Investigating parametric flexibility in Reduced Order Models to speed up ASML's design process

F.R.C. Kaal

Delft University of Technology

TU Delft

Investigating parametric flexibility in Reduced Order Models to speed up ASML's design process

For the degree of Master of Science in Civil Engineering and in Construction Management and Engineering.

Name:	Falco Rinse Christian Kaal	
Student number:	5173272	
Course code:	CIEM0500 & CME5200	
Project duration:	November 11, 2024 – August 18, 2025	
Faculty:	Faculty of Civil Engineering and Geosciences, Delft	
Thesis committee:	Dr. Ir. P.C.J. Hoogenboom,	TU Delft, supervisor
	Dr. Ir. E. Papadonikolaki	TU Delft, supervisor
	Dr. Ir. J.S. Hoving,	TU Delft, chair
	Ir. F.W.F. Colin,	ASML, supervisor
	Ir. P.L. van der Meer,	ASML, supervisor

Cover: Assembly of one of the substructures of a High-NA EUV machine in the lab at ASML's headquarters in Veldhoven.

TU Delft

Preface

This thesis marks the final step in completing the master's degrees in Civil Engineering and Construction Management and Engineering. Thanks to the unique opportunity to combine both disciplines within a single graduation project, I was able to approach this research from both a technical-content perspective and a process- and implementation-oriented angle. This interaction between content and application has strengthened the study and made it more relevant for practical use. By completing this thesis, I also bring my six years of studying at TU Delft to an end. It has been a period I look back on with great joy. I have gained countless valuable memories and experiences during my studies here, as well as in everyday life here in this beautiful city.

I was fortunate to carry out this project at the fantastic company ASML. During my time here I received outstanding daily support from Flip Colin and Pieter van der Meer. They were always approachable, and their sharp insights and genuine involvement helped me move forward whenever I had questions. I am truly grateful for their guidance. I would also like to thank my other colleagues and fellow interns at ASML. Everyone is always willing to help, and the passion and expertise of the people here are truly admirable.

I would also like to thank my supervisors from TU Delft. I am grateful to Eleni Papadonikolaki for her always sharp insights from a Construction Management and Engineering perspective, which helped me broaden my view as an engineer. I would also like to thank Pierre Hoogenboom for his valuable input from the Civil Engineering side. I really appreciated being able to walk into your office and discuss my research. On top of that, I would like to thank Jeroen Hoving for being there to help me with issues related to the structure and organization of the research. Your insights were incredibly valuable.

Finally, I would like to thank my dear family and friends for always being there and for their continuous support throughout this process.

F.R.C. Kaal
Delft, August 2025

Executive summary

ASML designs extremely advanced lithography machines. When designing a new machine or one of its components, the behavior of the machine must be calculated and analyzed at system level to ensure that both static and dynamic requirements are met. Because meeting these requirements can be quite challenging, the design process often requires multiple iterations. Due to the size and complexity of these machines, the Finite Element Method (FEM) model is too large to solve in its entirety. Therefore, ASML uses a substructuring approach, in which multiple substructures are defined within the machine. For each of these, a Reduced Order Model (ROM) is created [1]. This is a simplified representation of a complex model, which significantly reduces its size [2]. To calculate the static and dynamic behavior at system level, the ROMs of the individual substructures are then connected.

Generating these ROMs is very time-consuming and they cannot be reused when parameters change. As a result, whenever a substructure is modified during one of the many iterations, a new ROM must be generated for that substructure. This significantly increases the overall duration of the design process. To address this issue, ASML is interested in methods that allow parameter changes to be applied directly within a ROM. Therefore, the aim of this research was to explore such techniques, assess how they could be integrated into ASML's design process, and evaluate the extent to which they can help to speed it up. The scope of this research was limited to techniques used for static analysis. In line with this goal and scope, the following main research question was introduced: '*To what extent can techniques that enable parametric changes within Reduced Order Models for static analysis contribute to speeding up ASML's design process?*'.

To answer the main research question, four sub-questions were formulated. The research followed the Design Science Research Methodology (DSRM), which provided a structured and iterative approach. The first sub-question was: '*What is the current state of the art in techniques that enable parametric changes within Reduced Order Models for static analysis?*'. Based on an extensive literature review, the following four techniques were identified that allow for direct or faster parametric changes in ROMs for static analysis: the Multiparameter Moment Matching Method (MMMM), the interpolation-based method Proper Orthogonal Decomposition Radial Basis Function (POD-RBF) approximation, Multilevel Substructuring (MLS), and the Woodbury Identity. For each of these techniques, their strengths, weaknesses, and aspects that remain unknown based on the current literature were outlined.

The second sub-question was: '*What does ASML's current design process look like, and what requirements must techniques meet to be implemented in this process?*'. To answer this question, a semi-structured interview was conducted with five employees who are familiar with the design process. It was found that typically around X to X design iterations are needed, with the generation of a ROM taking X to X each time (X inserted; not for public release). This clearly illustrates the potential for time savings, although it became clear that these savings would primarily apply to dynamic analysis, rather than static analysis. Based on the outlined current design process, an ideal design process with parametric flexibility within ROM was developed using the idealized design approach [3]. Using this ideal design process and the knowledge gained from the literature on the techniques and their strengths and weaknesses,

several potentially improved design processes were proposed in which these techniques are integrated. In this, the POD-RBF approximation was the only technique that skips not only the reduction step but also the step of regenerating the FEM model. Since many aspects from the literature were still unknown or unclear at this point, a set of implementation requirements was defined that must be met in order for these techniques to be successfully implemented.

The third sub-question asked: '*Do the existing techniques, either in their existing form or adapted, meet the requirements for implementation within ASML's design process?*'. To answer this question, two test structures were created. Using ANSYS, the stiffness matrices were generated, after which the various techniques, either as described in the literature or adapted, were applied in MATLAB. It was found that extracting the stiffness matrix from ANSYS is limited to a single processor core, while solving the full system using four cores was already faster than exporting the matrix with one core. Since one of the requirements was that only software already in use at ASML could be used, it was concluded that techniques which require retrieving new stiffness matrices for each parameter change do not meet the requirements. These include the MMMM, MLS, and the Woodbury Identity. Only the POD-RBF approximation remained, and this technique did meet all the requirements based on the tests with the simple structures. The only drawback found for this technique was that, in a substructuring context, accurate estimates can only be made when the coupling between the interface Degrees of Freedom (DOFs) is minimal. The tests for MLS also showed that generating a ROM for static analysis is already quite fast. This supported the earlier conclusion that, because this thesis focuses only on static analysis, the potential to significantly speed up the process is limited.

The fourth and final sub-question was: '*How do the techniques that meet the implementation requirements perform when applied to a designed ASML case study?*'. To answer this question, POD-RBF approximation was applied to the standardized factory floors on which the machines are placed. To obtain the interpolation data for this method, a parameterized ANSYS model of the factory floor was built. Subsequently, the POD-RBF approximation was again applied in MATLAB. While this technique had performed well on the previous simple test structures, the results now showed that on this structure it had difficulty achieving accurate interpolation. This is because the DOFs in the factory floor structure did not vary linearly with respect to each other when certain parameters were changed, making accurate interpolation challenging.

Having answered all sub-questions, it became possible to formulate an answer to the main research question. It was concluded that the POD-RBF approximation can indeed help speed up ASML's design process, although its constraints make the technique suitable only for scenarios where the coupling between interface DOFs is minimal and the DOFs in a structure respond fairly linearly in relation to one another. The reason why the POD-RBF approximation can speed up the design process in these specific scenarios is not, as initially expected, due to skipping the reduction step, but rather because it also skips the regeneration of the FEM model. The resulting time savings are therefore much smaller than the potential gains from skipping the reduction step for dynamic analysis, which, unlike in static analysis, takes a considerably long time. It is therefore recommended that future research focuses on this direction.

Table of contents

List of figures

List of tables

Nomenclature

List of abbreviations

DOFs	Degree(s) of Freedom
DSRM	Design Science Research Methodology
EB	Euler Bernoulli
FEM	Finite Element Method
MLS	Multilevel Substructuring
MMMM	Multiparameter Moment Matching Method
POD	Proper Orthogonal Decomposition
POD-RBF	Proper Orthogonal Decomposition Radial Basis Functions
PROM	Parametric Reduced Order Model
RBF	Radial Basis Functions
ROM	Reduced Order Model
SQ	Sub-Question

List of symbols

A	Cross-sectional area $[m^2]$
$\mathbf{A}$	Coefficient matrix in the system $\mathbf{Ax} = \mathbf{b}$
$\mathbf{b}$	Right-hand side vector in the system $\mathbf{Ax} = \mathbf{b}$
$\mathbf{B}$	RBF interpolation coefficient matrix
$\mathbf{C}$	Damping matrix
E	Modulus of elasticity $[N/m^2]$
$\mathbf{f}$	Force vector
F	Force $[N]$
$\mathbf{H}$	Hessenberg matrix
I	Second moment of area $[m^4]$
K	Cut-off value in POD-RBF approximation
$\mathbf{K}$	Stiffness matrix
L	Length $[m]$
$\mathbf{M}$	Mass matrix
Δp	Parameter change
$\mathbf{P}$	Parameter matrix in POD-RBF approximation
t	Thickness $[m]$
$\mathbf{T}$	Reduction matrix
$\mathbf{u}$	Displacement vector
$\mathbf{U}$	Displacement matrix in POD-RBF approximation
$\mathbf{v_i}$	Eigenvectors
$\mathbf{V}$	Projection matrix
$\mathbf{x}$	Solution vector in the system $\mathbf{Ax} = \mathbf{b}$
$\mathbf{Z}$	Krylov matrix
$\lambda_\mathbf{i}$	Eigenvalues
$\mathbf{\Phi}$	POD basis

1 Introduction

This chapter introduces the thesis. First, Section 1.1 provides background information about ASML, followed by Section 1.2, which defines the problem statement and outlines the scope. Section 1.3 presents the research questions, and finally, Section 1.4 describes the methodology and outline of the thesis.

1.1. Background

ASML is a Dutch technology company that develops lithography machines for semiconductor production. The machines produced by the company are used by chip manufacturers such as TSMC, Intel, and Samsung to create advanced chips for various electronic devices. There is a continuous global demand for more powerful and faster chips to support technological innovations such as artificial intelligence and 5G. ASML is continuously working on developing new machines capable of meeting these demands.

Once an initial concept of such a new machine or one of its components has been designed, its behavior must be calculated and analyzed. This is necessary to ensure that the machine meets predefined requirements, such as those related to the static and dynamic characteristics of the system. Since meeting these requirements is often very difficult, the process requires multiple iterations. Each iteration involves analyzing the results, making small adjustments to the design, and then recalculating and reanalyzing the model. These adjustments are kept small to ensure that the impact of each change can be properly understood.

Calculating the entire machine directly is not feasible due to the size of its Finite Element Method (FEM) model. ASML machines are highly complex, and as a result, their FEM models consist of tens of millions of Degrees of Freedom (DOFs). To make calculations on the full machine possible, ASML applies a technique called substructuring. In substructuring multiple substructures are defined within a structure, where for each of these a Reduced Order Model (ROM) is created [1]. A ROM is a simplified representation of a complex model that keeps its essential properties [2]. The ROMs of the different substructures are connected to calculate the static and dynamic behavior of the entire machine. Because the total size of all ROMs together is much smaller than that of the full model, these calculations can now be done relatively quickly.

1.2. Problem statement and scope

By using this substructuring approach, ASML encounters a problem. Generating the ROMs is computationally very expensive and therefore very time consuming. At the same time, ROMs cannot easily be reused when structural parameters change, even for small modifications. Therefore, in the many iterations of the design process, where small changes are made each time, a new ROM must be generated for any substructure in which a change occurs. This significantly contributes to the overall length of the design process.

