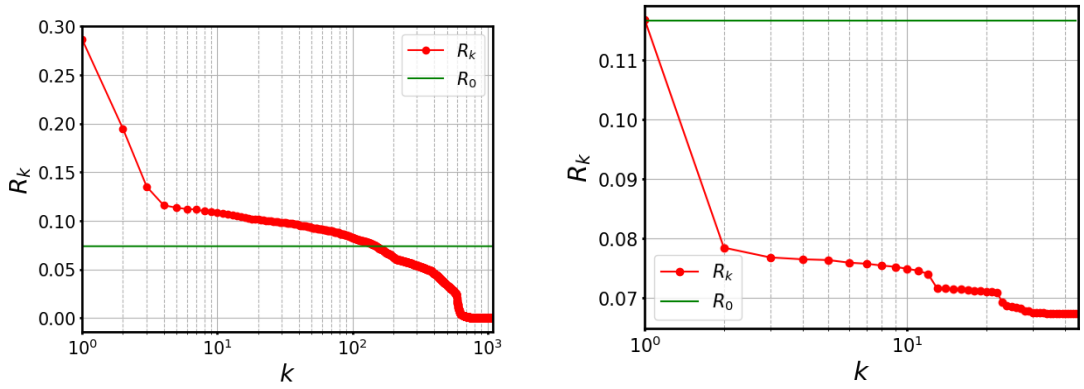



FIGURE 5.4: Gradient Descent for hybrid-NL for $\boldsymbol{\mu}^3_{\text{test}}$ (left) for $\boldsymbol{\mu}^4_{\text{test}}$ (right). Shown is a plot of the residuals such that $R_k = \|F(\boldsymbol{u}_\Gamma^{(k)})\|_2$ is the residual in the $k$-th iteration. The target residual is denoted by $R_0 := \|F(\boldsymbol{u}_\Gamma^{\text{exact}})\|_2$ which is based on the FOM solution on the interface. Due to the error of the mapping $\hat{\boldsymbol{t}}_\Gamma$ from the training stage, this target value is not exactly 0. For both parameters we see that the algorithm terminates below the target value.

We report the reduced-order relative errors for the interface in Table 5.1.

| | Reduced-order relative error on $\Gamma$ | |
|---|---|---|
| | Hybrid-NL | Surrogate-NL |
| $\boldsymbol{\mu}^3_{\text{test}}$ | 0.0163 | 0.0164 |
| $\boldsymbol{\mu}^4_{\text{test}}$ | 0.246 | 0.00491 |

TABLE 5.1: Relative errors compared to their exact reduced-order solutions $\boldsymbol{u}_\Gamma^{\text{exact}}$ for $\boldsymbol{\mu}^3_{\text{test}}$ and $\boldsymbol{\mu}^4_{\text{test}}$. We note a high relative error for the hybrid-NL for $\boldsymbol{\mu}^4_{\text{test}}$.

We remark for $\boldsymbol{\mu}^4_{\text{test}}$ that even though the gradient descent algorithm terminates below the target value, the output still has a high relative error. Next, we plot the reconstructed solutions of the ROMs together with the FOM and plot the absolute errors for both unseen parameter locations. See Figures 5.5 and 5.6 below. The relative errors per domain are given in Tables 5.2 and 5.3.

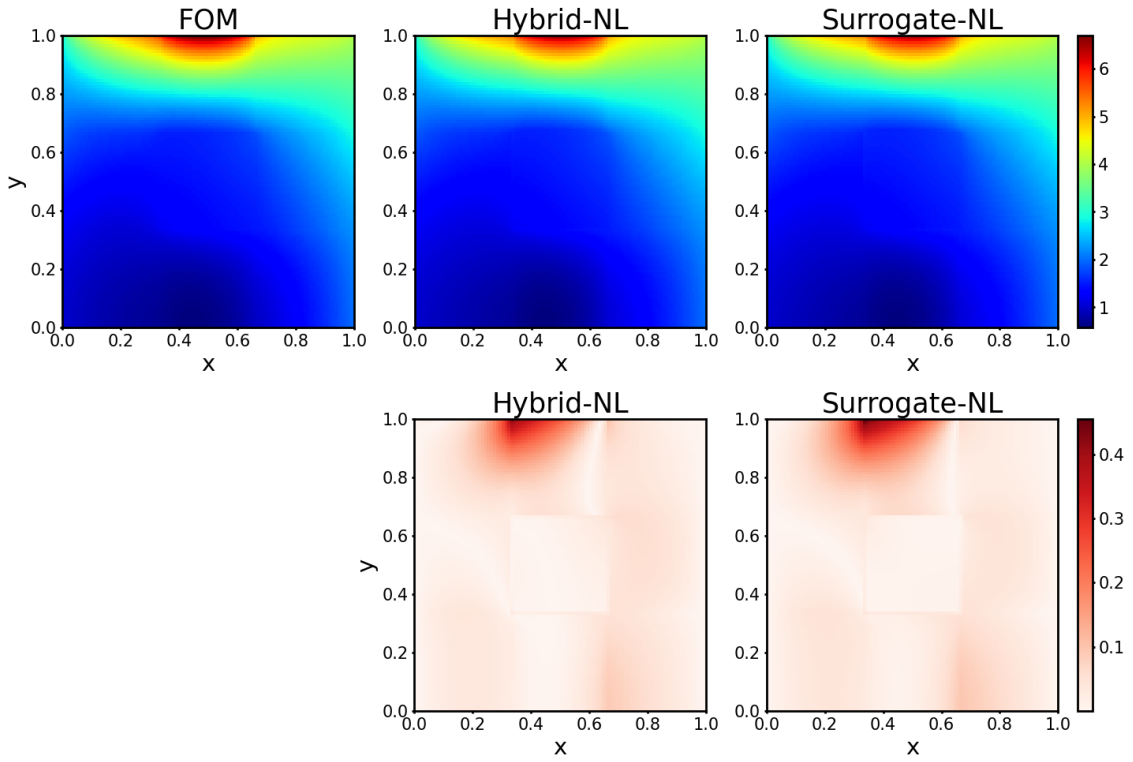



FIGURE 5.5: Solutions for $\boldsymbol{\mu}^3_{\text{test}}$ (top) using FOM and the hybrid-NL and surrogate-NL from left to right. We see a good correspondence between the FOM and both ROMs.

| | $\epsilon_1$ | $\epsilon_2$ | $\epsilon_\Gamma$ | $\epsilon$ |
|---|---|---|---|---|
| Hybrid-NL | 0.0258 | 0.00740 | 0.0615 | 0.0252 |
| Surrogate-NL | 0.0291 | 0.00548 | 0.0616 | 0.0285 |

TABLE 5.2: Relative errors for each domains for the hybrid-NL and surrogate-NL for $\boldsymbol{\mu}^3_{\text{test}}$. Both show low errors.

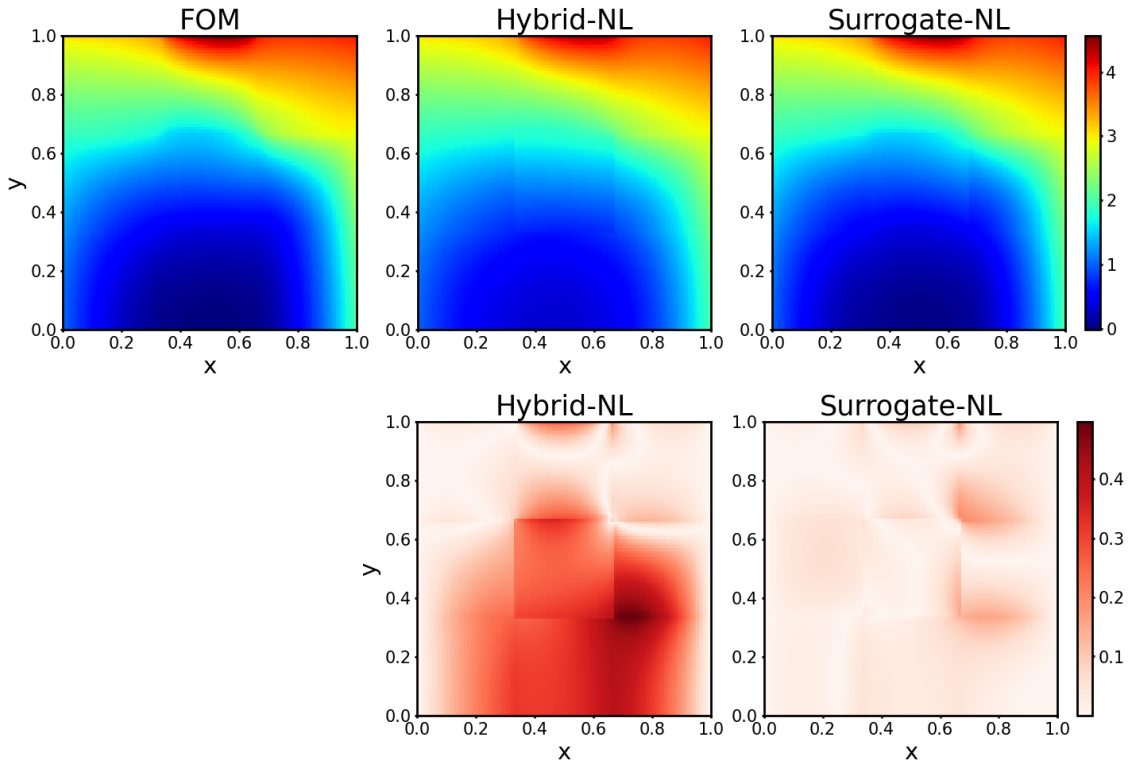



FIGURE 5.6: Solutions for $\boldsymbol{\mu}^4_{\text{test}}$ (top) using FOM and the ROMs from left to right. The absolute errors (bottom) are compared with the FOM. We remark relatively high errors in the hybrid-NL as opposed to the surrogate-NL.

| | $\epsilon_1$ | $\epsilon_2$ | $\epsilon_\Gamma$ | $\epsilon$ |
|---|---|---|---|---|
| Hybrid-NL | 0.102 | 0.257 | 0.247 | 0.111 |
| Surrogate-NL | 0.0236 | 0.0337 | 0.0523 | 0.0241 |

TABLE 5.3: Relative errors for each domain for the hybrid-NL and surrogate-NL with $\boldsymbol{\mu}^4_{\text{test}}$. We remark high relative errors for the hybrid-NL on the interface $\Gamma$ and domain $\Omega_2$.

We remark that the error in the computation of the reduced interface vector is propagated in the computation of the reduced vector on domain $\Omega_2$ for the hybrid-NL. For 100 test samples, we computed the average relative errors for each domain for methods 3 and 4.

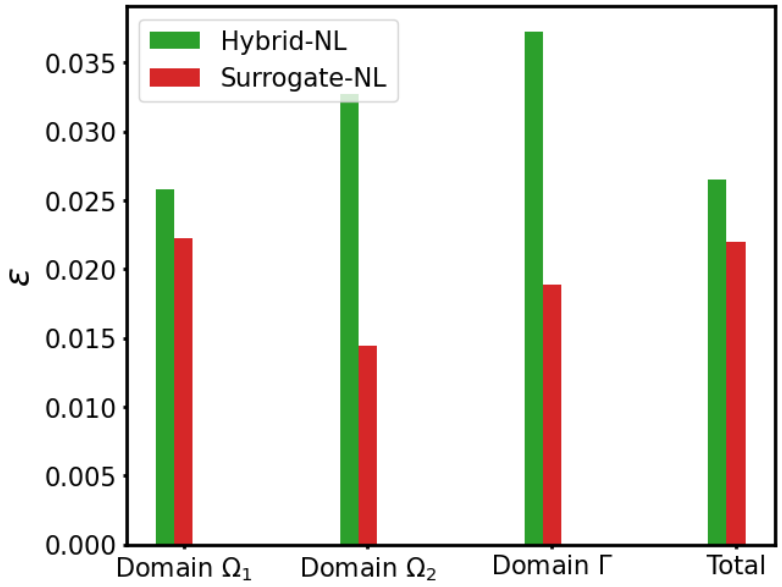



FIGURE 5.7: Average relative errors per domain over 100 test samples. Both models provide low relative errors on all domains. The surrogate-NL yields slightly better results than the hybrid-NL on all domains.