This is why ASML is interested in a method that allows parameter changes to be applied directly within a ROM. The difference between the current workflow with ROMs and the desired approach is shown in Figure 1.1. The left diagram shows that in a ROM, the structural parameters are set before the reduction. In the desired approach, shown in the right diagram, it is still possible to make parametric changes after the model has been reduced. In this way, the time-consuming reduction step does not need to be repeated for every iteration, which could significantly speed up ASML's design process.

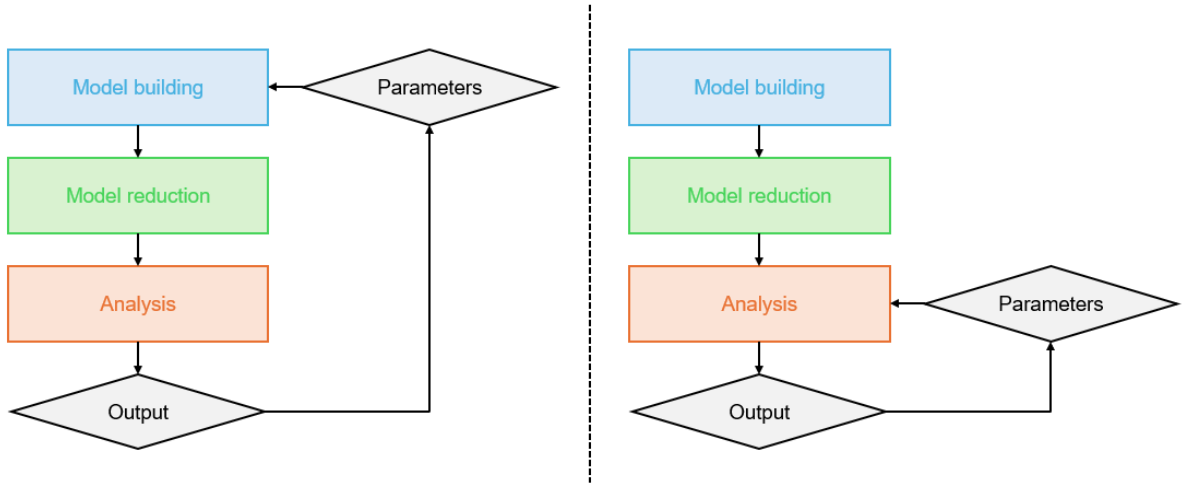

Figure 1.1: Current ROM approach vs. desired approach with parameter flexibility.

The goal of this research is to explore techniques that allow direct parametric changes in ROMs, assess how they could be used within ASML's design process, and evaluate the extent to which they can speed it up. The scope of this research is limited to techniques used for static analysis. These analyses are used to determine the displacements of structures under constant loads and form the basis for important design decisions within ASML. By defining this scope, the study allows for a more controlled evaluation of the methods. Although dynamic analyses fall outside the current scope their potential future application will also be briefly addressed.

1.3. Research questions

In line with the research goal and scope, the following main research question is introduced:

RQ: *To what extent can techniques that enable parametric changes within Reduced Order Models for static analysis contribute to speeding up ASML's design process?*

To answer this main research question, the following four sub-questions have been formulated:

SQ1: *What is the current state of the art in techniques that enable parametric changes within Reduced Order Models for static analysis?*

It is first important to obtain a complete overview of the current state of the art in techniques that allow parametric changes within ROMs. Only then it can be evaluated whether these techniques have the potential to speed up ASML's design process.

SQ2: *What does ASML's current design process look like, and what requirements must techniques meet to be implemented in this process?*

It is also needed to get a clear understanding of ASML's current design process. In addition, it should become clear which requirements techniques must meet in order to be useful within this process.

SQ3: *Do the existing techniques, either in their existing form or adapted, meet the requirements for implementation within ASML's design process?*

It is important to determine whether existing techniques meet the defined requirements. Additionally, it is relevant to assess whether certain adaptations or combinations of techniques would make them suitable.

SQ4: *How do the techniques that meet the implementation requirements perform when applied to a designed ASML case study?*

Finally, the techniques that meet the requirements, whether in their existing form or adapted, will be applied to a relevant structure within ASML. Their performance in terms of computation time and practical applicability will be evaluated to help answer the main research question.

1.4. Methodology and thesis outline

To answer the main research question, the defined sub-questions must be addressed. Although each sub-question is answered using a specific methodology, there is also an overall methodology that applies to the entire research, in particular the Design Science Research Methodology (DSRM). This overarching methodology is explained first, before the methodology for each sub-question is discussed.

Overarching methodology

The DSRM is a research approach focused on designing and developing artifacts, such as models, methods, and processes, that not only solve practical problems but also generate new scientific knowledge with broader applicability [4, 5]. This approach aligns well with the objectives of this research, as it facilitates the development of techniques that allow for direct changes in ROMs and an updated design process as artifacts, while generating new scientific insights into these techniques and their applicabilities.

The DSRM facilitates the artifact development process by providing a structured and repeatable approach [6]. Peffers [4] introduced the following six steps in the DSRM. These steps include iterative feedback, where insights from the evaluation in step 5 can lead to changes in the goals of the artifact in step 2 and the design of the artifact in step 3. This iterative nature is important because it allows the artifact to be improved continuously based on new knowledge [6]. Step 1 of the DSRM is discussed at the beginning of this chapter, and step 6 will take the form of this research report being published in the TU Delft repository, making the findings accessible to other researchers.

- ***Step 1*: Identify problem and motivate:** In this step the goal is to define the specific research problem and justify the value of a solution.
- ***Step 2*: Define objectives of a solution:** In this step the aim is to define clear objectives for a solution. These objectives should be based on the problem definition and

knowledge of what is possible and feasible.

- ***Step 3*: Design and development:** The artifact is designed and developed in this step.
- ***Step 4*: Demonstration:** This step involves demonstrating the use of the artifact by solving one or more instances of the problem.
- ***Step 5*: Evaluation:** The effectiveness of the artifact in solving the problem is observed and measured in this step.
- ***Step 6*: Communication:** This step involves communicating the problem, the developed artifact, and its effectiveness to researchers and other relevant audiences.

Sub-question-specific methodology

The steps of the DSRM support the research as a whole, but in the end it is a matter of answering the sub-questions in order to answer the main research question. That is why a specific methodology must be defined for each sub-question. Below, the methodology per sub-question is explained. Figure 1.2 gives a simple overview of the methods used for each sub-question, the structure of the thesis, and the DSRM steps followed in answering the different sub-questions.

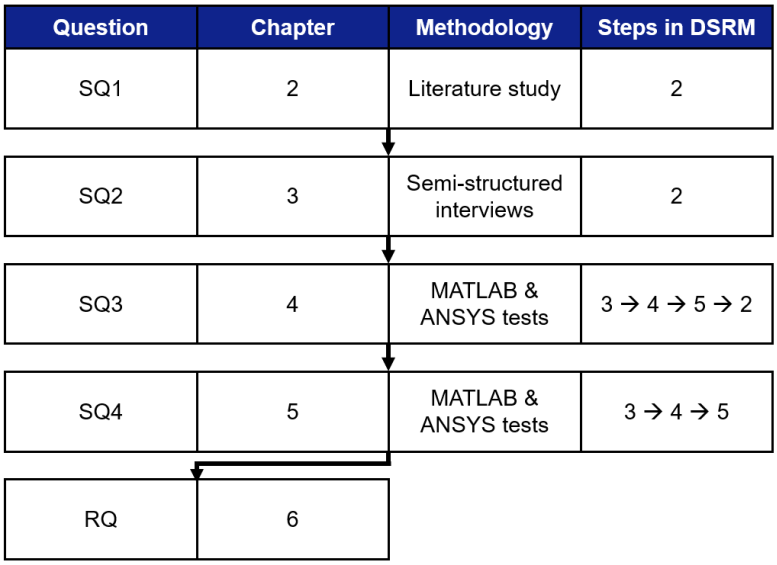

Figure 1.2: Overview of the methodology and the thesis outline.

To answer SQ1, a comprehensive literature study is conducted on existing techniques to allow for direct or faster changes in ROMs for static analyses. Scientific articles and relevant books are analyzed to gain an overview of the available methods and their theoretical foundation. Various academic databases, such as Google Scholar, ScienceDirect, and IEEE are used for this research. With theoretical knowledge about existing techniques, it becomes possible to define reasonable objectives, which is why step 2 of the DSRM is carried out. The literature study, the answer to SQ1, and the first definition of the objectives are presented in Chapter 2: '*The state of the art in techniques enabling changes in Reduced Order Models*'.

To answer SQ2 semi-structured interviews are conducted with ASML engineers to gain insight into the current design processes within ASML and the challenges encountered. These interviews helped to identify the requirements that techniques must meet in order to be successfully implemented within this process. Based on these findings, updated objectives are formulated,

corresponding to step 2 of the DSRM. The approach of the interviews, the defined requirements, as well as the updated objectives, can be found in Chapter 3: '*ASML's design process and definition of implementation requirements*'.

To evaluate whether the techniques meet the defined requirements and answer SQ3, it is important to gain a complete understanding of the different techniques. The literature alone does not provide enough clarity on whether all existing techniques satisfy these requirements. Therefore, extensive tests are carried out in Chapter 4: '*Verification of techniques against implementation requirements*', using ANSYS and MATLAB to provide more insight. ANSYS is used to set up the FEM model and generate the necessary matrices and vectors of the structures. These are then imported into MATLAB, where the techniques are applied. This chapter also explores possible adaptations and combinations of techniques that could meet the requirements. These are also tested using ANSYS and MATLAB. In this chapter, steps 3, 4, and 5 of the DSRM are followed. First, the techniques are developed and set up, then demonstrated on a test case, and finally evaluated to see if the requirements are met. At the end of this chapter, an iteration back to step 2 of the DSRM is made. With the insights gained, the final objectives can now be defined. These are presented at the end of Chapter 4.

Finally, a case study is carried out in which the techniques that meet the specified requirements are tested on a case study within ASML. The case study is a factory hall where ASML machines are installed. The performance of the techniques is compared to that of the original solution method. Once again, the necessary system matrices are generated in ANSYS, then imported into MATLAB, where the techniques are applied. This analysis provides concrete insights into the practical applicability of the techniques and the extent to which they help to speed up the design process. In this way, SQ4 can be answered. Developing the case study model, demonstrating its use, and evaluating the results correspond to steps 3, 4, and 5 of the DSRM. These steps, along with the answer to SQ4, can be found in Chapter 5: '*Case study on factory floor*'.

After all sub-questions have been answered, the main research question can be addressed. This can be found in Chapter 6: '*Conclusion and recommendations*'.

2

The state of the art in techniques enabling changes in Reduced Order Models

To answer SQ1: '*What is the current state of the art in techniques that enable parametric changes within Reduced Order Models for static analysis?*', a literature study is conducted in this chapter. Before diving into techniques that enable parametric changes in ROMs, it is important to first understand the basics of structural modeling and the ROM techniques themselves. These are discussed in Section 2.1 and Section 2.2, respectively.

After that, Section 2.3 covers Parametric Reduced Order Model (PROM) techniques that allow for parametric changes in ROMs. Techniques that also enable such changes, but are not actual PROMs, are discussed in Section 2.4.

Finally, Section 2.5 gives a clear answer to SQ1. In addition, this last section defines the first objectives for the solution, based on the insights gathered from the literature. With this, step 2 of the DSRM has been carried out. The whole chapter focuses on techniques suitable for static analyses, in line with the defined scope of this study.

2.1. Basics of structural modeling

To understand various ROM and PROM techniques and get definitions straight, some background information on structural modeling is given. This section sets out some basic information on finite element analysis and static and dynamic analysis.

Finite Element Method

The Finite Element Method (FEM) is a numerical method used to analyze complex structures. For undamped structures, the principle of FEM involves dividing a structure into smaller elements each with its own mass and stiffness. These elements are then combined, resulting in a global stiffness ($\mathbf{K}$) and mass matrix ($\mathbf{M}$) that simulates the behavior of the entire structure [7].

Depending on the complexity and geometry of the structure, different types of elements can be used. The most commonly used elements are outlined below:

- **Beam elements:** This type of element is used for slender structures such as beams and columns. Rotations, moments and forces in axial and shear directions can be modeled.

- **Shell elements:** For thin-walled structures such as plates, this type of element can be used. Membrane and bending stresses can be modeled. A common application of shell elements is in (thin) floors, sheet metal parts and flexure mechanisms.
- **Solid elements:** These are volumetric elements used for modeling complex three-dimensional geometries. Solid elements provide the highest level of detail but are also the most computationally intensive.

In Figure 2.1, the different types of elements are shown. Choosing the right type of element is important to find a balance between accuracy and computational efficiency. The choice also depends on the structure being analyzed. For instance, if a sheet metal part is modeled as a solid with at least two elements through its thickness, maintaining a good element aspect ratio would require an excessive number of elements. In such cases, it is more efficient to model the part as a shell element.

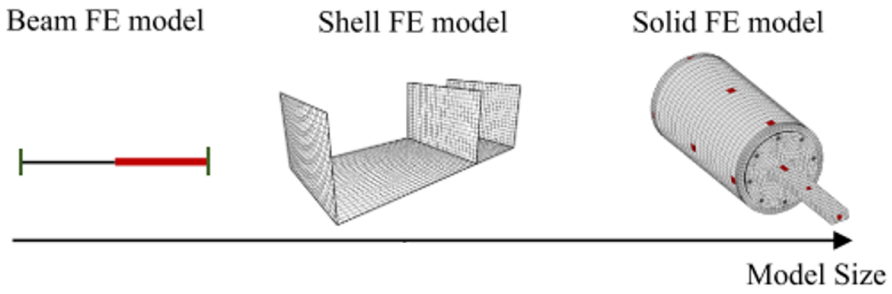

Figure 2.1: Different types of finite elements [8].

Static analysis

Once the structure is correctly modelled, an analysis can be performed to understand how the structure behaves under a certain load. In static analysis this load remains constant over time. The governing equation for static analysis is shown in Equation (2.1). It is important to note that, to ensure consistency throughout the report, fixed notations are used for scalars, vectors, and matrices. Scalars are denoted by non-bold letters, vectors by bold lowercase letters, and matrices by bold uppercase letters.

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{2.1}$$

As can be seen in the equation, only the global stiffness matrix $\mathbf{K}$ is included, and no mass matrix $\mathbf{M}$ or damping matrix $\mathbf{C}$. This is because accelerations $\ddot{\mathbf{u}}$ and velocities $\dot{\mathbf{u}}$ are zero in a static analysis. Using the stiffness matrix $\mathbf{K}$ and the force vector $\mathbf{f}$, the displacement vector $\mathbf{u}$ can be calculated with $\mathbf{u} = \mathbf{K}^{-1}\mathbf{f}$. Computing the inverse of the stiffness matrix $\mathbf{K}$ is a computationally intensive calculation for a system with a large number of DOFs.

Dynamic analysis

Dynamic analysis is used to investigate the behavior of structures under time-dependent loads. In dynamic analyses accelerations ($\ddot{\mathbf{u}}$) and velocities ($\dot{\mathbf{u}}$) must be considered, which introduces the mass matrix $\mathbf{M}$ and the damping matrix $\mathbf{C}$ into the governing equation. This is because the behavior of the structure depends on its inertia and energy dissipation [7]. As a result, dynamic analysis is significantly more complex than static analysis. The governing equation is shown in Equation (2.2).

$$\mathbf{M}\ddot{\mathbf{u}}(t) + \mathbf{C}\dot{\mathbf{u}}(t) + \mathbf{K}\mathbf{u}(t) = \mathbf{f}(t) \tag{2.2}$$

Since this research focuses on static analyses, dynamic analyses will not be explored in depth in this research.

2.2. Reduced Order Model techniques

Now that an overview of the basics of structural modeling has been provided, model reduction techniques can be explored. When analyzing complex systems ROMs play an important role. A ROM is a simplified version of a full model that retains the essential system behavior while significantly reducing the required computation time [2]. This allows a complex FEM model with many DOFs to be analyzed with a lower computational load.

It is important to mention that the given model reduction techniques can at least be applied in static analysis but may also be applicable in dynamic analysis. However, the explanation in this thesis will be limited to their application in static analysis.

2.2.1. Guyan reduction

The basis of a model reduction technique is finding a representation space of the solution that allows for obtaining a good approximation with a significantly lower computational cost compared to solving the full problem. One way to achieve this was proposed by Guyan [9], as described below. The representation space here is the reduction matrix $\mathbf{T}$. The reduction matrix maps the stiffness of the full system to a subset of DOFs of interest and thereby lowering the size of the problem significantly.

In Equation (2.3), a matrix equation is introduced that describes the static system as shown in Equation (2.1). In this matrix equation, the system is divided into a subset of eliminated DOFs and remaining DOFs, respectively $\mathbf{u}_1$ and $\mathbf{u}_2$, where no forces act on $\mathbf{u}_1$. The retained DOFs $\mathbf{u}_2$ and the eliminated DOFs $\mathbf{u}_1$ are also referred to as 'master DOFs' and 'slave DOFs'. The number of master DOFs i is chosen to be much smaller than the number of slave DOFs j. The size of the matrices can be expressed as $\mathbf{K}_{22} \in \mathbb{R}^{i \times i}$ and $\mathbf{K}_{11} \in \mathbb{R}^{j \times j}$.

$$\begin{bmatrix} \mathbf{K}_{22} & \mathbf{K}_{21} \\ \mathbf{K}_{12} & \mathbf{K}_{11} \end{bmatrix} \begin{bmatrix} \mathbf{u}_2 \\ \mathbf{u}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{f}_2 \\ \mathbf{0} \end{bmatrix} \tag{2.3}$$

This can be expressed as Equation (2.4) and Equation (2.5).

$$\mathbf{K}_{22}\mathbf{u}_2 + \mathbf{K}_{21}\mathbf{u}_1 = \mathbf{f}_2 \tag{2.4}$$

$$\mathbf{K}_{12}\mathbf{u}_2 + \mathbf{K}_{11}\mathbf{u}_1 = \mathbf{0} \tag{2.5}$$

Where $\mathbf{u}_1$ from (2.5) can be expressed as shown in (2.6).

$$\mathbf{u}_1 = -\mathbf{K}_{11}^{-1}\mathbf{K}_{12}\mathbf{u}_2 \tag{2.6}$$

The Guyan reduction matrix $\mathbf{T}_G$ can then be defined as shown in Equation (2.7).

$$\begin{bmatrix} \mathbf{u}_2 \\ \mathbf{u}_1 \end{bmatrix} = \mathbf{T}_G \mathbf{u}_2 \tag{2.7}$$

$$\text{where:} \quad \mathbf{T}_G = \begin{bmatrix} \mathbf{I} \\ -\mathbf{K}_{11}^{-1}\mathbf{K}_{12} \end{bmatrix}$$

Subsequently, the reduced stiffness matrix can be found using Equation (2.8). Substituted for Guyan reduction, it is shown in Equation (2.9).

$$\overline{\mathbf{K}} = \mathbf{T}^T \mathbf{K} \mathbf{T} \tag{2.8}$$

$$\overline{\mathbf{K}}_G = \mathbf{T}_G^T \mathbf{K} \mathbf{T}_G = \mathbf{K}_{22} - \mathbf{K}_{21}\mathbf{K}_{11}^{-1}\mathbf{K}_{12} \tag{2.9}$$

Using Guyan reduction, the system is now reduced to Equation (2.10). The full stiffness matrix $\mathbf{K} \in \mathbb{R}^{(i+j)\times(i+j)}$ is reduced in size to $\overline{\mathbf{K}}_G \in \mathbb{R}^{i\times i}$. If after determining $\mathbf{u}_2$ with this equation now also $\mathbf{u}_1$ needs to be determined, Equation (2.7) can be used.

$$\overline{\mathbf{K}}_G \mathbf{u}_2 = \mathbf{f}_2 \tag{2.10}$$

When Guyan reduction is applied to a static system the exact solution is obtained. However, when applied to dynamic systems the method only provides an approximation of the response.

2.2.2. Substructuring

A reduction method such as Guyan reduction can be effectively applied through substructuring. Substructuring allows a structure to be divided into smaller, manageable components that can be analyzed separately [1]. The method was introduced to address challenges in computational efficiency in projects where different teams work on separate parts of a structure [10]. According to Rixen [1], substructuring can be applied using the following procedure:

1. Define different subparts within the structure, known as substructures. These substructures can be parts of the model designed by different teams. In a rocket these could be the main jet, side jets, and the top, as shown in Figure 2.2. The stiffness matrices of the substructures are denoted as $\mathbf{K}^{(s)}$.
2. For each substructure, a reduction matrix $\mathbf{T}^{(s)}$ must be defined that at least retains the DOFs at the interface boundaries. This reduction matrix can be obtained using a ROM technique like Guyan reduction. When using Guyan reduction the interface DOFs are the master DOFs ($\mathbf{u}_2$), and the internal DOFs are the slave DOFs ($\mathbf{u}_1$). Each substructure should be reduced as shown in Equation (2.11). These ROMs can now easily be shared between different teams.

$$\overline{\mathbf{K}}^{(s)} = \mathbf{T}^{(s)T} \mathbf{K}^{(s)} \mathbf{T}^{(s)} \tag{2.11}$$

3. The final step is to reassemble the various ROMs into a complete structure. This is done by connecting the different reduced stiffness matrices $\overline{\mathbf{K}}^{(s)}$ via the interface DOFs into a single reduced stiffess matrix $\overline{\mathbf{K}}$.

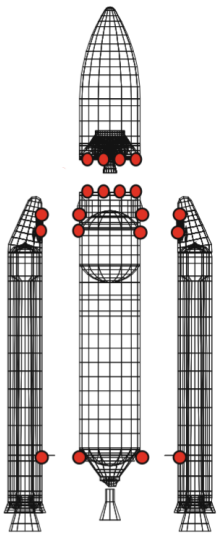

Figure 2.2: Substructured rocket where the red nodes are the interface nodes [1].

The method is very useful in a design situation where different teams work on various parts of a structure as each design change only affects the specific substructure. In this case, recalculations need to be performed only for the modified substructure. Another additional benefit is that parallel processing of the structure becomes possible by dividing it into substructures. [10]

The substructuring method is not limited to Guyan reduction. In dynamic analyses, it is much more common to use substructuring in combination with the Craig-Bampton reduction method, which is also referred to as Component Mode Synthesis. Since this thesis focuses only on static analyses, this explanation has been omitted. For more information on this topic, the reader is referred to [11] and [8].

2.2.3. Krylov subspace method

In addition to the Guyan reduction method, the Krylov subspace method can be used to determine a representation space for reducing the computational size of static analyses. For the Krylov subspace method, this is the projector matrix $\mathbf{V}$. The method is a indirect solution approach that generates a sequence of approximate solutions by solving a minimization problem over a particular type of subspace [12]. While a direct solution of Equation (2.1) requires determining $\mathbf{K}^{-1}$, this is avoided in the Krylov subspace method. Instead, the method uses matrix-vector products, which are computationally much more efficient, especially for large sparse matrices like stiffness matrices.