The offline computation times can be seen in Table 5.4

| | Hybrid-NL | Surrogate-NL |
|---|---|---|
| Offline CPU ($s$) | 372 | 323 |

TABLE 5.4: Offline computation times per ROM. We note that the hybrid-NL takes longer since for every snapshot we also require to extract the matrices and vectors to generate $\mathcal{T}_{n_s}$ and $\tilde{\mathcal{T}}_{n_s}$.

In Figure 5.8 we report online computation time against relative error over the same 100 test samples.

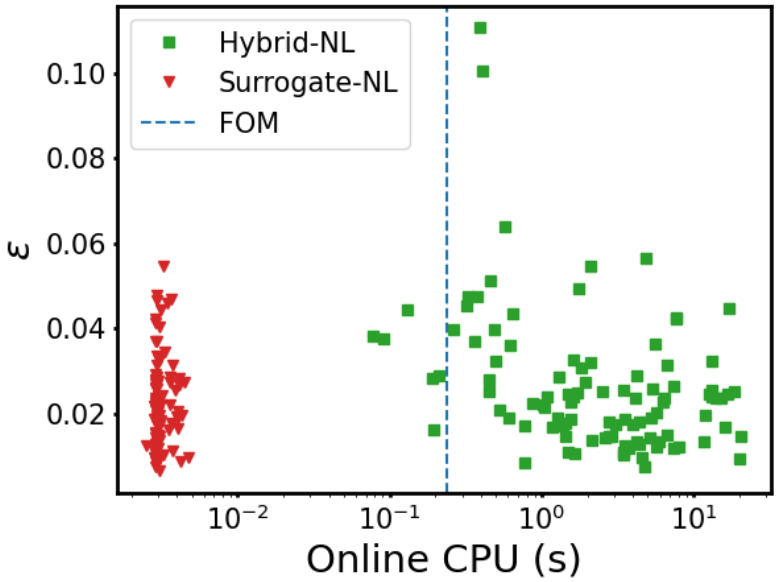



FIGURE 5.8: For the 100 test samples, we report the online computation time against the total relative error for the hybrid-NL and surrogate-NL. The vertical blue dashed line represents the computation time using the FOM. We see that the solutions for the hybrid-NL take substantially longer to compute than for the surrogate-NL. Regularly, the online computation time is even longer than that of the FOM. The computation times of the surrogate-NL is much faster than the FOM. Moreover, they show small variance of the relative error indicating more robustness in contrast to the hybrid-NL.

We conclude that both the hybrid-NL and surrogate-NL provide tolerable solutions. With high variance in the relative errors of the hybrid-NL, the robustness may not be satisfactory. The offline computation time is also substantially longer than the surrogate-NL. Moreover, the online computation time is larger than the computation time of the FOM. In contrast, the surrogate-NL shows a high level of robustness with low relative errors and an extremely small online computation time. Even with the higher level of non-linearity for $\boldsymbol{\mu}^3_{\text{test}}$ than for $\boldsymbol{\mu}^4_{\text{test}}$, the hybrid-NL performs better in the latter case. It remains open why some parameter locations perform poorly in the computation of the reduced interface vector in the hybrid-NL.

## 5.4 Effect of the number of modes

While it may seem natural that adding more modes on each domain increases the accuracy, this is not necessarily true for the reduced-order surrogate models. This is because of error propagation. The reduced-order interface vector obtained either by solving the non-linear equation in (3.18) or via the mapping in (3.20) is paired with some error. This reduced-order interface vector with error is then served as input for computing the reduced-order vector on domain $\Omega_2$ which can cause even larger errors. Regarding robustness of the methods, this may be undesirable. We can make a few changes to improve the robustness although this can decrease the accuracy. For the hybrid-NL and surrogate-NL, we change the input of the neural network for the modes on domain $\Omega_2$, that is

$$\pi_2^{(i)} \; : \; \{\boldsymbol{\mu}_{nlin}\} \longrightarrow \boldsymbol{u}_2^{(i)}, \qquad i = 1, 2, ..., n_2. \quad (5.6)$$

This prevents error propagation as we remove the input of the reduced interface vector. Further, for the surrogate-NL we train separate neural networks for the modes on the interface so that

$$\pi_{\Gamma}^{(i)} \ : \ \{\boldsymbol{\mu}\} \longrightarrow \boldsymbol{u}_{\Gamma}^{(i)}, \qquad i = 1, 2, ..., n_{\Gamma}. \quad (5.7)$$

With the above changes, we will now look at what influence the number of modes on domain $\Omega_2$ and the interface have on the accuracy with the hybrid-NL and surrogate-NL on the linear-nonlinear coupled problem with non-linear diffusion as in (1.3). The ROMs are constructed with the same set-up described in 5.3. All relative errors are based on an average of 100 test samples. First, we look at the relative errors on domain $\Omega_2$. See Figure 5.9 below.

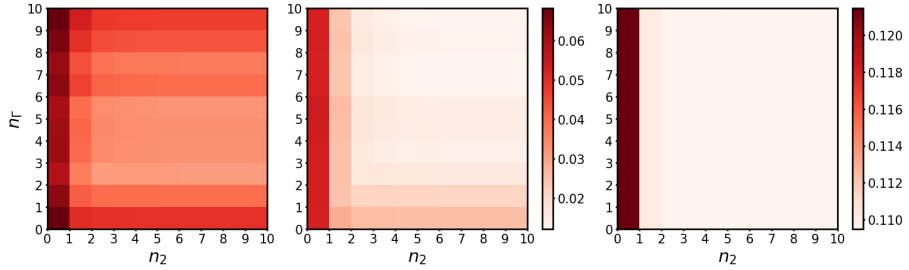



FIGURE 5.9: Relative errors on domain $\Omega_2$ while increasing the number of modes on domain $\Omega_2$ and $\Gamma$ with the old setting (left) for the hybrid-NL (left-left) and surrogate-NL (left-right) and new setting (right) as in (5.6) for both ROMs.

Figure 5.9 shows that in the old setting the relative errors do not strictly decrease as we increase the number of modes for both ROMs. On the other hand, in the new setting the strict decay is guaranteed. However, the relative error now is much larger than in the old setting. Second, we look at the relative erros on domain $\Gamma$ for the surrogate-NL in the old setting and new setting as in (5.7) in Figure 5.10.

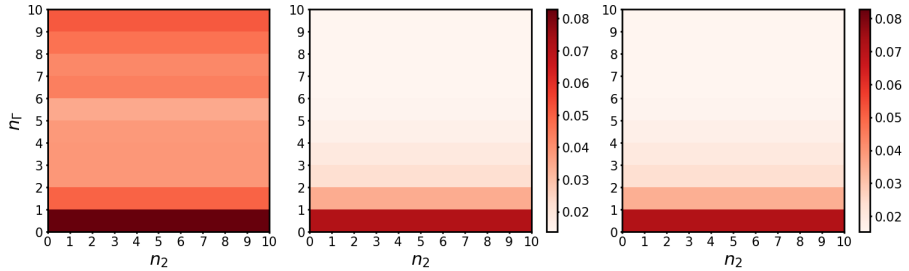



FIGURE 5.10: Relative errors on domain $\Gamma$ while increasing the number of modes on domain $\Omega_2$ and $\Gamma$ with the old setting (left) for the hybrid-NL (left-left) and surrogate-NL (left-right) and new setting (right) as in (5.6) for the surrogate-NL.

For the surrogate-NL we see that the old and new setting barely differ in terms of relative error. For the hybrid-NL, we see that increasing the modes on domain $\Gamma$ does not result in lower relative errors. Lastly, we look at the total relative error in the old and new setting for both ROMs in Figure 5.11.

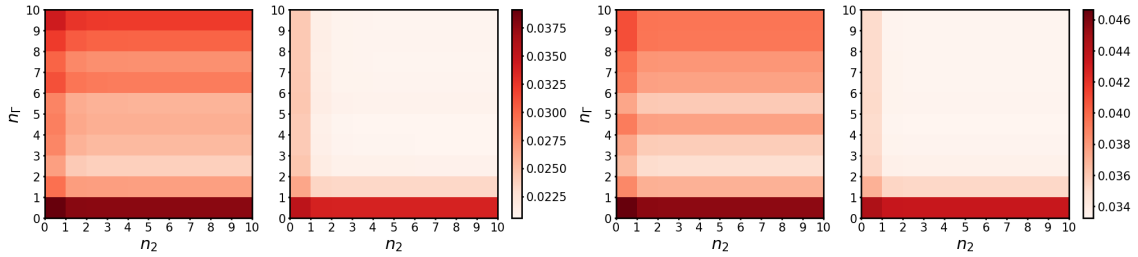



FIGURE 5.11: Total relative errors while increasing the number of modes on domain $\Omega_2$ and $\Gamma$ with the old setting (left) for the hybrid-NL (left-left) and surrogate-NL (left-right) and new setting (right) as in (5.6) for the hybrid-NL (right-left) and surrogate-NL (right-right).

With the new changes, the surrogate-NL is now guaranteed to decrease strictly in total relative error, although at the cost of some accuracy. On the contrary, the hybrid-NL strict relative error decay does not guarantee this property because of the non-strict error decay on domain $\Gamma$. The accuracy gain of the surrogate-NL while increasing the number of modes is minimal.

To conclude, have seen that the new neural networks in (5.6) to determine the reduced vector on domain $\Omega_2$ are paired with significant error increase for this particular problem. The new neural networks in (5.7) to determine the reduced-interface vector did not show significant differences. The total relative errors with the new setting are only slightly higher than in the old case but this is because domain $\Omega_2$ is much smaller than domain $\Omega_1$. Generally, increasing the size of the reduced-order spaces in the ROMs only contributes minimally to the error decrease. Strict error decay benefits the robustness yet it must be carefully investigated per problem to prevent significant error loss.

## 5.5 Comparison of Gradient Descent, Newton's method and BFGS

In the hybrid-NL, the non-linear equations in (3.18) need to be solved. This can be done using methods such as Gradient Descent, Newton's method for finding roots and the BFGS algorithm as explained in Section 3.4. Here, we look more into detail of the performance of these methods for different initial conditions.

With the same training procedure in Section 5.3 we have trained the hybrid-NL. For the unseen parameter locations $\boldsymbol{\mu}^3_{\text{test}}$ described in (5.4) and

$$\boldsymbol{\mu}^5_{\text{test}} = [0.620\ 0.697\ 0.769\ 0.164\ 0.694\ 0.528\ 0.360\ 0.541\ 0.762],$$

we solve the non-linear equations using the different solvers. During the offline stage, we obtained the set of reduced-order interface vectors $\{\boldsymbol{u}_\Gamma(\boldsymbol{\mu}^i)\}_{i=1}^{n_s}$. Using this data, we construct 4 different initial conditions: 'Good IC', 'Max IC', 'Min IC' and 'Avg IC'. The 'Good IC' corresponds to the choice as in (3.28). The 'Max IC', 'Min IC' and 'Avg IC' correspond to the maximum, minimum and average values over each element of the $\boldsymbol{u}_\Gamma(\boldsymbol{\mu}^i)$, $i = 1, 2, ..., n_s$. In Figure 5.12 we have plotted for each method the residuals against the iterations. In tables 5.5, 5.6 and 5.7 we reported the relative errors.