To define Krylov in its common notation, not Equation (2.1) is used, but $\mathbf{Ax} = \mathbf{b}$. The linear combinations of $\mathbf{b}, \mathbf{Ab}, \ldots, \mathbf{A}^{i-1}\mathbf{b}$ form the i^{th} Krylov subspace. Here is $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{b} \in \mathbb{R}^{n \times 1}$. This definition of the Krylov subspace is written in Equation (2.12). The Krylov matrix $\mathbf{Z}_i$ is introduced in Equation (2.13).

$$\mathbb{K}_i(\mathbf{A}, \mathbf{b}) = \text{span}\{\mathbf{b}, \mathbf{Ab}, \ldots, \mathbf{A}^{i-1}\mathbf{b}\} \tag{2.12}$$

$$\mathbf{Z}_i = \begin{bmatrix} \mathbf{b} & \mathbf{Ab} & \mathbf{A}^2\mathbf{b} & \ldots & \mathbf{A}^{i-1}\mathbf{b} \end{bmatrix} \tag{2.13}$$

The vectors from the Krylov matrix $\mathbf{Z}_i$ cannot be directly used as a basis for the subspace projector matrix $\mathbf{V}$. In many cases, these vectors become linearly dependent due to numerical rounding errors. The vectors are also not orthogonal, which is problematic when creating a projection. The Lanczos and Arnoldi algorithms solve these issues and generate the projector matrix $\mathbf{V}$. In this thesis, only the Arnoldi algorithm will be explained because it is used for the PROM technique Multiparameter Moment Matching Method, which is explained later in Subsection 2.3.1. For the Lanczos algorithm the reader is referred to [12].

Arnoldi algorithm

The fundamental idea of the Arnoldi algorithm is to remove all components parallel to the previous bases to ensure orthogonality. It is an iterative process in which $\mathbf{A}$ is reduced with the projector matrix $\mathbf{V}$ to the matrix $\mathbf{H}$ in upper-Hessenberg form, as shown in Equation (2.14). A matrix is written in upper-Hessenberg form if $a_{ij} = 0$ for all $i > j + 1$. The projector matrix $\mathbf{V}$ is an orthogonal basis consisting of the columns $\mathbf{v}_1, \ldots, \mathbf{v}_m$. The number of columns m influences the accuracy of $\mathbf{V}$. The more columns, the more accurate the projection, but this comes at the cost of the reduction size.

$$\mathbf{V}^T\mathbf{A}\mathbf{V} = \mathbf{H} \tag{2.14}$$

The Arnoldi algorithm uses a modified Gram-Schmidt process to produce the orthogonal vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$. The algorithm is shown in Appendix A.1.

The projector matrix has a size of $\mathbf{V} \in \mathbb{R}^{n\times m}$, which reduces $\mathbf{A} \in \mathbb{R}^{n\times n}$ to $\mathbf{H} \in \mathbb{R}^{m\times m}$, as shown in Equation (2.14). By choosing $m << n$, $\mathbf{A}$ is significantly reduced. In the end the system in Equation (2.15) can be solved to find $\mathbf{x}$. This is computationally much more efficient than directly solving $\mathbf{A}\mathbf{x} = \mathbf{b}$, especially when n is large.

$$\mathbf{H}\mathbf{x}_r = \mathbf{V}^T\mathbf{b} \tag{2.15}$$

$$\text{where:} \quad \mathbf{x} = \mathbf{V}\mathbf{x}_r$$

2.3. Parametric Reduced Order Model techniques

In the previous section, general ROM techniques for static analysis were introduced. With these techniques the reduced model is no longer valid if structural parameters change as the reduced system is built based on these specific system properties. Therefore, when parameters change the reduction procedure must be repeated. To avoid this repetition methods have been developed that retain the parametric effects after the reduction. These methods are referred to as Parametric Reduced Order Modeling (PROM) techniques [13]. Different PROM methods are introduced in this section. At the end of the explanation of each PROM technique suited for static analysis, its strengths, weaknesses, and unknowns are listed based on insights from the literature.

2.3.1. Multiparameter Moment Matching Method (MMMM)

In general, PROM techniques can be divided into *common projector methods* and *interpolation based methods*. Common projector methods are based on finding a reduction matrix that captures the model behaviour for a certain range of parameter values. The Multiparameter Moment Matching Method (MMMM) is an example of a common projector method. In interpolation methods the system for a specific parameter set is reduced individually, and an estimation

for a parameter within this set is made by interpolation. More about this will be discussed in Subsection 2.3.2.

The MMMM was originally developed by Daniel [14]. The method finds a common projection space which contains parametric effects as well as the characteristics of the original system. The developed method was applied to electro-thermal systems and electro-mechanical systems, but did not work well for structures. The most prominent reason for this was that the method generated singular matrices in structural applications. Yoo [18] resolved these issues by introducing improvements to the MMMM, including the use of enhanced Krylov subspace projections and adjustments to the Gram-Schmidt process. This eliminated the generation of singular matrices and made the method applicable to structural systems. In this subsection, the original MMMM, as proposed by Daniel, is first explained in the context of a static system. Subsequently, the modifications made by Yoo to improve the method are discussed.

The governing equation of a static system is given in Equation (2.1). If the stiffness matrix $\mathbf{K}$ depends on parameters $p_1, p_2, \ldots, p_{np}$, Equation (2.1) can be written as Equation (2.16). This equation can be rewritten as a Taylor series expansion, as shown in Equation (2.17).

$$\mathbf{K}(p_1, p_2, \ldots, p_{np})\mathbf{u} = \mathbf{f} \tag{2.16}$$

$$\mathbf{K}(p_1, p_2, \ldots, p_{np}) = \widetilde{\mathbf{K}}_0 + \sum_i \Delta\tilde{p}_i \widetilde{\mathbf{K}}_i + \sum_{j,k} \Delta\tilde{p}_j \Delta\tilde{p}_k \widetilde{\mathbf{K}}_{jk} + \sum_{l,m,n} \Delta\tilde{p}_l \Delta\tilde{p}_m \Delta\tilde{p}_n \widetilde{\mathbf{K}}_{lmn} + \ldots \tag{2.17}$$

$$\text{where:} \quad \widetilde{\mathbf{K}}_0 = \mathbf{K}(\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_{np}) = \mathbf{K}(\bar{\mathbf{p}}), \quad \widetilde{\mathbf{K}}_i = \left.\frac{\partial \mathbf{K}}{\partial p_i}\right|_{\bar{\mathbf{p}}}, \quad \widetilde{\mathbf{K}}_{ij} = \left.\frac{\partial^2 \mathbf{K}}{\partial p_i \partial p_j}\right|_{\bar{\mathbf{p}}},$$

$$\widetilde{\mathbf{K}}_{ijk} = \left.\frac{\partial^3 \mathbf{K}}{\partial p_i \partial p_j \partial p_k}\right|_{\bar{\mathbf{p}}}, \quad \Delta\tilde{p}_i = p_i - \bar{p}_i$$

Since the exact derivative is nearly impossible to find for large matrices, the first-order approximation is used, as shown in Equation (2.18).

$$\left.\frac{\partial \mathbf{K}}{\partial p_i}\right|_{\bar{\mathbf{p}}} \approx \frac{\Delta \mathbf{K}}{\Delta p_i} = \frac{\mathbf{K}(\bar{p}_{j,j\neq i}, \bar{p}_i + \Delta p_i) - \mathbf{K}(\bar{\mathbf{p}})}{(\bar{p}_i + \Delta p_i) - \bar{p}_i} \tag{2.18}$$

To simplify the formula, the notations have been reorganized as shown below. The static equation including parameters can ultimately be written as Equation (2.19).

$$\mathbf{K}_i = \begin{cases} \widetilde{\mathbf{K}}_i & i = 1, \ldots, np \\ \widetilde{\mathbf{K}}_{ij} & i = 1, \ldots, np, \quad j = 1, \ldots, np \\ \widetilde{\mathbf{K}}_{ijk} & i = 1, \ldots, np, \quad j = 1, \ldots, np, \quad k = 1, \ldots, np \\ \vdots & \end{cases}$$

$$\Delta p_i = \begin{cases} \Delta\tilde{p}_i & i = 1, \ldots, np \\ \Delta\tilde{p}_i \Delta\tilde{p}_j & i = 1, \ldots, np, \quad j = 1, \ldots, np \\ \Delta\tilde{p}_i \Delta\tilde{p}_j \Delta\tilde{p}_k & i = 1, \ldots, np, \quad j = 1, \ldots, np, \quad k = 1, \ldots, np \\ \vdots & \end{cases}$$

$$\mathbf{K}(p_1, p_2, \ldots, p_{np})\mathbf{u} = \left(\mathbf{K}_0 + \sum_{i=1}^{q} \Delta p_i \mathbf{K}_i \right) \mathbf{u} = \mathbf{f} \tag{2.19}$$

The displacement vector $\mathbf{u}$ can be expressed as shown in Equation (2.20).

$$\begin{aligned} \mathbf{u} = \mathbf{K}^{-1}\mathbf{f} &= \left(\mathbf{K}_0 + \sum_{i=1}^{q} \Delta p_i \mathbf{K}_i \right)^{-1} \mathbf{f} \\ &= \left(\mathbf{I} + \sum_{i=1}^{q} \Delta p_i \mathbf{K}_0^{-1} \mathbf{K}_i \right)^{-1} \mathbf{K}_0^{-1}\mathbf{f} \\ &= \left(\mathbf{I} - \sum_{i=1}^{q} \Delta p_i \mathbf{K}_i' \right)^{-1} \mathbf{f}_M \end{aligned} \tag{2.20}$$

$$\text{where:} \quad \mathbf{K}_i' = -\mathbf{K}_0^{-1}\mathbf{K}_i, \quad \mathbf{f}_M = \mathbf{K}_0^{-1}\mathbf{f}$$

By converting the inverse term into an infinite summation, Equation (2.21) is obtained.

$$\mathbf{u} = \sum_{i=0}^{\infty} \left(\Delta p_1 \mathbf{K}_1' + \Delta p_2 \mathbf{K}_2' + \cdots + \Delta p_q \mathbf{K}_q' \right)^i \mathbf{f}_M \tag{2.21}$$

Since the parameter changes Δp_i are scalar values, the space on which $\mathbf{u}$ lies can be expressed as shown in Equation (2.22). In the second equation this space written as the Krylov subspace.

$$\begin{aligned} \mathbf{u} &\in \text{colspan} \left\{ \sum_{i=0}^{\infty} \left(\mathbf{K}_1' + \mathbf{K}_2' + \cdots + \mathbf{K}_q' \right)^i \mathbf{f}_M \right\} \\ &\in \mathbb{K}\left(\mathbf{K}_1' + \mathbf{K}_2' + \cdots + \mathbf{K}_q', \mathbf{f}_M \right) \end{aligned} \tag{2.22}$$

The projector matrix $\mathbf{V}$ is a matrix composed of Krylov subspace vectors, as shown in Equation (2.23). It can be seen here that $\mathbf{V}$ depends on the initial load vector.

$$\mathbf{V} = \mathbb{K}\left(\mathbf{K}_1' + \mathbf{K}_2' + \cdots + \mathbf{K}_q', \mathbf{f}_M \right) \tag{2.23}$$

Once the projector matrix $\mathbf{V}$ is constructed for a specific set of parameters, it can be used to implement parameter changes in the reduced model, as shown in Equation (2.24).

$$\begin{aligned} \mathbf{K}_r(p)\mathbf{u}_r &= \mathbf{f}_r \\ \left(\mathbf{V}^T\mathbf{K}_0\mathbf{V} + \sum_{i=1}^{q} \Delta p_i \left(\mathbf{V}^T\mathbf{K}_i\mathbf{V}\right)\right)\mathbf{u}_r &= \mathbf{f}_r \\ \left(\mathbf{K}_{0,r} + \sum_{i=1}^{q} \Delta p_i \mathbf{K}_{i,r}\right)\mathbf{u}_r &= \mathbf{f}_r \end{aligned} \tag{2.24}$$

$$\text{where:} \quad \mathbf{K}_{i,r} = \mathbf{V}^T\mathbf{K}_i\mathbf{V} \quad (i = 0, 1, \ldots, q), \quad \mathbf{V}\mathbf{u}_r = \mathbf{u}, \quad \mathbf{V}^T\mathbf{f} = \mathbf{f}_r$$

The MMMM is of great value because it ultimately results in a significant reduction in computation time. While a direct solution of u requires determining $\mathbf{K}^{-1}$, this is avoided in the MMMM. Instead, the method uses matrix-vector products, which are computationally much more efficient, especially for large sparse matrices like stiffness matrices. A simplified visualization of how this technique reduces the FEM calculations compared to direct calculation of u is shown in Figure 2.3.

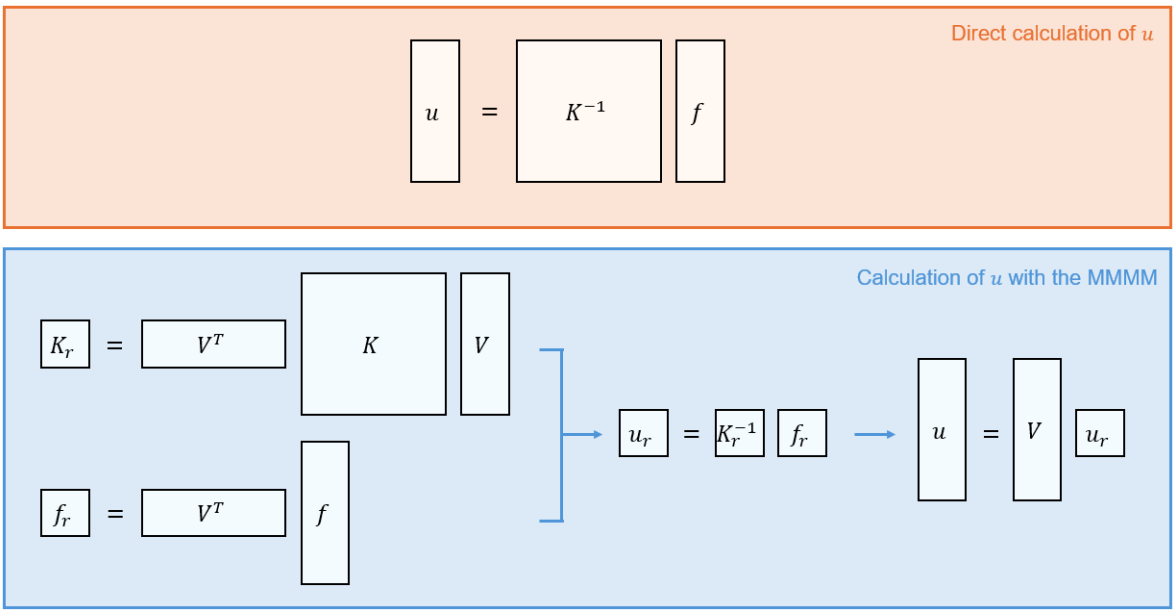

Figure 2.3: Simplified visualization of direct calculation of the displacement vector versus calculation using the MMMM.

Two issues arise when applying the MMMM developed by Daniel [14] to structural problems. First, the accuracy is highly dependent on the initial loading vector. If the given loading is close to the initial load vector used in the reduction process, the results match well with the original model. However, if the loads differ significantly the method becomes inaccurate. The second issue is that due to the high sparsity of the stiffness matrix, the projector matrix $\mathbf{V}$ often becomes a singular matrix. To address these two issues, Yoo [15] proposed two modifications.

Modification 1: two-step Arnoldi method

The two-step Arnoldi method solves the problem that arises when the load vector deviates significantly from the load vector in the reduction process, resulting in low accuracy. Since for

a particular initial load the reduced model is valid near that loading condition, Yoo [15] proposed the idea to collect various possible loads and compress them using the Gram-Schmidt method. After this, the block Krylov subspace method can be applied to the compressed input matrix. These two steps are outlined below in Equation (2.25).

$$\begin{aligned} &\text{Step 1:} \quad \mathbf{W} = \mathbb{K}(\mathbf{A}, \mathbf{B}) + \text{deflation}, \quad \text{where: } \mathbf{B} = \{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_p\} \\ &\text{Step 2:} \quad \mathbf{V} = \mathbb{K}(\mathbf{A}, \mathbf{W}) + \text{deflation}, \quad \text{where: } \mathbf{W} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p\} \end{aligned} \tag{2.25}$$

To construct $\mathbf{W}$ and $\mathbf{V}$ the Arnoldi algorithm is used (see Appendix A.1). The final MMMM algorithm is presented in Appendix A.2. After step 4, all columns $\mathbf{v}_j$ of $\mathbf{V}$ are obtained. This projector matrix can then be used for different loading conditions while maintaining accuracy. By applying deflation in steps 2 and 4 of this algorithm, the issue of obtaining a singular matrix $\mathbf{V}$ is also resolved. How this is achieved is described below.

Modification 2: deflation

With the modified Gram-Schmidt method, as used in the Arnoldi algorithm, orthogonality among the columns of the projector matrix $\mathbf{V}$ is often lost, causing the matrix to become singular. This is related to the sparsity of the stiffness matrix in an FEM model. According to the modified Gram-Schmidt method, if a newly generated vector does not contain any element orthogonal to one of the previous bases, it reverts to the previous bases. The resulting singular projector matrix $\mathbf{V}$ causes the reduced system to also become singular.

To prevent this, deflation is applied. This means that only the bases orthogonal to each other are selected after the Arnoldi algorithm. This is done by removing the columns that are near zero for $\mathbf{v}_i^T \mathbf{v}_j (i \neq j)$.

While this subsection has focused on the MMMM as a commonly used projection-based method, it is not the only method that follows this general approach. Other projection-based techniques exist, such as Component-Based PROM [16] and Next Generation PROM [17], but as these techniques are specifically applied to dynamic problems they have been omitted in this thesis. It should be noted that the MMMM also has an extension for dynamic problems. For this theoretical background the reader is referred to [15].

Strengths, weaknesses, and unknowns of MMMM

Below, the strengths, weaknesses, and unknowns of the MMMM are outlined. The strengths and weaknesses are based solely on the literature by Daniel [14] and Yoo [15]. The unknowns refer to aspects that could not be identified or clarified in the available literature and are therefore still considered unexplored.

Strengths

- **Supports parameter changes across the entire structure:** This method allows changes to be made to various parameters of the entire structure at once. It is therefore not limited to making changes in only a small part of the structure.
- **Supports variation in force location:** With the implementation of the 'Two-step Arnoldi Method' 1 by Yoo, it has become possible to vary the location of the applied force.

Weaknesses

- **Low accuracy for nonlinear parameter dependencies:** The MMMM starts as a common projector method based on the Taylor series approximation of parametric effects. As a result, the more the parameters deviate from their initial values, the larger the error becomes. Therefore, the accuracy of this method is low when the system depends nonlinearly on the parameters.
- **Requires many new stiffness matrices for multiple parameter changes:** As shown in Equation (2.24), when a system needs to be recalculated for new parameter values, new stiffness matrices $\mathbf{K_{i,r}}$ must be created. For a single parameter change, this means only one new stiffness matrix is needed. However, for two parameter changes, three stiffness matrices must be generated: one for each individual change and one for the combination of both. With more parameter changes, the number of required stiffness matrices increases even further. This can potentially reduce the efficiency of the method, especially if the new stiffness matrices cannot be generated quickly.

Unknowns

- **Unknown if the method works for local changes:** The literature only includes tests in which a parameter change is applied to the entire structure. It is therefore uncertain whether the method also works and remains effective for local changes in the structure.
- **Unknown which parameters can be accurately varied:** The literature shows that parameters which appear linearly in the stiffness matrix can be varied accurately, while nonlinear parameters are more difficult. But what about parameters that appear only partly nonlinearly in the stiffness matrix?
- **Unclear impact of projector $\mathbf{V}$ size on accuracy:** The MMMM seems to offer some flexibility in the accuracy of the PROM as the number of columns generated for the projector matrix $\mathbf{V}$ can be chosen (see line 1, step 3 in the MMMM algorithm in Appendix A.2). However, it is unknown to what extent more columns actually lead to higher accuracy.

2.3.2. Proper Orthogonal Decomposition Radial Basis Function approximation (POD-RBF approximation)

The counterpart of common projection methods is interpolation-based methods. Interpolation-based methods take place after the reduction has already been performed. The method works by utilizing a database of reduced models and interpolating between them to generate a new reduced model for the desired parameter set. This interpolation can be applied to the reduced system matrix, such as the stiffness matrix $\mathbf{K}$ used in static analysis, or to the reduction matrix $\mathbf{T}$ [18].

The simplest interpolation method is to directly interpolate between the different matrices. The directly interpolated stiffness matrix $\overline{\mathbf{K}}(p)$ for parameter p is defined by Lee [19] as shown in Equation (2.26). Here, n is the total number of reduced stiffness matrices to be interpolated between, $\omega_i(p)$ are the corresponding weight factors, and $\mathbf{K}_i$ are the reduced stiffness matrices themselves. The reduction of these stiffness matrices can, for instance, be performed using Guyan reduction as described in Subsection 2.2.1.

$$\overline{\mathbf{K}}(p) = \sum_{i=1}^{n} \omega_i(p)\mathbf{K}_i \tag{2.26}$$

$$\text{where: } \sum_{i=1}^{n} \omega_i(p) = 1$$

To determine the weight factors, the simplest approach is linear approximation. For linear interpolation between only two reduced stiffness matrices, the weight factors can be determined based on Figure 2.4.

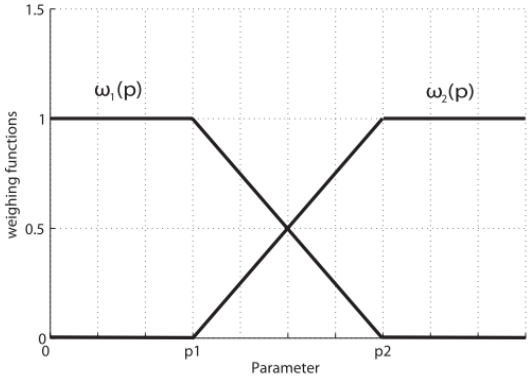

Figure 2.4: Shows the values of ω_1 and ω_2 for parameter p, for interpolation between 2 reduced models [20].

Amsallem [21] and Benner [22] state that directly interpolating between the stiffness matrix and reduction matrices $\mathbf{T}$ can affect the symmetric positive definiteness, resulting in a meaningless PROM. It is also crucial for these matrices to maintain orthogonality, which is not always the case.

Although this direct interpolation method provides a simple example, it is often not useful in practice. However, this direct interpolation method formed the basis for more complex interpolation methods, such as Proper Orthogonal Decomposition Radial Basis Function (POD-RBF) approximation. In this method, Proper Orthogonal Decomposition (POD) and Radial Basis Functions (RBF) are combined to generate a continuous approximation of a system over a certain parameter domain [23].

POD is used to reduce the original dataset to a limited number of orthogonal modes that describe the dominant characteristics of the system. Then, RBF is used to interpolate the coefficients of these modes over the parameter domain. Below, the training procedure for setting up POD-RBF approximation for static analyses, as introduced by Buljak [23], is explained.

Training procedure

The training procedure is shown in Figure 2.5. The first step is to define the input parameter set. The input parameter set depends on the number of parameters S that need to be varied and the number of chosen interpolation points per parameter. In total, there are M input parameter sets. For each input parameter set $\mathbf{p}_i$, the displacement vector $\mathbf{u}_i$ must be generated. These displacement vectors are collected in $\mathbf{U} \in \mathbb{R}^{N \times M}$, where N represents the number of DOFs.