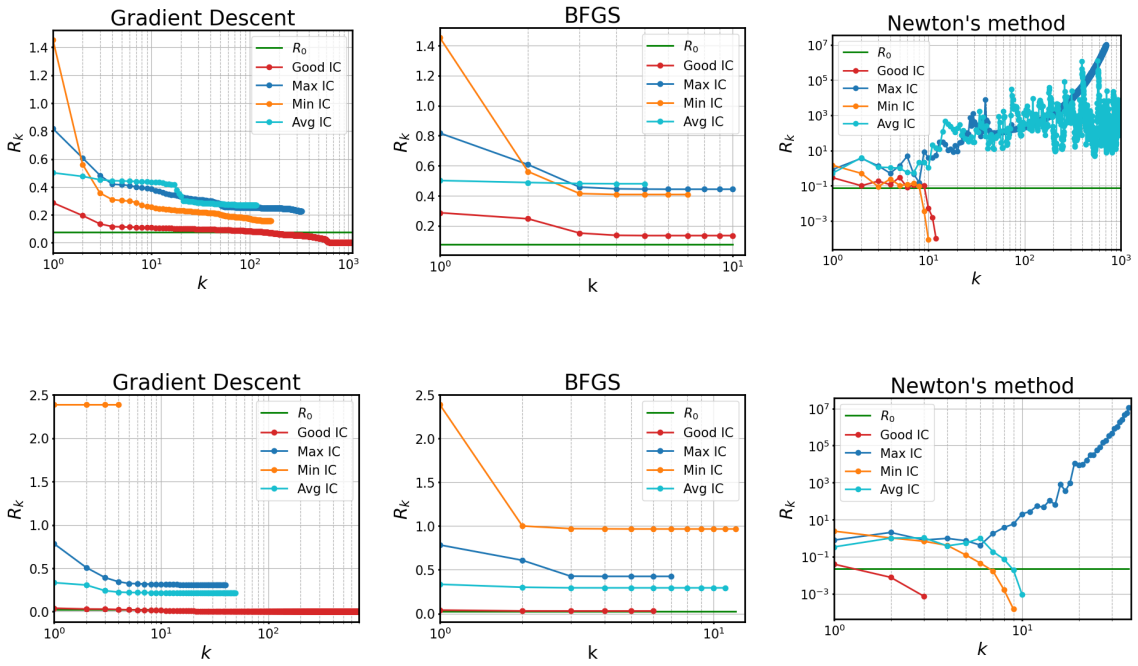



FIGURE 5.12: Residual plots of Gradient Descent (left), BFGS (middle) and Newton's method (right) for different initial conditions for $\boldsymbol{\mu}^3_{\text{test}}$ (top) and $\boldsymbol{\mu}^5_{\text{test}}$ (bottom). The initial condition greatly affects the outputs of the algorithms. Regarding convergence, we see that in some cases Newton's method fails to get convergence. On average, the BFGS algorithm converges after the fewest iterations, followed by Newton's method and gradient descent.

| | Reduced-order relative error on $\Gamma$ | | | |
|---|---|---|---|---|
| | Good IC | Max IC | Min IC | Avg IC |
| $\boldsymbol{\mu}^3_{\text{test}}$ | 0.0163 | 0.475 | 0.225 | 0.266 |
| $\boldsymbol{\mu}^5_{\text{test}}$ | 0.0104 | 0.411 | 0.492 | 0.259 |

TABLE 5.5: Reduced-order relative errors on $\Gamma$ for different initial conditions for $\boldsymbol{\mu}^3_{\text{test}}$ and $\boldsymbol{\mu}^5_{\text{test}}$ using gradient descent

| | Reduced-order relative error on $\Gamma$ | | | |
|---|---|---|---|---|
| | Good IC | Max IC | Min IC | Avg IC |
| $\boldsymbol{\mu}^3_{\text{test}}$ | 0.145 | 0.601 | 0.335 | 0.256 |
| $\boldsymbol{\mu}^5_{\text{test}}$ | 0.0201 | 0.397 | 0.437 | 0.258 |

TABLE 5.6: Reduced-order relative errors on $\Gamma$ for different initial conditions for $\boldsymbol{\mu}^3_{\text{test}}$ and $\boldsymbol{\mu}^5_{\text{test}}$ using the BFGS algorithm.

| | Reduced-order relative error on $\Gamma$ | | | |
|---|---|---|---|---|
| | Good IC | Max IC | Min IC | Avg IC |
| $\boldsymbol{\mu}^3_{\text{test}}$ | 0.0163 | 2.75e8 | 0.0163 | 2.67e8 |
| $\boldsymbol{\mu}^5_{\text{test}}$ | 0.0104 | 3.38e6 | 0.0104 | 0.0104 |

TABLE 5.7: Reduced-order relative errors on $\Gamma$ for different initial conditions for $\boldsymbol{\mu}^3_{\text{test}}$ and $\boldsymbol{\mu}^5_{\text{test}}$ using Newton's method.

From Table 5.5, Table 5.6 and 5.7 we see that for the 'Good IC' the gradient descent algorithm and Newton's method achieve the lowest relative errors. In Newton's method however, the convergence is not always guaranteed for different initial conditions. Regarding robustness, this is not desired. The BFGS algorithm stops after very few iterations and performs poorly when the initial condition is far away. We conclude that the Gradient Descent algorithm performs best among the methods.

# 6 Numerical Experiments Part 3: linear-nonlinear coupled problem with non-linear reaction

In this section we report numerical results of the linear-nonlinear coupled problem with non-linear Rayleigh-Bénard reaction. Starting with the problem description and corresponding domain decomposition, we evaluate the surrogate models by comparing them to the full-order FEM through the finite element method.

## 6.1 Problem description

Let $\mathcal{N}_R$ be a non-linear reaction operator given by

$$\mathcal{N}_R u = -\nabla \cdot (k(\boldsymbol{x}; \boldsymbol{\mu}) \nabla u) + R(u; \boldsymbol{\mu}) \tag{6.1}$$

with $R(u; \boldsymbol{\mu}) = \mu_{10} u(1+u^2)\mathbb{1}_{R_5}(\boldsymbol{x})$. This term is referred to as Rayleigh-Bénard convection which is a form of convection existing in fluid dynamics [50]. Consider the domain described by (1.1) with the steady-state Poisson problem in (1.3) but now with the non-linear reaction operator in (6.1). We decompose the domain of the problem in order to separate the linear and non-linear equations on respective domains. In particular, we have that

$$\begin{cases} -\nabla \cdot (k_D(\boldsymbol{x}; \boldsymbol{\mu}) \nabla u_1) = -6 & \text{in} \quad \Omega_1, \\ -\nabla \cdot (\mu_5 \nabla u_2) + R(\boldsymbol{x}; \boldsymbol{\mu}) = -6 & \text{in} \quad \Omega_2, \\ u_1 = 1 + x^2 + 2y^2 & \text{on} \quad \Gamma_L \cup \Gamma_R, \\ \frac{\partial u_1}{\partial \boldsymbol{n}} = 4y & \text{on} \quad \Gamma_T \cup \Gamma_B, \end{cases} \tag{6.2}$$

with $\mu_{10} \in [0.01,\ 10]$ and $\boldsymbol{\mu} \in [0.1,\ 1]^9 \times [0.01,\ 10]$ and the coupling conditions described in (5.3).

## 6.2 Numerical results of reduced-order modeling methods

**Offline stage**

Again, using the FOM we generate $n_s = 500$ snapshots and perform a POD analysis on each of the domains. The singular value decay is given in Figure 6.1 below.

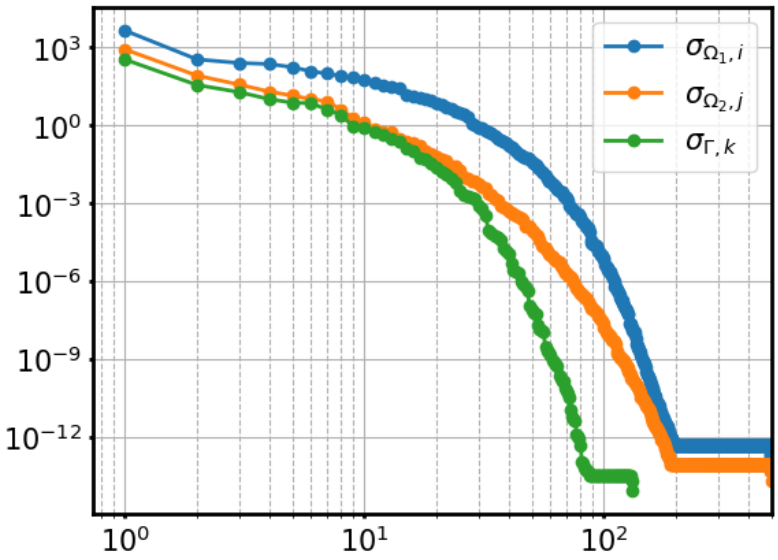



FIGURE 6.1: The singular values are denoted by $\sigma_{\Omega_1,i}$ for $i = 1, 2, ..., N_1$, $\sigma_{\Omega_2,j}$ for $j = 1, 2, ..., N_2$ and $\sigma_{\Omega_\Gamma,k}$ for $k = 1, 2, ..., N_\Gamma$ for each domain respectively. The singular values are plotted in blue, orange and green respectively. We see that the decay on domain $\Omega_1$ is the slowest due to the parametric complexity.

We choose $n_1 = 10$, $n_2 = n_\Gamma = 5$ for the number of modes on each domain respectively. The set-up of the neural networks is similar to the ones described in Section 4.2 with Tables 4.1, 4.2 and 4.3.

**Online stage**
For the unseen parameter locations

$$\boldsymbol{\mu}^4_{\text{test}} = [0.788\ 0.935\ 0.160\ 0.208\ 0.233\ 0.854\ 0.573\ 0.835\ 0.639\ 1.43]$$
$$\boldsymbol{\mu}^5_{\text{test}} = [0.224\ 0.221\ 0.760\ 0.227\ 0.193\ 0.233\ 0.231\ 0.456\ 0.199\ 6.13]$$

we compare the FOM with the hybrid-NL and surrogate-NL. Note that the last element of the parameter locations corresponds to the non-linearity. Regarding the hybrid-NL, the reduced-order interface vector is determined by solving (3.18) using gradient descent with backtracking line search using similar hyperparameters described in Section 4.2. See Figure 6.2.

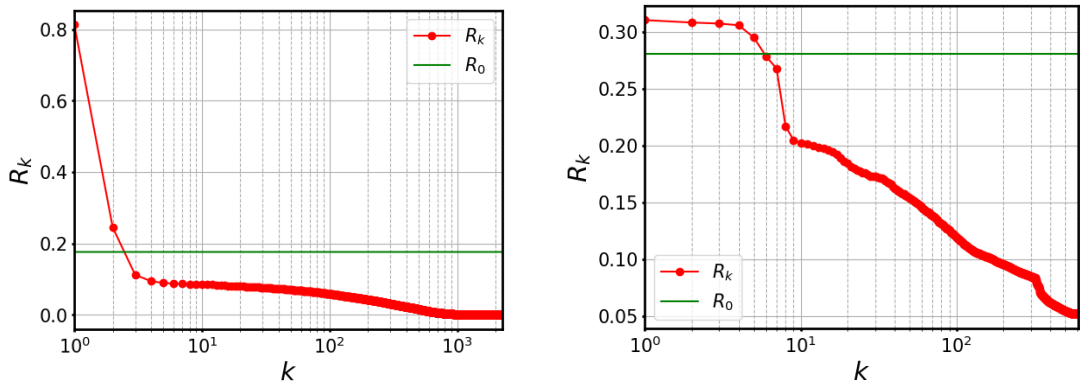



FIGURE 6.2: Gradient Descent for hybrid-NL for $\boldsymbol{\mu}^4_{\text{test}}$ (left) for $\boldsymbol{\mu}^5_{\text{test}}$ (right). Shown is a plot of the residuals such that $R_k = \|F(\boldsymbol{u}_\Gamma^{(k)})\|_2$ is the residual in the $k$-th iteration. The target residual is denoted by $R_0 := \|F(\boldsymbol{u}_\Gamma^{\text{exact}})\|_2$ which is based on the FOM solution on the interface. Due to the error of the mapping $\hat{\boldsymbol{t}}_\Gamma$ from the training stage, this target value is not exactly 0. For both parameters we see that the algorithm terminates below the target value.

The reduced-order relative errors for the interface are given in Table 6.1.

| | Reduced-order relative error on $\Gamma$ | |
|---|---|---|
| | Hybrid-NL | Surrogate-NL |
| $\boldsymbol{\mu}^4_{\text{test}}$ | 0.0534 | 0.0125 |
| $\boldsymbol{\mu}^5_{\text{test}}$ | 0.364 | 0.0757 |

TABLE 6.1: Relative errors compared to their exact reduced-order solutions $\boldsymbol{u}_\Gamma^{\text{exact}}$ for $\boldsymbol{\mu}^4_{\text{test}}$ and $\boldsymbol{\mu}^5_{\text{test}}$. We note a high relative error for the hybrid-NL for $\boldsymbol{\mu}^5_{\text{test}}$.

We remark that for $\boldsymbol{\mu}^5_{\text{test}}$ that even though the gradient descent algorithm terminates well below the target value, the output still shows a high relative error. The reconstructed solutions of the ROMs are compared with the FOM for both parameter locations in Figure 6.3 and 6.4 with relative errors reported in Tables 6.2 and 6.3.

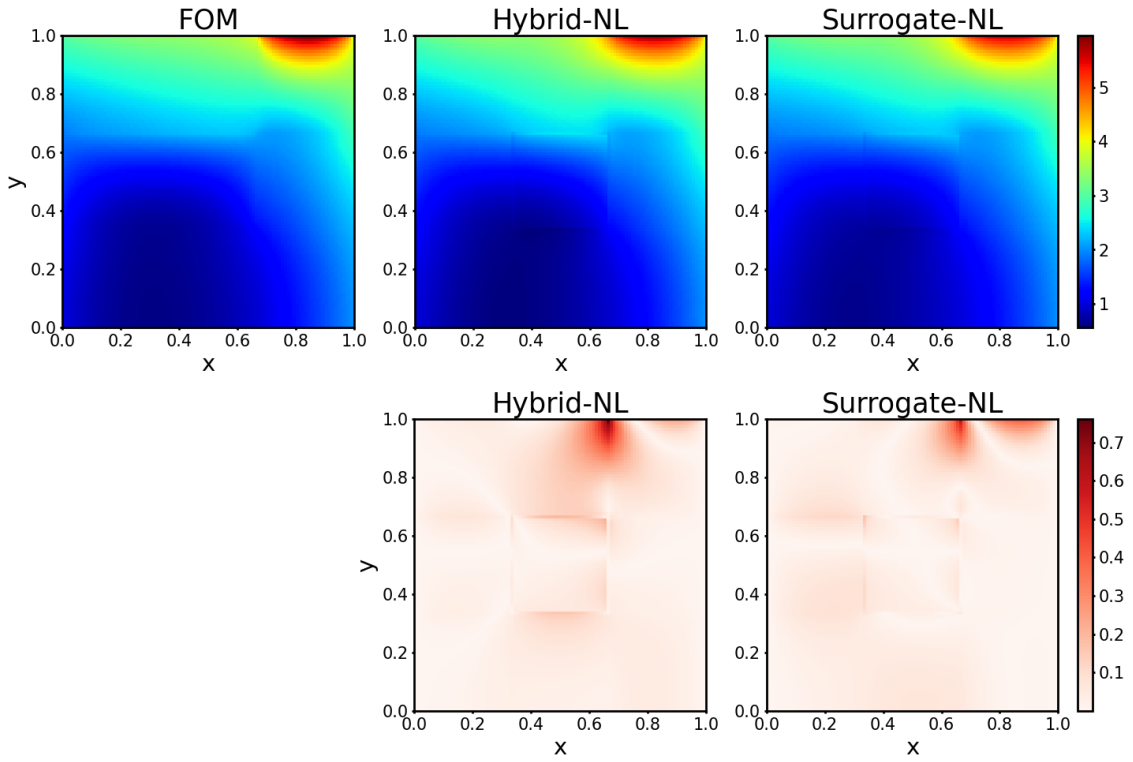



FIGURE 6.3: Solutions for $\boldsymbol{\mu}^4_{\text{test}}$ (top) using FOM and the hybrid-NL and surrogate-NL from left to right. We see a good correspondence between the FOM and both ROMs.

| | $\epsilon_1$ | $\epsilon_2$ | $\epsilon_\Gamma$ | $\epsilon$ |
|---|---|---|---|---|
| Hybrid-NL | 0.0381 | 0.0403 | 0.0616 | 0.0385 |
| Surrogate-NL | 0.0289 | 0.0231 | 0.0332 | 0.0287 |

TABLE 6.2: Relative errors for each domains for the hybrid-NL and surrogate-NL for $\boldsymbol{\mu}^4_{\text{test}}$. Both show low errors.

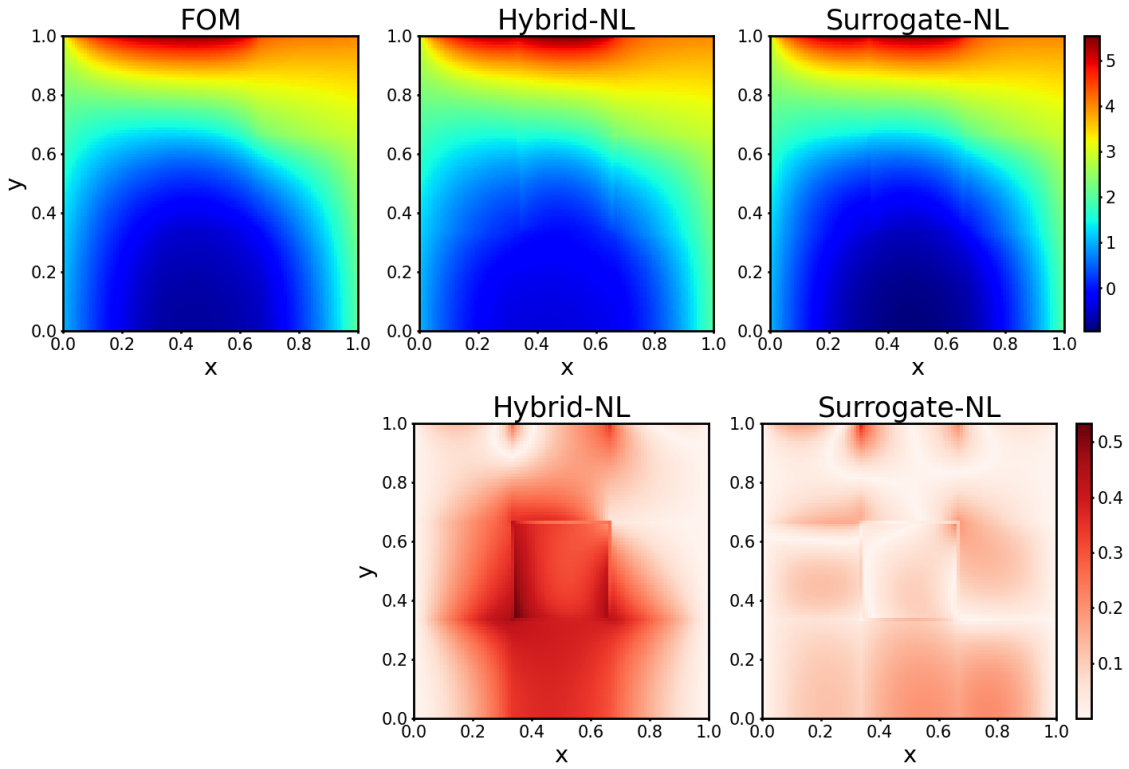



FIGURE 6.4: Solutions for $\boldsymbol{\mu}^2_{\text{test}}$ (top) using FOM and the ROMs from left to right. The absolute errors (bottom) are compared with the FOM. We remark relatively high errors in the hybrid-NL as opposed to the surrogate-NL.

| | $\epsilon_1$ | $\epsilon_2$ | $\epsilon_\Gamma$ | $\epsilon$ |
|---|---|---|---|---|
| Hybrid-NL | 0.0967 | 0.579 | 0.364 | 0.113 |
| Surrogate-NL | 0.0444 | 0.101 | 0.0830 | 0.0454 |

TABLE 6.3: Relative errors for each domains for the hybrid-NL and surrogate-NL for $\boldsymbol{\mu}^5_{\text{test}}$. There is a high relative error for the hybrid-NL on the interface that is propagated in the computation on domain $\Omega_2$.

For 100 test samples, we computed the average relative errors for each domain for the hybrid-NL and surrogate-NL. See Figure 6.5.

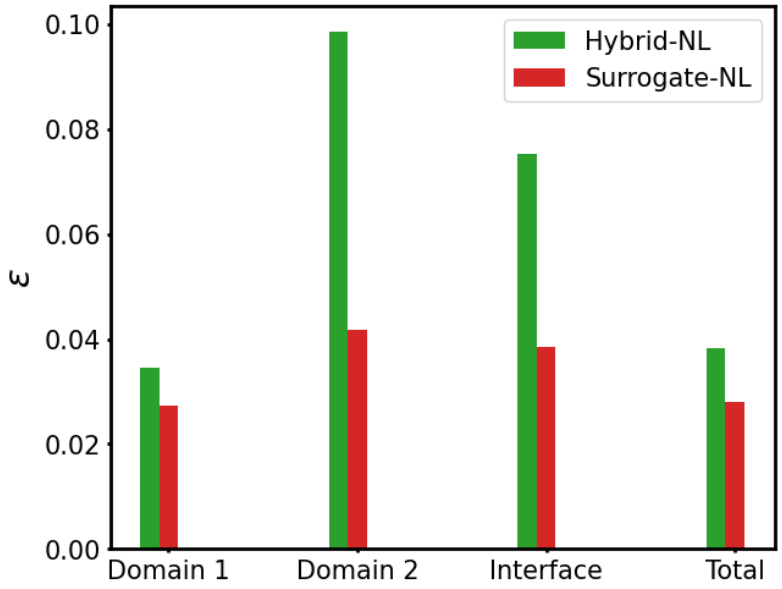



FIGURE 6.5: Average relative errors per domain over 100 test samples. Both models provide tolerable relative errors on all domains. The surrogate-NL performs better on all domains compared to the hybrid-NL.

The offline computation times can be seen in Table 6.4

| | Hybrid-NL | Surrogate-NL |
|---|---|---|
| Offline CPU ($s$) | 348 | 295 |

TABLE 6.4: Offline computation times per ROM. We note that the hybrid-NL takes longer since for every snapshot we also require to extract the matrices and vectors to generate $\mathcal{T}_{n_s}$ and $\tilde{\mathcal{T}}_{n_s}$.

In Figure 6.6 we have plotted the relative errors against the online computation time for the 100 test samples.