The second step is to construct $\mathbf{D}$ or $\mathbf{C}$ using Equation (2.27). The construction that results in the smallest and thus most efficient matrix is chosen. From $\mathbf{D}$ or $\mathbf{C}$, the eigenvalues λ_i and eigenvectors $\mathbf{v}_i$ are extracted. This can be done using a built-in function in most programming languages. The eigenvalues λ_i are sorted in descending order as $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_M \geq 0$.

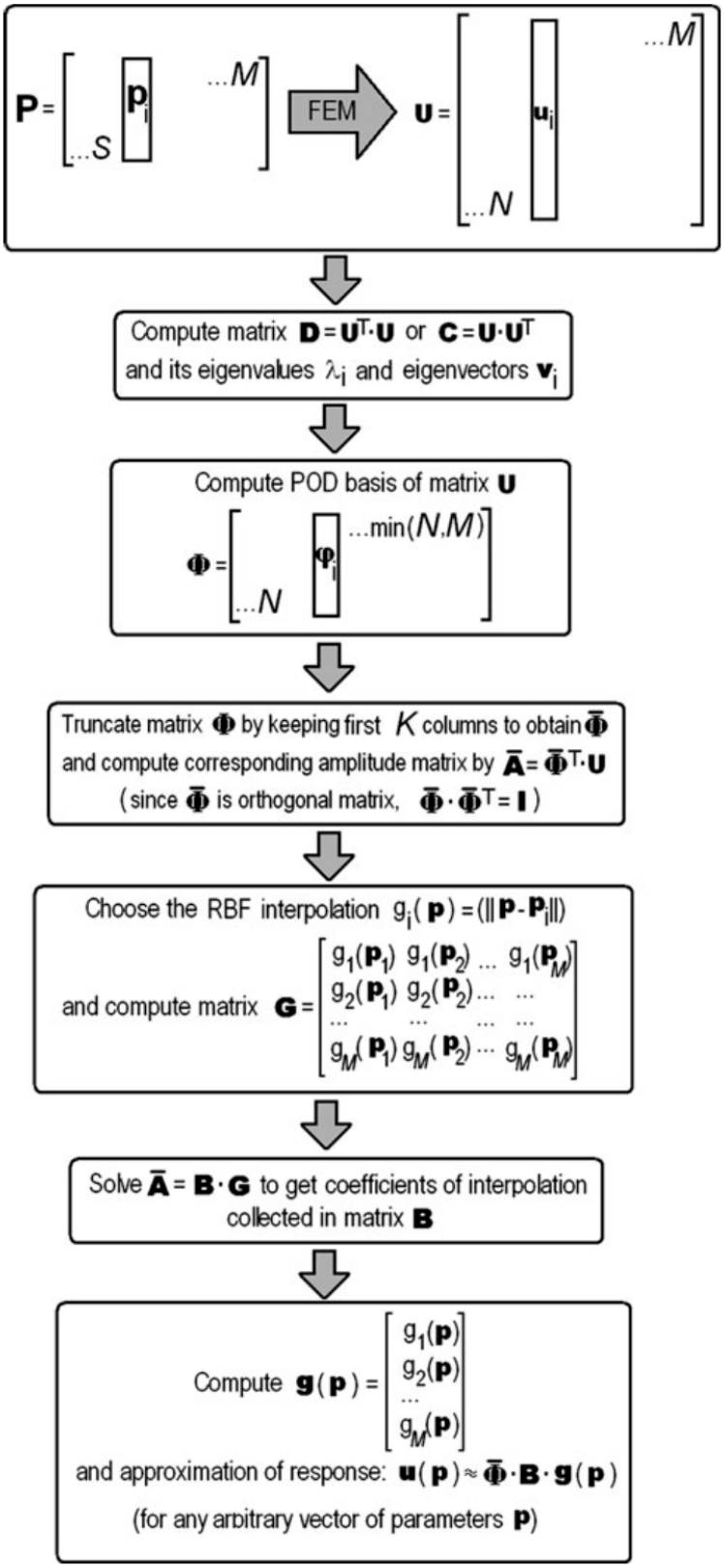

Figure 2.5: Flow chart of the steps needed in order to train and perform the POD-RBF approximation method [23].

$$\mathbf{D}, \mathbf{C} = \begin{cases} \mathbf{D} = \mathbf{U}^T\mathbf{U}, & \text{if } M < N \\ \mathbf{C} = \mathbf{U}\mathbf{U}^T, & \text{if } M \geq N \end{cases} \tag{2.27}$$

The third step is to construct the actual POD basis $\mathbf{\Phi}$, formed by the POD modes ϕ_i, from $\mathbf{U}$. The procedure to obtain the POD modes ϕ_i differs for $M < N$ and $M > N$, as shown in Equation (2.28). In theory, there are $\min(M, N)$ POD modes, but in practice, truncation is applied up to cut-off value K modes, where $K \ll M$ or $K \ll N$, to reduce the model. The POD basis containing only the first K modes is $\overline{\mathbf{\Phi}} \in \mathbb{R}^{N \times K}$. The final step within the POD process in the POD-RBF training procedure is to obtain the corresponding amplitude matrix $\overline{\mathbf{A}}$ using Equation (2.29).

$$\phi_i = \begin{cases} \frac{1}{\sqrt{\lambda_i}}\mathbf{U}\mathbf{v}_i, & \text{if } M < N \\ \mathbf{v}_i, & \text{if } M \geq N \end{cases} \tag{2.28}$$

$$\overline{\mathbf{A}} = \overline{\mathbf{\Phi}}^T\mathbf{U} \tag{2.29}$$

Now, the RBF interpolation can begin. This method operates using a set of basis functions applied to the different input parameter sets $\mathbf{p}_i$. A typical form of such a basis function is shown in Equation (2.30).

Other basis functions can also be used, such as *Gaussian* or *Thin Plate Spline*. For these definitions, refer to [23]. Together, the functions g_i form the matrix $\mathbf{G}$. Using Equation (2.31), the RBF interpolation coefficient matrix $\mathbf{B}$ is then determined.

$$g_i(\mathbf{p}) = \|\mathbf{p} - \mathbf{p}_i\| \tag{2.30}$$

$$\mathbf{B} = \mathbf{G}^{-1}\overline{\mathbf{A}} \tag{2.31}$$

The training procedure is now essentially complete. Now, for the parameter set $\mathbf{p}$ for which the displacement needs to be retrieved, $\mathbf{g}(\mathbf{p})$ can be constructed. After this the POD-RBF approximation of the displacements can be made using Equation (2.32).

$$\mathbf{u}(\mathbf{p}) = \overline{\mathbf{\Phi}}\mathbf{B}\mathbf{g}(\mathbf{p}) \tag{2.32}$$

What stands out about this method is that for each new parameter set where $\mathbf{u}(\mathbf{p})$ needs to be determined, only $\mathbf{g}(\mathbf{p})$ has to be calculated. In contrast to the MMMM, where a new stiffness matrix must be created for each parameter set, this appears to be a relatively fast step.

POD-RBF approximation is not the only interpolation technique. There is also Manifold Tangent Plane Interpolation. This technique projects the matrices onto a tangent plane, where interpolation is performed, and then re-projects them back. However, this method is more focused on dynamics and is therefore not included in this study. For a detailed description and implementation of this method, refer to [18], [24], and [25].

Strengths, weaknesses, and unknowns of POD-RBF approximation

Below, the strengths, weaknesses, and unknowns of POD-RBF approximation are outlined.

Strengths

- **Supports global and local parameter changes:** Because this is an interpolation method, it is possible to interpolate for modified parameters both in the entire system and in local parts of the structure, as long as these cases are included in the training data.
- **Handles nonlinear parameter dependencies:** One of the strengths of POD-RBF approximation is that, unlike the MMMM, it allows for fairly accurate estimations of parameters with non-linear dependencies as long as there are enough interpolation points.
- **No stiffness matrices required after training:** Once the training procedure is completed, unlike the MMMM, no new stiffness matrices are needed for changes in parameters. This makes it a very fast method. This also means that no software license is required to generate new stiffness matrices when using the PROM. The importance of this will become clear later in this report.

Weaknesses

- **Dimensionality issues with many parameters:** As the number of parameters increases, the number of required interpolation points grows exponentially. This means the number of reduced models that must be generated and stored can become unmanageable. This problem becomes worse when high accuracy is needed.
- **Requires prior knowledge of parameter changes:** The location and the parameter to be changed in the structure must be known beforehand as a change in a specific parameter and location is not possible if this case was not included in the training data.

Unknowns

- **Unknown performance for varying force locations:** The literature only discusses structural changes and not variations in force location. It is therefore unclear how the method performs in this case.

2.4. Additional techniques

In the previous subsection, PROM techniques were discussed. A typical characteristic of a PROM technique is that after the reduction, parametric effects are still retained. The techniques explained below do not retain parametric effects after the reduction, but they do offer the potential to apply parametric changes in reduced models in a fast way.

2.4.1. Multilevel Substructuring (MLS)

One of the first researchers to investigate the sensitivity of ROMs to parameter changes was Balmes [26]. One of the methods he proposed was to increase the size of the reduced model. By this, he meant selecting more master DOFs in a Guyan or Craig-Bampton reduction to improve accuracy when parameters change. Although his research concluded that adding more master DOFs was not effective when parameters varied, it sparked the idea of including more interface DOFs within the substructuring method. By adding more interface DOFs, more substructures are essentially formed within the model. As explained in Subsection 2.2.2, when a change occurs within a substructure, only that specific substructure needs to be reduced again rather than the entire model.

By dividing a system into more and more substructures, adjustments can be made within each substructure individually, allowing parameter changes in that specific subpart of the total system to be executed more quickly. Substructuring remains not a true PROM method, as the

reduction procedure still needs to be repeated for every single substructure. However, due to the potential for significant computational gains, it is included in this chapter.

One of the challenges of dividing a system into many substructures is that a significant number of interface DOFs also arise. This results in a system that still has a substantial number of DOFs, requiring considerable computational effort. Multilevel Substructuring (MLS) addresses this issue. Avhinav et al. [27] propose that with this method, it is not necessary to include a large number of interfaces in the static analysis step by further subdividing each substructure into multiple substructures.

With this method the system is essentially divided into multiple levels: level 0 represents the entire system, level 1 is a division of the system into multiple substructures, and level 2 is where each substructure is again subdivided into multiple substructures. Figure 2.6 illustrates this clearly. It can be seen that the substructures follow a tree topology. This process can continue until the substructures are small enough.

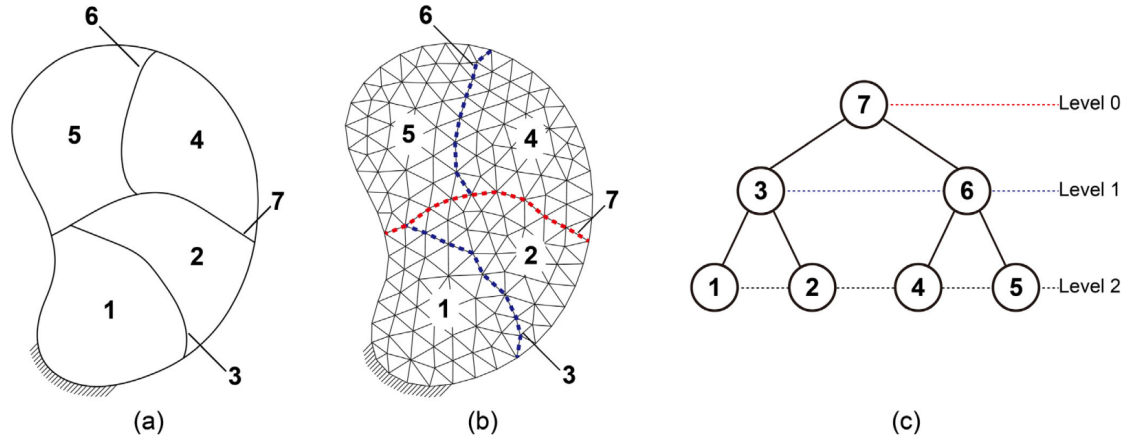

Figure 2.6: Two-level substructuring of generic FEM model: a) partitioned structure, b) partitioned FE model, c) substructure tree [28].