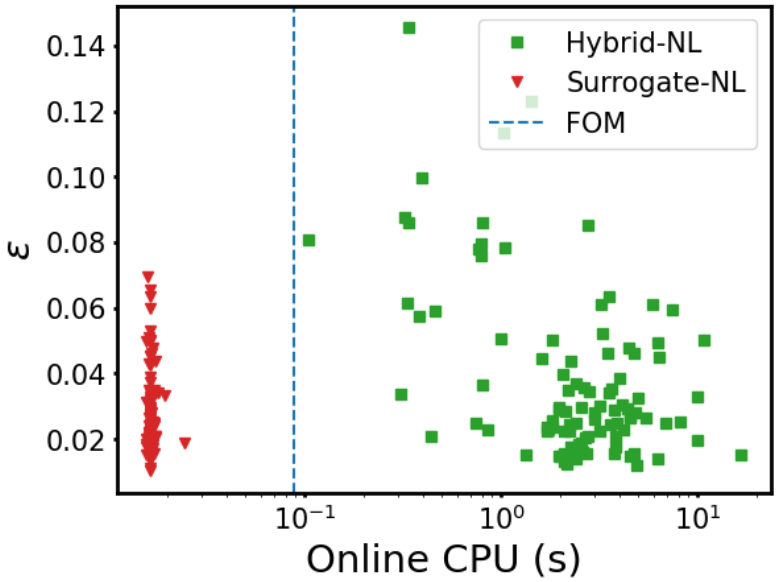



FIGURE 6.6: For the 100 test samples, we report the online computation time against the total relative error for the hybrid-NL and surrogate-NL. The vertical blue dashed line represents the computation time using the FOM. We see that the solutions for the hybrid-NL take substantially longer to compute than for the surrogate-NL. The online computation times are even longer than that of the FOM. The computation times of the surrogate-NL is much faster than the FOM. Moreover, they show small variance of the relative error indicating more robustness in contrast to the hybrid-NL.

The conclusion is similar to that in previous section. Both the hybrid-NL and surrogate-NL provide tolerable solutions but for the hybrid-NL the computational gain is unsatisfactory and the relative erros are on average higher compared to the surrogate-NL.

# 7 Numerical Experiments Part 4: Uncertainty quantification using Monte Carlo simulation

In this section we perform several uncertainty quantification tasks for PDEs using Monte Carlo simulation. While this is an expensive task for full-order models in terms of computational cost, reduced-order models address this issue.

## 7.1 Estimating statistical information

This section covers the estimation of an expected value. Consider the linear-nonlinear coupled problem with non-linear reaction described in (6.2). Let the output of interest be given by the average value on domain 2, i.e.

$$F(u(\boldsymbol{x};\boldsymbol{\mu})) = \frac{1}{|\Omega_2|}\int_{\Omega_2} u(\boldsymbol{x};\boldsymbol{\mu})\ d\Omega_2 \tag{7.1}$$

$$\approx \frac{1}{N_2}\sum_{i=1}^{N_2} U_2^{rb,(i)}, \tag{7.2}$$

which is approximated by the average value of the elements of the vector $\boldsymbol{U}_2^{rb}$. Suppose the quantity of interest is given by the expectation, that is

$$\mathbb{E}[F] = \int_{\mathcal{P}} F(u(\boldsymbol{x};\boldsymbol{\mu}))\rho(\boldsymbol{\mu})\ d\boldsymbol{\mu}. \tag{7.3}$$

Using Monte Carlo Simulation, we aim to approximate (7.3) using the surrogate-NL from Algorithm 3.4 and compare the results with the FOM. We investigate the influence of the number of modes on domain $\Omega_2$ and the interface. Specifically, we construct a reduced-order model using $n_2 = n_\Gamma = 2$ denoted by ROM-2 and one using $n_2 = n_\Gamma = 5$ denoted by ROM-5. Iterating over $Q = 1000$ samples, we compute $\mathbb{E}[F]$ for both the ROMs and FOM.

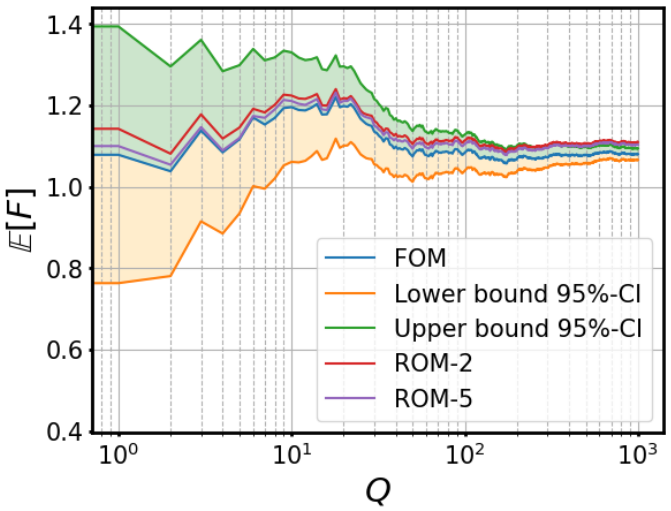



FIGURE 7.1: Convergence of the quantity of interest $\mathbb{E}[F]$ for both the FOM and two ROMs using different RB sizes. After every iteration $q \leq Q$ we compute the quantity of interest and the 95% confidence interval of the FOM. Between the FOM and ROMs there is a small bias as the ROMs slightly overestimate the quantity of interest. We note that ROM-5 performs slightly better than ROM-2.

## 7.2 Estimating failure probability

In this section, we will approximate a failure probability using Monte Carlo simulations with and without importance sampling described in Section 2.7.1. As approximating the failure probability using the FOM is too costly, we instead utilize the surrogate-NL.

Consider the linear-linear coupled Poisson problem described in (3.2) but with $\mathcal{P} = [0\ 0.5] \times [0\ 0.25] \times [0.001\ 0.1]$. Let the output of interest be given by the average value on the interface. This can be approximated using the solutions from the surrogate model, that is

$$\begin{aligned} F(u(\boldsymbol{x};\boldsymbol{\mu})) &= \frac{1}{|\Gamma|}\int_{\Gamma} u(\boldsymbol{x};\boldsymbol{\mu})\ d\Gamma \\ &\approx \frac{1}{N_\Gamma}\sum_{j=1}^{N_\Gamma} U_\Gamma^{rb,(i)}. \end{aligned} \tag{7.4}$$

Suppose we assign a 1 (failure) for when the output of interest is greater than 3.9 and assign 0 (success) else. This gives

$$G(u(\boldsymbol{x};\boldsymbol{\mu})) = \begin{cases} 1 & \text{if} \quad F(u(\boldsymbol{x};\boldsymbol{\mu})) > F_0 = 3.9, \\ 0 & \text{else.} \end{cases} \tag{7.5}$$

The quantity of interest is the failure probability, that is

$$P_{F_0} = P[F(u(\boldsymbol{x};\boldsymbol{\mu})) \geq F_0] = \int_{\mathcal{P}} G(u(\boldsymbol{x};\boldsymbol{\mu}))\rho(\boldsymbol{\mu})\ d\boldsymbol{\mu}. \tag{7.6}$$

Using regular Monte Carlo simulation, the failure probability is approximated such that

$$P_{F_0} \approx P_{F_0,MC} = \frac{1}{Q}\sum_{q=1}^{Q} G(\boldsymbol{U}^{rb}(\boldsymbol{x};\boldsymbol{\mu}^q)).$$

As we have seen in Section 2.7.1, estimating small probabilities is more effectively accomplished through the utilization of importance sampling. For that, we first construct a suitable biasing distribution $q(\boldsymbol{\mu})$. Prior, the parameter locations were taken from independent uniform distributions so that the original distribution is a multiplication of three uniform distributions, that is

$$p(\boldsymbol{\mu}) = p_1(\mu^{(1)})p_2(\mu^{(2)})p_3(\mu^{(3)}), \tag{7.7}$$

$$p_1(\mu^{(1)}) = \begin{cases} 2 & \text{for} \quad 0 \leq \mu^{(1)} \leq 0.5, \\ 0 & \text{else,} \end{cases} \qquad p_2(\mu^{(2)}) = \begin{cases} 4 & \text{for} \quad 0 \leq \mu^{(2)} \leq 0.25, \\ 0 & \text{else,} \end{cases}$$

$$p_3(\mu^{(3)}) = \begin{cases} \frac{1}{0.099} & \text{for} \quad 0.001 \leq \mu^{(3)} \leq 0.1, \\ 0 & \text{else.} \end{cases}$$

For parameter location $\boldsymbol{\mu}_q = [0.482\ 0.213\ 0.0918]$, we know that $F(\boldsymbol{u}(\boldsymbol{x};\boldsymbol{\mu}_q)) > 3.9$. Note that this parameter resides towards the extremity of the parameter space. To favour more

samples around this parameter location, we ensure that the biasing distribution is centered around $\boldsymbol{\mu}_q$. This leads to the following biasing normal distributions

$$q(\boldsymbol{\mu}) = q_1(\mu^{(1)})q_2(\mu^{(2)})q_3(\mu^{(3)}), \tag{7.8}$$

$$q_1(\mu^{(1)}; \mu_q^{(1)}, \sigma_1) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\mu^{(1)} - \mu_q^{(1)}}{\sigma_1}\right)^2\right),$$

$$q_2(\mu^{(2)}; \mu_q^{(2)}, \sigma_2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\mu^{(2)} - \mu_q^{(2)}}{\sigma_2}\right)^2\right) \quad \text{and}$$

$$q_3(\mu^{(3)}; \mu_q^{(3)}, \sigma_3) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{\mu^{(3)} - \mu_q^{(3)}}{\sigma_3}\right)^2\right).$$

In Figure 7.2 we have depicted the prior and biasing distributions.

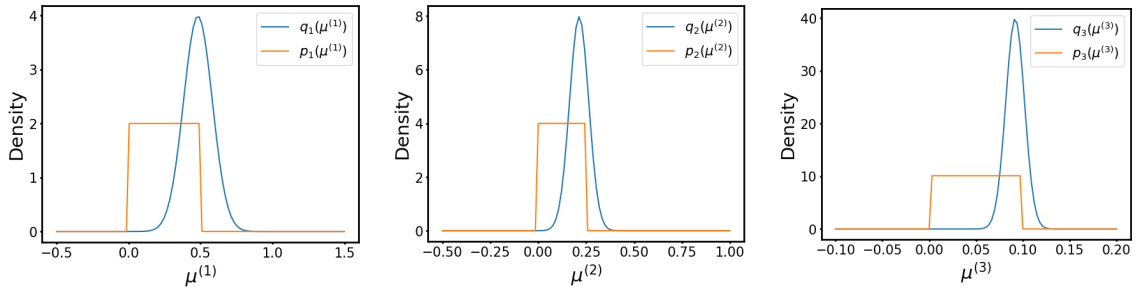



FIGURE 7.2: From left to right, plotted are the original uniform distributions with biasing distributions for $\sigma_1 = \sqrt{0.1}$, $\sigma_2 = \sqrt{0.05}$ and $\sigma_3 = \sqrt{0.01}$ respectively. When sampling from the biasing distribution, we take samples that are more at the extremity of the parameter space.

Using importance sampling, the failure probability is approximated so that

$$P_{F_0} \approx P_{F_0,IS} = \frac{1}{Q}\sum_{i=1}^{Q} G(\boldsymbol{U}^{rb}(\boldsymbol{x}; \boldsymbol{\mu}^i)\frac{\rho(\boldsymbol{\mu}^i)}{q(\boldsymbol{\mu}^i)}.$$

With the same training strategy explained in Section 4.2, we have train the surrogate-NL. For $Q = 60\ 000$ Monte Carlo samples we approximate the failure probability after every sample for both regular Monte Carlo simulation and with importance sampling. The results are depicted in Figure 7.3

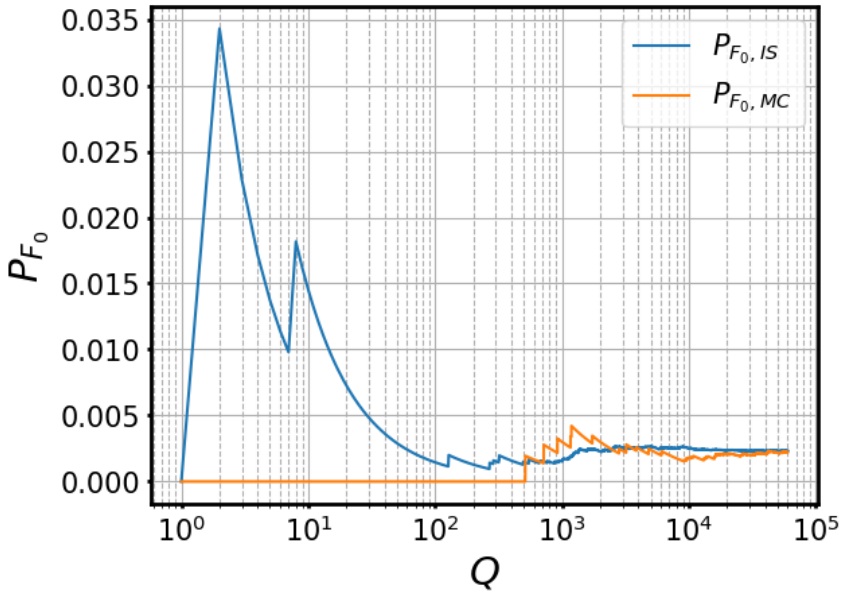



FIGURE 7.3: Estimates $P_{F_0,MC}$, $P_{F_0,IS}$ of failure probabilities using both regular Monte Carlo simulation and with importance sampling respectively. Even though we are unable to compare the failure probability by utilizing full-order models, we can still see that the approximated failure probability using importance sampling converges to the same failure probability using regular Monte Carlo simulation. With importance sampling, we see that we converge faster than in the regular case. The failure probabilities are given by $P_{F_0,MC} = 0.00222$ and $P_{F_0,IS} = 0.00233$.

## 7.3 Estimation of probability density function

Consider the linear-nonlinear coupled problem with non-linear diffusion described in (1.3). Let the output of interest be given by the average value on domain $\Omega_2$ as in (7.1) and the quantity of interest the expected average value on domain $\Omega_2$, similar to (7.3) and denoted by $\mathbb{E}[F]$. We aim to approximate the probability density function (PDF) of $\mathbb{E}[F]$ using the surrogate-NL.

Let the PDF estimate be denoted by $\hat{f}_X$. A natural approach would be to run multiple Monte Carlo simulations to obtain sample expectations and construct a discrete PDF. This requires a high computational cost. For that reason we use another method called bootstrapping. Instead of running multiple Monte Carlo simulations, we generate one simulation of $Q$ samples. By taking a random sub-sample of size $M$ of the $Q$ samples $K$ times, we compute estimate sample expectations $\hat{\mathbb{E}}_i[F]$ for $i = 1, 2, ..., K$. From these samples, we generate a histogram of equal-sized intervals of length $\Delta$ and note that

$$P(\hat{\mathbb{E}}_i[F] \leq X \leq \hat{\mathbb{E}}_i[F] + \Delta) = \frac{k_i}{K}, \qquad i = 1, 2, ..., K \quad (7.9)$$

where $k_i$ denotes the frequency corresponding to the interval $[\hat{\mathbb{E}}_i[F],\ \hat{\mathbb{E}}_i[F] + \Delta]$. At $\hat{\mathbb{E}}_i[F]$ the approximate PDF value is given by

$$\hat{f}_X(\hat{\mathbb{E}}_i[F]) = \frac{P(\hat{\mathbb{E}}_i[F] \leq X \leq \hat{\mathbb{E}}_i[F] + \Delta)}{\Delta}, \qquad i = 1, 2, ..., K. \quad (7.10)$$

Before estimating the PDF, we run a Monte Carlo simulation of $Q = 1000$ samples for both the FOM and ROM (in this case surrogate-NL). We report the average values on

domain $\Omega_2$ in a histogram and in each iteration calculate the estimated expected value. See Figures 7.4 and 7.5 below.

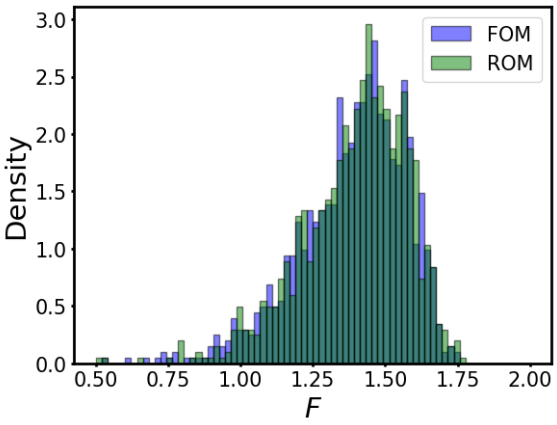



FIGURE 7.4: Histogram of the outputs of interest $F$ for both the FOM and ROM for $Q = 1000$ samples with relative frequency on the y-axis. We see a high level of correspondence but the ROM seems to slightly overestimate the outputs of interests.

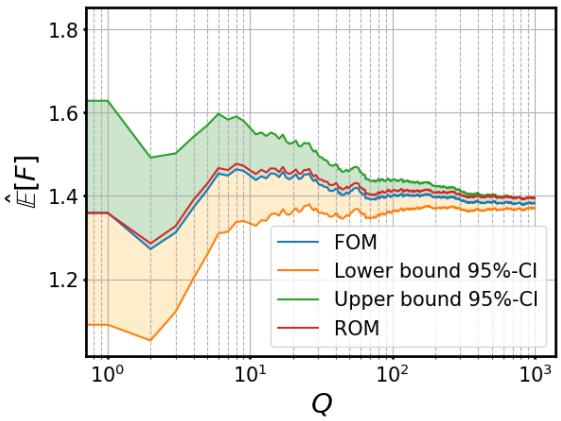



FIGURE 7.5: Convergence of the estimate $\hat{\mathbb{E}}[F]$ for both the FOM and surrogate-NL. After every iteration $q \leq Q$ we estimate the quantity of interest and the 95% confidence interval of the FOM. Between the FOM and the surrogate-NL there is a small bias as the surrogate-NL slightly overestimates the quantity of interest.

Proceeding with the PDF estimation, in the bootstrapping phase we generate $Q = 50000$ samples and take $K = 30000$ random sub-samples of size $M = 4000$. See Figure 7.6.

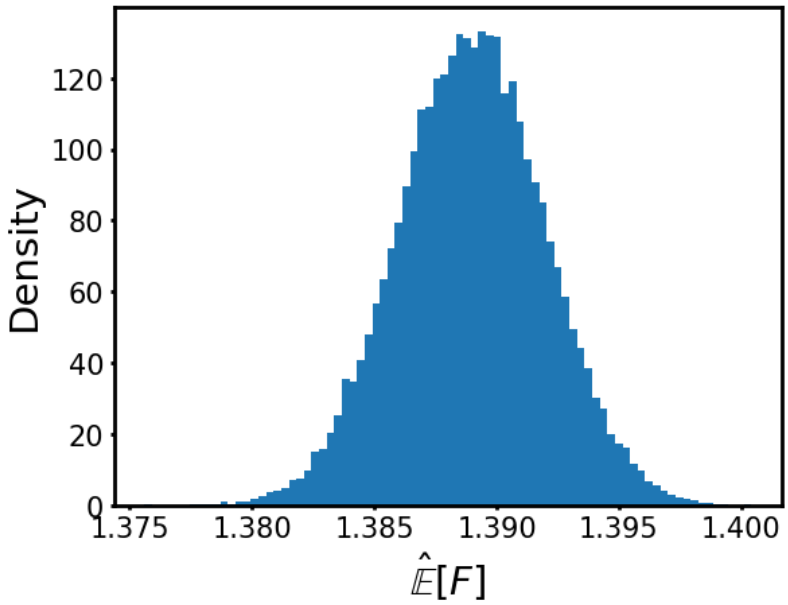



FIGURE 7.6: PDF estimation of $\mathbb{E}[F]$ with the surrogate-NL using bootstrapping. The length of the intervals are given by $\Delta = 0.000311$.

# 8 Discussion and conclusions

In this thesis, we developed reduced-order surrogate models for linear-nonlinear coupled problems. The aim of these models is to address the limitations when utilizing conventional full-order techniques such as FEM that require prohibitive computational costs. The hybrid-LL and surrogate-LL are two developed reduced-order methods for linear-linear coupled problems acting as proof of concepts, while the ROMs termed hybrid-NL and surrogate-NL are intended to solve linear-nonlinear coupled problems. The four ROMs combine the use of both intrusive and non-intrusive methods with POD-ROM, POD-NN and domain decomposition on sub-domains respectively. The choices of the reduction techniques on the sub-domains are based on the parametric complexity and the underlying equations (linear or non-linear) in order to apply reduction effectively.

Starting with the linear-linear coupled problem discussed in Section 4, we have seen that all ROMs guarantee solutions. The surrogate ROMs consistently achieve low relative errors contributing to a high level of robustness. In contrast, the hybrid ROMs are unable to achieve suitable relative errors compared to the FOM. Equally, the error deviations are much larger in the hybrid ROMs. For particular parameter locations, the hybrid ROMs fail to find a good estimate for the reduced interface values of which their error propagates through the computations on domain $\Omega_1$ and especially $\Omega_2$. Regarding computational cost, the offline stage is clearly expensive for all methods and especially for the hybrid ROMs as they require a substantial amount of memory cost at the same time. For the surrogate ROMs however, the expensive offline stage is easily compensated for in the extremely fast online stage.

Continuing with the linear-nonlinear coupled problems in Section 5 and 6, we see similar behaviour of the performance of the hybrid-NL and surrogate-NL. Although guaranteeing solutions, the hybrid-NL fails to fulfill the purpose of a ROM with often larger online computations than the full-order method. The issue on error propagation on domain $\Omega_2$ for both ROMs could be resolved by omitting the reduced interface vector as input for the neural networks although this significantly reduces accuracy of the solution. Generally, increasing the size of the reduced-order spaces in the ROMs only contributes minimally to the error decrease.

In Section 7, the power of the surrogate-NL is successfully tested in uncertainty quantification tasks in Monte Carlo simulations. Showing a small bias compared to the FOM for particular statistical quantities, this is readily counteracted by the noticeably more rapid online computation time.

While the ROMs have shown to work in our numerical experiments, it should be important to note that there exist numerous hyperparameters that need to be optimized depending on the problem. For example, the number of snapshots required in the offline stage as well as the tuning of the hyperparameters in the neural networks is highly dependent on the parametric complexity. Tuning the hyperparameters per problem may be a challenging task.

In conclusion, this thesis has made progress in solving linear-nonlinear coupled problems effectively by constructing ROMs that employ domain decomposition and both regular and data-driven reduced-order modeling techniques. Especially the surrogate-NL has shown to

be promising under our numerical experiments and it will be interesting to see the performance on more challenging tasks.

# 9 Future research

First, one could aim to address the limitations of the hybrid-NL. As the hybrid-NL fails to find an accurate reduced-interface vector at times with high computational cost, future work could focus on this issue. The key is to find out why the hybrid-NL underachieves for particular parameter locations. A comparison of some iterative solvers such as the BFGS algorithm and Newton's method has already been discussed in Section 5.5 which may lead to new insights.