The stiffness matrix of each substructure (not including the lowest level) can be assembled by connecting the reduced stiffness matrices of the underlying substructures through the interface DOFs. Once all substructures at a certain level have been assembled, the process can continue to the level above. Only a small portion of the interface DOFs at one level are also interface DOFs at the level above. The remaining DOFs become internal DOFs and can now be eliminated.

As a result, only the interface DOFs of level 1 remain, effectively reducing the large number of interface DOFs at the lowest level, which improves the computational effort. A method to automate the MLS process, called *Automated Multilevel Substructuring*, was introduced by Bennighof et al. [29]. For further explanation the reader is referred to their article.

Strengths and weaknesses of multilevel substructuring

Multilevel substructuring and regular substructuring essentially offer the same strengths and weaknesses. The only difference is that MLS applies the method in multiple levels. The strengths and weaknesses listed below therefore apply to both multilevel and regular substructuring. Since substructuring is a widely researched topic, no direct unknowns remain, in contrast to the MMMM and the POD-RBF approximation method.

Strengths

- **Provides exact solutions:** When substructuring is used with Guyan reduction, exact solutions are obtained.
- **No need to know the location of parameter changes in advance:** Basically, a change can be made in any substructure. Therefore, no prior knowledge is needed about where changes might be made later, unlike with the POD-RBF approximation method where this must be known beforehand.
- **Supports variation in force location:** It is possible to change the force location. This is treated simply as a modification within a substructure.

Weaknesses

- **Not effective for changes in large parts of the structure:** The advantage of substructuring is that when a change is made in a single substructure, only that specific substructure needs to be regenerated. However, this advantage disappears when changes are required in a large part of the system. In such cases, every affected substructure must still be regenerated. Therefore, this method is not suitable for parameter changes that affect a large portion of the structure.

2.4.2. Woodbury Identity

The Woodbury Identity is a mathematical method to perform low-rank matrix updates in a computationally cheap way [12]. This method offers the potential to calculate the system $\mathbf{u} = \mathbf{K}^{-1}\mathbf{f}$ much faster when parametric changes occur in only a small part of the structure. This is because with this method, the inverse of the new stiffness matrix does not need to be computed directly. The Woodbury Identity is defined in Equation (2.33).

$$\left(\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{U}^{\mathrm{T}}\right)^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{U}\left(\mathbf{C}^{-1} + \mathbf{U}^{\mathrm{T}}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{U}^{\mathrm{T}}\mathbf{A}^{-1} \tag{2.33}$$

In the equation above, $\mathbf{A}$ is the stiffness matrix of the original structure that has been computed. Now, $\mathbf{A} + \mathbf{U}\mathbf{C}\mathbf{U}^{\mathrm{T}}$ is the stiffness matrix of the modified system, where the low-rank matrix change is described by $\mathbf{U}\mathbf{C}\mathbf{U}^{\mathrm{T}}$. Below, it is explained how to define $\mathbf{U}$ and $\mathbf{C}$, and how the new system can then be calculated in a computationally cheap way. The explanation is based on the research by Swart [12].

The first step is to retrieve the changes in the local element matrices. For each modified element e_i, the difference between the new and the original element matrix $\mathbf{K}_{e_i}$ is retrieved using Equation (2.34).

$$\mathbf{D}_{e_i} = \mathbf{K}_{e_i,new} - \mathbf{K}_{e_i,0} \tag{2.34}$$

The next step is to set up the matrices $\mathbf{C}_{e_i}$ and $\mathbf{U}_{e_i}$ for each $\mathbf{D}_{e_i}$. The matrix $\mathbf{C}_{e_i}$ is a diagonal matrix that contains the eigenvalues of $\mathbf{D}_{e_i}$. The matrix $\mathbf{U}_{e_i}$ contains the corresponding eigenvectors.

In this eigendecomposition, only the sufficiently large eigenvalues are kept, while the small eigenvalues are discarded. This is done using Equation (2.35). In this equation, λ_j is the j-th eigenvalue, $j \in 1, 2, \ldots, n$ of the element stiffness matrix of size n, and $\lambda_{\mathsf{max}} = \max|\lambda_1|, \ldots, |\lambda_n|$. A minimum value of, for example, $\epsilon > 10^{-4}$ is chosen, with which all eigenvalues that are at most a factor 10^4 smaller than the largest eigenvalue are selected.

$$\epsilon = \frac{|\lambda_j|}{\lambda_{\text{max}}} \tag{2.35}$$

For each modified element, a reduced $\mathbf{C}_{e_i}$ and $\mathbf{U}_{e_i}$ has now been found. The next step is to set up the global $\mathbf{C}$ and $\mathbf{U}$. With these matrices, the difference in the total matrix can eventually be described. The structure of $\mathbf{C}$ and $\mathbf{U}$ for k modified elements is shown in Equation (2.36).

$$\mathbf{K}_{new} = \mathbf{K}_0 + \mathbf{U}\mathbf{C}\mathbf{U}^T = \mathbf{K}_0 + [\mathbf{U}_{e_1} \ \cdots \ \mathbf{U}_{e_k}] \begin{bmatrix} \mathbf{C}_{e_1} & & \\ & \ddots & \\ & & \mathbf{C}_{e_k} \end{bmatrix} \begin{bmatrix} \mathbf{U}_{e_1}^T \\ \vdots \\ \mathbf{U}_{e_k}^T \end{bmatrix} \tag{2.36}$$

As may already be noticeable, to be mathematically correct, matrix $\mathbf{U}$ must have the same number of rows as $\mathbf{K}_0$. In this matrix, the $\mathbf{U}_{e_i}$ are placed at the rows where a change has been made in the global matrix. The other entries are zero.

Now that it is clear how $\mathbf{C}$ and $\mathbf{U}$ are set up, it can be explained how the displacement vector $\mathbf{u}_{new}$ of the new system can be calculated in a computationally cheap way. The following steps must be followed for this.

Step 1
Solve the system

$$\mathbf{K}_0\mathbf{u}_0 = \mathbf{f}, \tag{2.37}$$

for $\mathbf{u}_0$ by using the known factorisation of $\mathbf{K}_0$.

Step 2
Solve the system

$$\mathbf{K}_0\mathbf{Z} = \mathbf{U}, \tag{2.38}$$

for $\mathbf{Z}$ by using the known factorisation of $\mathbf{K}_0$.

Step 3
Calculate

$$\mathbf{E} = \mathbf{C}^{-1} + \mathbf{U}^T\mathbf{Z} \tag{2.39}$$

Step 4
Calculate

$$\mathbf{y} = \mathbf{U}^T\mathbf{u}_0. \tag{2.40}$$

Step 5
Solve the system

$$\mathbf{E}\mathbf{z} = \mathbf{y} \tag{2.41}$$

for $\mathbf{z}$ by calculating a factorisation of $\mathbf{E}$.

Step 6
Calculate the solution to the system of equations with

$$\mathbf{u}_{new} = \mathbf{u}_0 - \mathbf{Z}\mathbf{z}. \tag{2.42}$$

Instead of having to solve a new expensive system, the Woodbury Identity makes it possible to reuse an existing factorization to obtain the solution. This factorization can, for example, be an LU decomposition, but this is not further explained in this thesis. For more information, see [15]. With the matrix and vector multiplications in the steps above, it becomes possible to solve a significantly smaller system of equations in step 5.

Strengths and weaknesses of Woodbury Identity

Below, the strengths and weaknesses of Woodbury Identity are outlined.

Strengths

- **Provides exact solutions:** The use of the Woodbury Identity, like substructuring, provides exact results.
- **No need to know the location of parameter changes in advance:** Woodbury is purely a mathematical method to more efficiently compute partially modified matrices. Therefore, no prior knowledge is needed about where changes will be made later, since changes can in principle occur at any location.

Weaknesses

- **Not effective for changes in large parts of the structure:** The more elements are changed, the larger $\mathbf{C}$ and $\mathbf{U}$ become. As a result, $\mathbf{E}$ from step 3 also becomes larger, and solving the system in step 5 becomes more computationally expensive. The fewer elements are changed in the structure, the more effective the Woodbury Identity is. Therefore, for large changes in the structure, the Woodbury Identity is not beneficial.
- **Does not support variation in force location:** It is not possible to change the force location. The Woodbury Identity only focuses on modifications to the stiffness matrix, not on changes to the right-hand side of the system.

2.5. Findings and initial objectives

This chapter originated from SQ1: '*What is the current state of the art in techniques that enable parametric changes within Reduced Order Models for static analysis?*'. Based on an extensive literature review, four techniques were identified that allow direct or faster parametric changes in ROMs for static analysis. These are the Multiparameter Moment Matching Method, Proper Orthogonal Decomposition Radial Basis Function Approximation, (Multilevel) Substructuring, and the Woodbury Identity. In this chapter, these techniques are explained, and their strengths, weaknesses, and aspects that are still unknown in the literature are discussed. In this way, the state of the art in techniques that enable parametric changes within Reduced Order Models for static analysis is clarified, and SQ1 is answered. This is the first time such a comprehensive and comparative overview of all relevant techniques for static analysis has been brought together in a single study.

In Figure 2.7 an overview is given of the strengths, weaknesses, and unknowns from the literature for the four identified techniques. Several points are highlighted with color to allow comparison of certain strengths or weaknesses across methods. Not all points are directly comparable, as the techniques are theoretically very different. For explanations of all the points, see the 'Strengths, Weaknesses, and Unknowns of ...' sections at the end of each technique its subsection.