Second, the numerical results for the surrogate-NL are promising. It will be interesting to see how the surrogate-NL performs under more challenging problems by adding time dependence for advection-diffusion-reaction equations such as in [36]. Another common problem of interest is the modeling of fluid flow around a cylinder and investigated in [34].

Lastly, while most ROMs in literature are paired with a posteriori error bounds which may be important in practical applications, this has been left unaddressed here. This is useful to control the participating errors.

# 10 References


[1] J. Parry, "Mathematical modelling and computer simulation of heat and mass transfer in agricultural grain drying: A review," *Journal of Agricultural Engineering Research*, vol. 32, no. 1, pp. 1–29, 1985.

[2] W.-C. Chuang, H.-L. Lee, P.-Z. Chang, and Y.-C. Hu, "Review on the modeling of electrostatic mems," *Sensors*, vol. 10, no. 6, pp. 6149–6171, 2010.

[3] H. Mohammed, G. Bhaskaran, N. Shuaib, and R. Saidur, "Heat transfer and fluid flow characteristics in microchannels heat exchanger using nanofluids: a review," *Renewable and Sustainable Energy Reviews*, vol. 15, no. 3, pp. 1502–1512, 2011.

[4] T. Belytschko, R. Gracie, and G. Ventura, "A review of extended/generalized finite element methods for material modeling," *Modelling and Simulation in Materials Science and Engineering*, vol. 17, no. 4, p. 043001, 2009.

[5] S. S. Rao, *The finite element method in engineering*. Butterworth-heinemann, 2017.

[6] A. Nouy, "Recent developments in spectral stochastic methods for the numerical solution of stochastic partial differential equations," *Archives of Computational Methods in Engineering*, vol. 16, no. 3, pp. 251–285, 2009.

[7] G. Alfonsi, "Reynolds-Averaged Navier–Stokes Equations for Turbulence Modeling," *Applied Mechanics Reviews*, vol. 62, p. 040802, 06 2009.

[8] R. Benzi, S. Succi, and M. Vergassola, "The lattice boltzmann equation: theory and applications," *Physics Reports*, vol. 222, no. 3, pp. 145–197, 1992.

[9] H. Bhabha, "Relativistic wave equations for the elementary particles," *Reviews of Modern Physics*, vol. 17, no. 2-3, p. 200, 1945.

[10] F. Ballarin, A. Manzoni, A. Quarteroni, and G. Rozza, "Supremizer stabilization of pod–galerkin approximation of parametrized steady incompressible navier–stokes equations," *International Journal for Numerical Methods in Engineering*, vol. 102, no. 5, pp. 1136–1161, 2015.

[11] R. C. Smith, *Uncertainty quantification: theory, implementation, and applications*, vol. 12. Siam, 2013.

[12] T. Bui-Thanh, C. Burstedde, O. Ghattas, J. Martin, G. Stadler, and L. C. Wilcox, "Extreme-scale uq for bayesian inverse problems governed by pdes," in *SC'12: Proceedings of the international conference on high performance computing, networking, storage and analysis*, pp. 1–11, IEEE, 2012.

[13] D. Hartmann and H. van der Auweraer, "Digital twins," 2020.

[14] S. A. Niederer, M. S. Sacks, M. Girolami, and K. Willcox, "Scaling digital twins from the artisanal to the industrial," *Nature Computational Science*, vol. 1, no. 5, pp. 313–320, 2021.

[15] R. Yondo, E. Andrés, and E. Valero, "A review on design of experiments and surrogate models in aircraft real-time and many-query aerodynamic analyses," *Progress in aerospace sciences*, vol. 96, pp. 23–61, 2018.

[16] D. J. Knezevic and A. T. Patera, "A certified reduced basis method for the fokker–planck equation of dilute polymeric fluids: Fene dumbbells in extensional flow," *SIAM Journal on Scientific Computing*, vol. 32, no. 2, pp. 793–817, 2010.

[17] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems," *SIAM review*, vol. 57, no. 4, pp. 483–531, 2015.

[18] D. J. Lucia, P. S. Beran, and W. A. Silva, "Reduced-order modeling: new approaches for computational physics," *Progress in aerospace sciences*, vol. 40, no. 1-2, pp. 51–117, 2004.

[19] F. Chinesta, P. Ladeveze, and E. Cueto, "A short review on model order reduction based on proper generalized decomposition," *Archives of Computational Methods in Engineering*, vol. 18, no. 4, pp. 395–404, 2011.

[20] G. Rozza, D. B. P. Huynh, and A. T. Patera, "Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations: application to transport and continuum mechanics," *Archives of Computational Methods in Engineering*, vol. 15, no. 3, pp. 229–275, 2008.

[21] A. E. Løvgren, Y. Maday, and E. M. Rønquist, "A reduced basis element method for the steady stokes problem," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 40, no. 3, pp. 529–552, 2006.

[22] A. Manzoni, "An efficient computational framework for reduced basis approximation and a posteriori error estimation of parametrized navier–stokes flows," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 48, no. 4, pp. 1199–1226, 2014.

[23] M. A. Grepl and A. T. Patera, "A posteriori error bounds for reduced-basis approximations of parametrized parabolic partial differential equations," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 39, no. 1, pp. 157–181, 2005.

[24] A. T. Patera, G. Rozza, *et al.*, "Reduced basis approximation and a posteriori error estimation for parametrized partial differential equations," 2007.

[25] J. S. Hesthaven and S. Ubbiali, "Non-intrusive reduced order modeling of nonlinear problems using neural networks," *Journal of Computational Physics*, vol. 363, pp. 55–78, 2018.

[26] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[27] M. Guo and J. S. Hesthaven, "Data-driven reduced order modeling for time-dependent problems," *Computer methods in applied mechanics and engineering*, vol. 345, pp. 75–99, 2019.

[28] N. Franco, A. Manzoni, and P. Zunino, "A deep learning approach to reduced order modelling of parameter dependent partial differential equations," *Mathematics of Computation*, vol. 92, no. 340, pp. 483–524, 2023.

[29] P.-L. Lions *et al.*, "On the schwarz alternating method. i," in *First international symposium on domain decomposition methods for partial differential equations*, vol. 1, p. 42, Paris, France, 1988.

[30] Y. Maday and F. Magoules, "Absorbing interface conditions for domain decomposition methods: A general presentation," *Computer methods in applied mechanics and engineering*, vol. 195, no. 29-32, pp. 3880–3900, 2006.

[31] A. de Castro, P. Kuberry, I. Tezaur, and P. Bochev, "A novel partitioned approach for reduced order model—finite element model (rom-fem) and rom-rom coupling," *Earth and Space 2022*, pp. 475–489, 2022.

[32] Y. Wu and X.-C. Cai, "A fully implicit domain decomposition based ale framework for three-dimensional fluid–structure interaction with application in blood flow computation," *Journal of Computational Physics*, vol. 258, pp. 524–537, 2014.

[33] A. Corigliano, M. Dossi, and S. Mariani, "Domain decomposition and model order reduction methods applied to the simulation of multi-physics problems in mems," *Computers & Structures*, vol. 122, pp. 113–127, 2013.

[34] I. Martini, B. Haasdonk, and G. Rozza, "Certified reduced basis approximation for the coupling of viscous and inviscid parametrized flow models," *Journal of Scientific Computing*, vol. 74, pp. 197–219, 2018.

[35] I. Martini, G. Rozza, and B. Haasdonk, "Reduced basis approximation and a-posteriori error estimation for the coupled stokes-darcy system," *Advances in Computational Mathematics*, vol. 41, pp. 1131–1157, 2015.

[36] N. Discacciati and J. S. Hesthaven, "Localized model order reduction and domain decomposition methods for coupled heterogeneous systems," *International Journal for Numerical Methods in Engineering*, 2023.

[37] L. Iapichino, A. Quarteroni, and G. Rozza, "A reduced basis hybrid method for the coupling of parametrized domains represented by fluidic networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 221, pp. 63–82, 2012.

[38] M. S. Alnaes, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells, "The FEniCS project version 1.5," *Archive of Numerical Software*, vol. 3, 2015.

[39] A. Logg, K. Mardal, G. N. Wells, *et al.*, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[40] A. Toselli and O. Widlund, *Domain decomposition methods-algorithms and theory*, vol. 34. Springer Science & Business Media, 2004.

[41] A. Quarteroni and S. Quarteroni, *Numerical models for differential problems*, vol. 2. Springer, 2009.

[42] A. Quarteroni, A. Manzoni, and F. Negri, *Reduced basis methods for partial differential equations: an introduction*, vol. 92. Springer, 2015.

[43] S. T. Tokdar and R. E. Kass, "Importance sampling: a review," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 1, pp. 54–60, 2010.

[44] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[45] M. Al-Baali, E. Spedicato, and F. Maggioni, "Broyden's quasi-newton methods for a nonlinear system of equations and unconstrained optimization: a review and open problems," *Optimization Methods and Software*, vol. 29, no. 5, pp. 937–954, 2014.

[46] S. Amat, S. Busquier, and S. Plaza, "Review of some iterative root-finding methods from a dynamical point of view," *Scientia*, vol. 10, no. 3, p. 35, 2004.

[47] J. Zhang, "Modern monte carlo methods for efficient uncertainty quantification and propagation: A survey," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 13, no. 5, p. e1539, 2021.

[48] M. Haase, *Functional analysis: an elementary introduction*, vol. 156. American Mathematical Society Providence, RI, USA, 2014.

[49] É. Pardoux and Y. Veretennikov, "On the poisson equation and diffusion approximation. i," *The Annals of Probability*, vol. 29, no. 3, pp. 1061–1085, 2001.

[50] A. C. Newell and J. A. Whitehead, "Finite bandwidth, finite amplitude convection," *Journal of Fluid Mechanics*, vol. 38, no. 2, pp. 279–303, 1969.

# A Implementation

Implementation of the numerical results can be found on Github: `https://github.com/PaulStuiver/FP`.

# B Loss curves numerical Experiments part 1: linear-linear coupled Poisson problem

We report the loss curves of the neural networks that were trained for the linear-linear coupled Poisson problem. The learning rate is given by $l$.

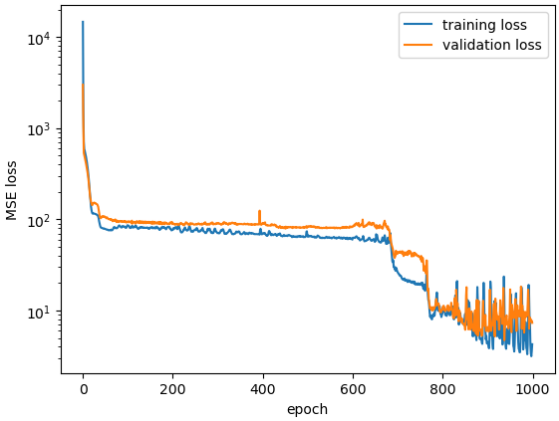



FIGURE B.1: Loss curve of $\hat{\pi}_2^{(1)}$ with $l = 0.03$ for hybrid-LL and surrogate-LL.

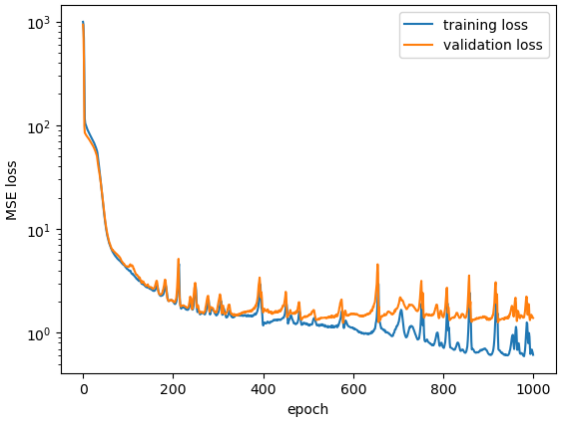



FIGURE B.2: Loss curve of $\hat{\pi}_2^{(2)}$ with $l = 0.003$ for hybrid-LL and surrogate-LL.

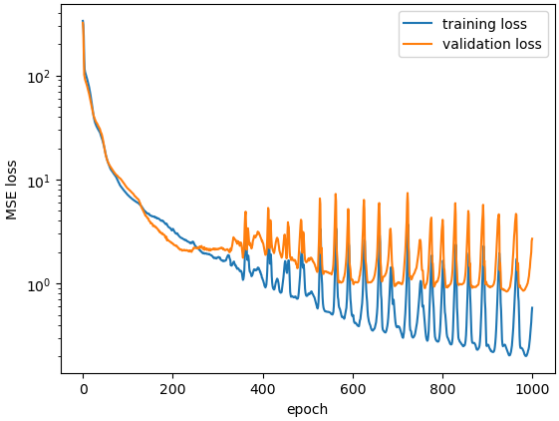



FIGURE B.3: Loss curve of $\hat{\pi}_2^{(3)}$ with $l = 0.003$ for hybrid-LL and surrogate-LL.

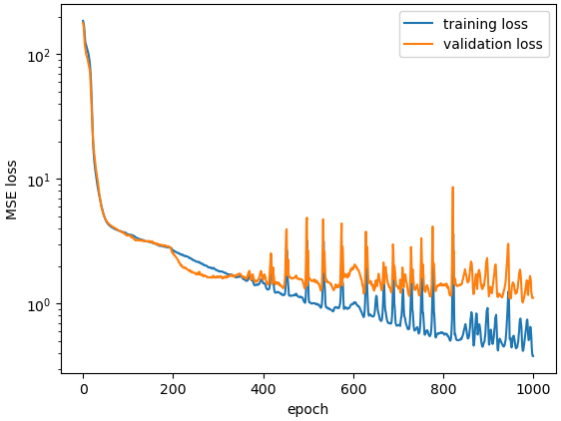



FIGURE B.4: Loss curve of $\hat{\pi}_2^{(4)}$ with $l = 0.003$ for hybrid-LL and surrogate-LL.

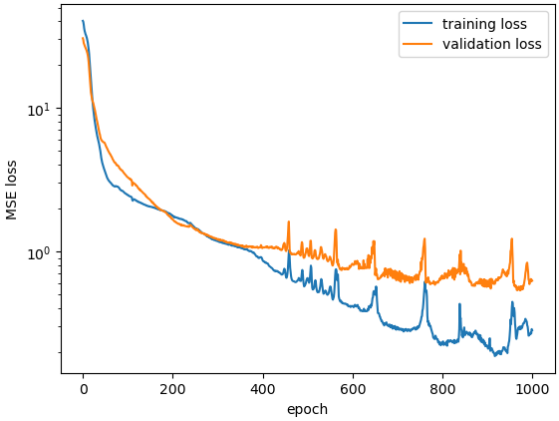



FIGURE B.5: Loss curve of $\hat{\pi}_2^{(5)}$ with $l = 0.003$ for hybrid-LL and surrogate-LL.

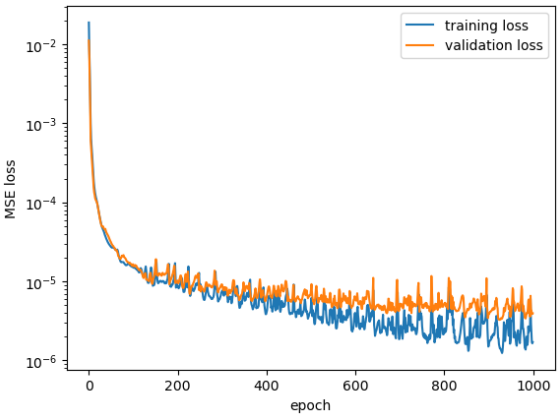



FIGURE B.6: Loss curve of $\hat{t}_\Gamma$ with $l = 0.0005$ for hybrid-LL and surrogate-LL.

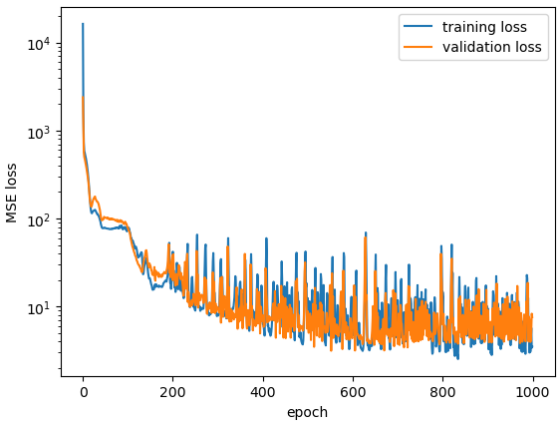



FIGURE B.7: Loss curve of $\hat{\pi}_2^{(1)}$ with $l = 0.03$ for hybrid-NL and surrogate-NL.

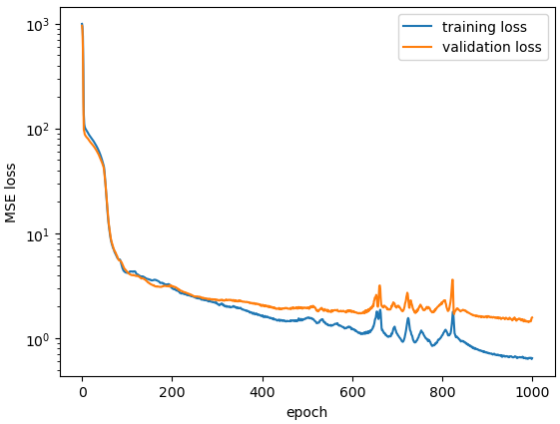



FIGURE B.8: Loss curve of $\hat{\pi}_2^{(2)}$ with $l = 0.003$ for hybrid-NL and surrogate-NL.

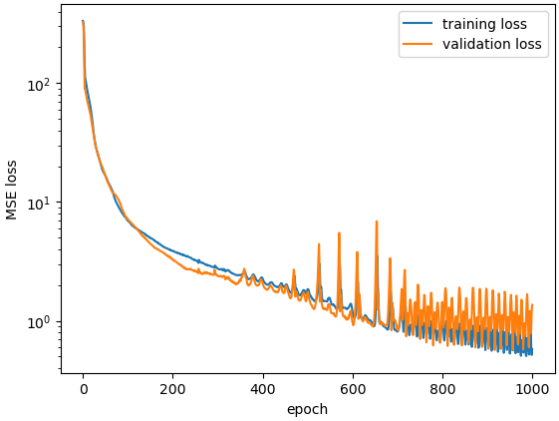



FIGURE B.9: Loss curve of $\hat{\pi}_2^{(3)}$ with $l = 0.003$ for hybrid-NL and surrogate-NL.

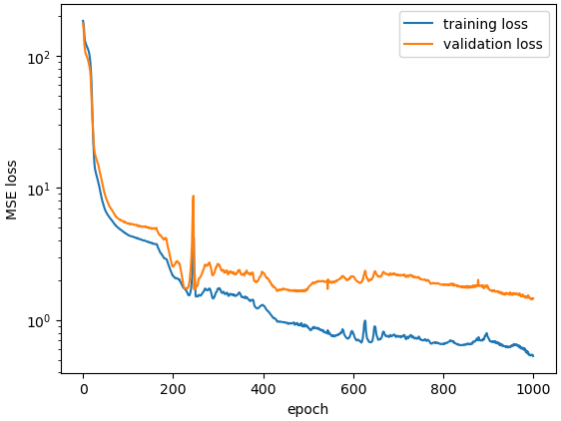



FIGURE B.10: Loss curve of $\hat{\pi}_2^{(4)}$ with $l = 0.003$ for hybrid-NL and surrogate-NL.

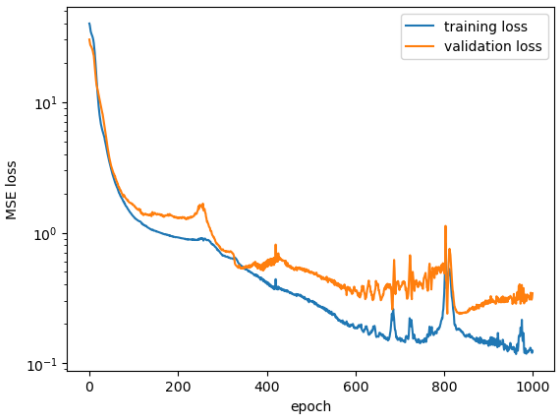



FIGURE B.11: Loss curve of $\hat{\pi}_2^{(5)}$ with $l = 0.003$ for hybrid-NL and surrogate-NL.

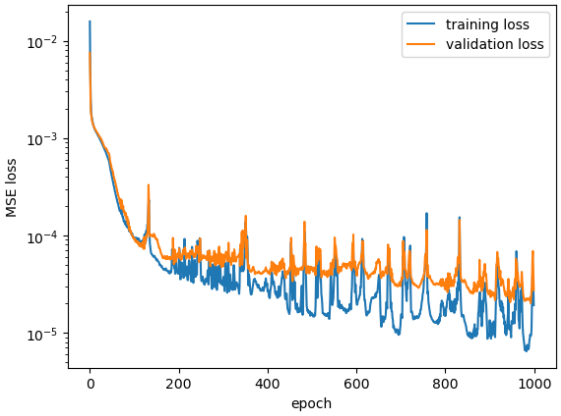



FIGURE B.12: Loss curve of $\hat{t}_\Gamma$ with $l = 0.0005$ for hybrid-NL and surrogate-NL.

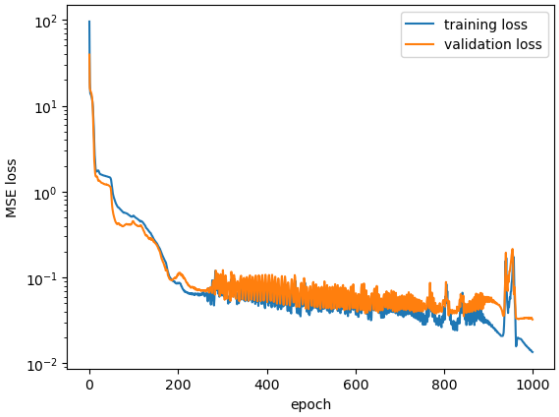



FIGURE B.13: Loss curve of $\hat{\pi}_2^{(5)}$ with $l = 0.001$ for hybrid-LL and surrogate-LL.

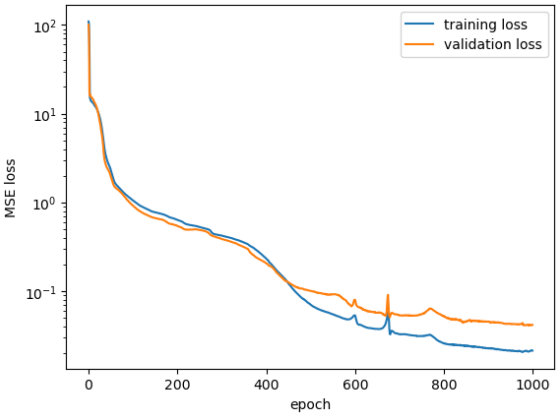



FIGURE B.14: Loss curve of $\hat{t}_\Gamma$ with $l = 0.001$ for hybrid-NL and surrogate-NL.