Based on the knowledge gained from the literature, initial objectives can now be defined for applying techniques that allow for parametric changes in ROMs to potentially speed up ASML's

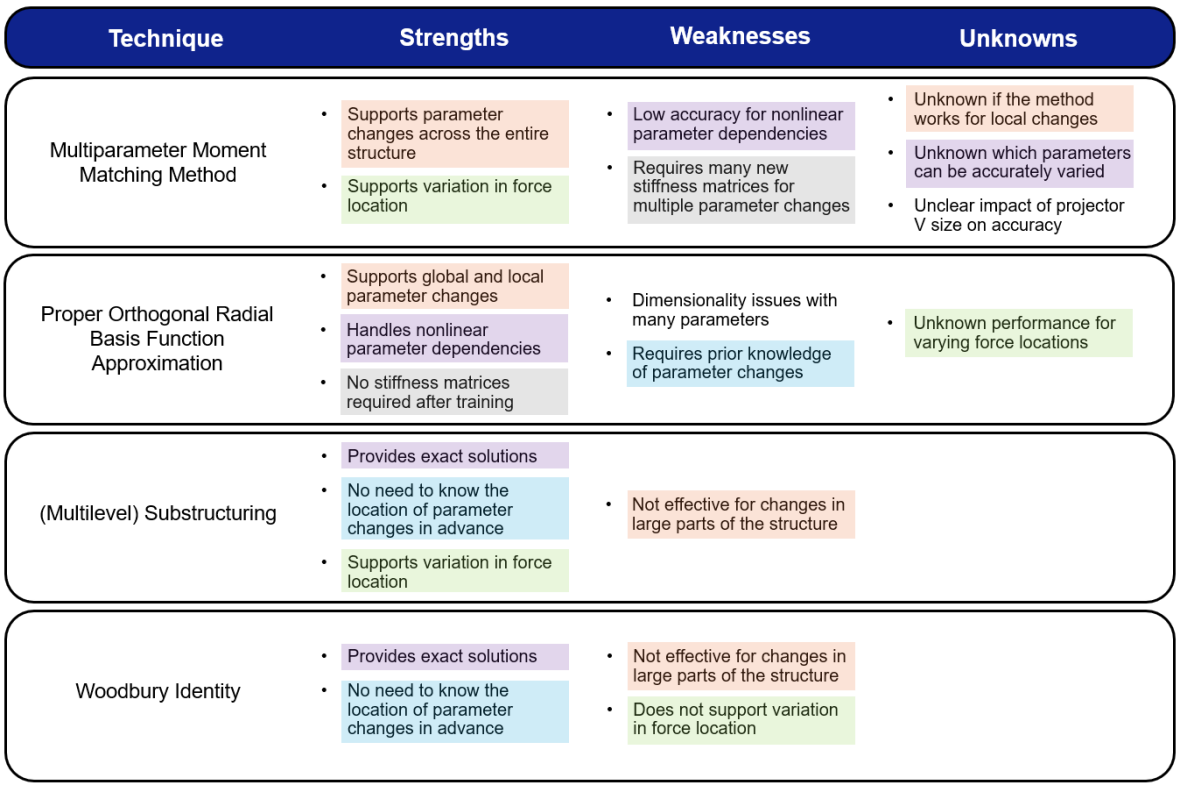

Figure 2.7: Comparison of strengths, weaknesses, and unknowns for the four techniques that enable parametric changes within ROMs for static analysis. Colored markers highlight comparable aspects.

design process. This addresses step 2 of the DSRM. The objectives for a solution are divided into the following four categories:

1. *Parameters to vary:* This refers to which parameters can be varied, such as linear and non-linear structural parameters, as well as force locations and magnitudes. It also includes whether parameters can be changed globally and/or locally in a structure.
2. *Accuracy:* This refers to the size of the error between the displacement vector $\mathbf{u}$ obtained by directly solving Equation (2.1) and the displacement vector obtained using a technique that enables parametric changes within ROMs.
3. *Time reduction:* This refers to the gain in speed within ASML's design process achieved by determining the displacement vector for a new parameter set using a technique that enables parametric changes within ROMs instead of directly solving the system with Equation (2.1).
4. *Integration capability within ASML:* This refers to the extent to which the methods can be applied within ASML's processes.

In Table 2.1 below, initial objectives are presented, along with an explanation of how these objectives were defined for each category. These first objectives are based purely on what seems to be possible according to the literature. At this moment, there is still limited knowledge about ASML's design process and their specific needs, so the objectives are still quite limited and uncertain. To gain more clarity, it is important to better understand ASML's current iterative design processes and their thoughts and wishes on how these techniques could help improve them. Therefore, the next chapter will explore this in more detail, after which the objectives will be revised accordingly.

Category	Objective	Explanation
Parameters to vary	• Multiple parameters can be varied at the same time. • Both structural parameters with linear and non-linear effects on the system, as well as the location and magnitude of the applied force, can be adjusted. • Parametric changes can be applied to both large parts of the structure and to local areas. • When working with non-linear parameters that affect large parts of the structure, the locations of the parametric changes need to be known in advance.	Based on the literature, all these objectives appear to be feasible using different techniques. • All techniques allow for simultaneous variation of multiple parameters. • All techniques except the MMMM can handle non-linear parameters. The MMMM and MLS support variations in the applied force vector (remains unknown for POD-RBF approximation). • POD-RBF approximation supports both global and local changes. For the MMMM, only global changes have been shown to work so far, and for MLS and Woodbury, only changes within the structure are supported. • Non-linear parameters affecting a large portion of the structure can only be handled using POD-RBF approximation, but this needs to be considered during the training phase.
Accuracy	• The displacement vector must be reasonably approximated.	According to the literature, MMMM can achieve reasonable accuracy for linear parameter terms, while POD RBF approximation, if enough interpolation points are used, can handle both linear and nonlinear terms. In addition, Substructuring and the Woodbury Identity provide exact results. A reasonably accurate approximation therefore appears to be feasible.
Time reduction	*No concrete objective(s) can yet be formulated*	The literature shows how the MMMM and POD RBF approximation avoid repeating the reduction step when parameters change, and how substructuring and Woodbury Identity could potentially speed up this process. However, no concrete objective for time savings can be determined yet, as there is still limited knowledge about the current design process.
Integration capability within ASML	*No concrete objective(s) can yet be formulated*	The literature focuses on the theoretical applicability of the different techniques but provides no information about their integration into processes. Combined with the fact that there is still limited knowledge about the current design processes within ASML, no objective can be defined about this at this stage.

Table 2.1: Initial objectives for implementation of techniques enabling parametric changes within ROMs to speed up ASML's design process.

3

ASML's design process and definition of implementation requirements

This chapter addresses SQ2: '*What does ASML's current design process look like, and what requirements must techniques meet to be implemented in this process?*'. Based on semi-structured interviews this chapter explains what the current process looks like, explores how techniques that allow for parametric changes within ROMs could be implemented, and defines the requirements these techniques must meet for successful implementation. How the interviews with ASML engineers were prepared and conducted is described in Section 3.1. The current design process is outlined in Section 3.2, and the idealized design of a new design process is presented in Section 3.3. Section 3.4 describes more feasible updated design processes using the techniques from the previous chapter, from which the requirements for successful implementation follow. Finally, Section 3.5 discusses the findings and defines updated objectives, which is a reiteration of step 2 of the DSRM.

3.1. Methodology

To gain a good understanding of ASML's current iterative design process and the potential benefits the techniques identified in Chapter 2 could offer, semi-structured interviews were conducted with ASML employees. The choice for semi-structured interviews was made because of the need for both structure and flexibility. This approach makes it possible to explore specific themes, such as ASML's current iterative design process and the possible implementation of the identified techniques, while still leaving space to ask follow-up questions based on individual experiences. Below, the selection of interviewees, the interview process, and the content that emerged during the interviews are explained.

Interviewee selection

When selecting the people to be interviewed, the following three aspects were important:

1. **Experience with ASML's design process:** To be able to say something meaningful about ASML's current design process, it is important that the interviewee has experience with it. Preference was given to people who are still involved in this process in their current role, to get a clear understanding of its current state.
2. **Representation from different teams:** The iterative design process is not limited to just one team. Therefore, it is important that the interviewees come from different parts of the organization to see whether they face similar challenges and identify the same

opportunities for the techniques that enable parametric changes within ROMs.

3. **Willingness to participate:** It is crucial that the interviewees are willing to be interviewed and open to share their experiences and insights.

In total, five ASML employees were selected. Details about the interviewees are shown in Table 3.1. All five have significant experience with the current design process, and except for interviewees 3 and 4, they each come from different teams.

The number of five interviewees is sufficient for this research to identify the main challenges within the iterative design process and the potential of PROM in this context. Even though the interviewees came from different teams, similar answers quickly started to appear, indicating data saturation. Moreover, this is an exploratory study within a single organization, which makes the chosen sample size appropriate.

Interviewee	Role	Experience	Team	Date of Interview
1	Senior Mechanical Engineer	> 30 years	T1	21-1-2025
2	Mechatronics Engineer	> 8 years	T2	29-1-2025
3	Mechanical Expert	> 25 years	T3	12-2-2025
4	Mechanical Engineer	> 2 years	T3	20-3-2025
5	Mechanical Engineer	> 10 years	T4	27-3-2025

Table 3.1: Details of interviewees.

Interview process

The process of conducting the interviews is outlined below:

1. **Initial contact and scheduling:** All interviewees were identified and contacted through a professional network within ASML. A personalized message was sent, explaining the purpose of the research and inviting them for an interview. When an invited participant accepted the invitation, the interview was scheduled based on the availability and preference of the participant.
2. **Informed consent process:** As part of the invitation, an informed consent form was included. This form explained the purpose of the study and how the data from the interview would be managed. By signing the form, the interviewee gave permission for the interview to be audio recorded and transcribed.
3. **Interview itself:** Once the informed consent form was signed, the interview took place on the agreed date. Interviews were conducted in English or Dutch, depending on the participant's preference. Audio and transcription were directly recorded with Microsoft Teams if the interview took place online, or with Clipchamp's audio recording and transcription tool if the interview was face-to-face. Each interview was scheduled for one hour. The next section, 'Interview Content', provides more detail about what was discussed during the interviews.
4. **Transcription and feedback:** After the interview, the transcription was anonymized, corrected based on the audio recording, and translated into English if needed. After this, the audio recording was deleted. Participants were given the opportunity to review the transcription and provide comments or corrections.

Interview content

Each interview was structured into three parts: the introduction, the main body, and the closure. For the full interview guide, including all questions used during the interviews, see Appendix B.

The introduction covered a short explanation of the research, a reminder about the signed consent form, and a request for the interviewee to briefly introduce themselves. Next, in the main body, the prepared questions were asked. These questions were divided into two themes. The first theme focused on the current iterative design process within ASML, while the second theme explored the potential use of techniques that enable parametric changes within ROMs in this process. Finally, in the closure, the interviewees were asked if they had any additional information to share and were thanked for their participation.

Based on the interview setup and execution described above, valuable insights were gathered about ASML's current design process and the techniques used within it. This will be explained in the following subsection. The interviews also revealed the potential role of the techniques that enable parametric changes within ROMs in the current process, which will be discussed in the subsection after.

3.2. The current design process

The interviews give insight into how ASML makes use of substructuring, as explained in the first section below. This is followed by a section that describes step-by-step what the current design process looks like. Finally, key considerations for this process regarding static vs. dynamic analysis are discussed.

Application of substructuring

ASML's machines have grown significantly in both size and complexity over the past decades. This has resulted in a FEM model of the entire machine containing tens of millions DOFs. Since analyzing the entire machine with so many DOFs requires an extremely high computational load, ASML applies substructuring. The steps described in Subsection 2.2.2 are followed here. First, a machine is divided into different modules that form the substructures, see Figure 3.1. The number of modules depends on the type of machine, but it is typically around twenty. In addition, the underlying factory floor is also considered a substructure in this setup. The interface points between the different modules form the master DOFs, while the rest are the slave DOFs. For static analyses, Guyan reduction is applied to obtain the reduced stiffness matrix of the module. For dynamic analyses, Craig-Bampton is used. The reduced matrices of all modules are then assembled into a system that describes the entire machine. This matrix now contains only the DOFs of the interface points, reducing it to approximately 500 DOFs. With this reduced matrix, analyses of the machine's behavior can now be performed in a computationally efficient manner.

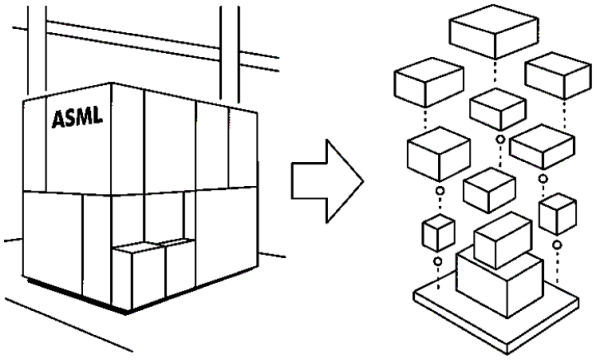

Figure 3.1: Visualization of the modular setup of the ASML machine and factory floor, used for substructuring purposes. [30]

Step-by-step outline of the current design process

During the design process of a machine changes are frequently made at the module level. For instance, certain parts within a module are varied in geometry or material properties trying to meet the desired design requirements. Based on the interviews, several steps in this design process have identified and are visualized in Figure 3.2 (absolute times removed; not for public release).

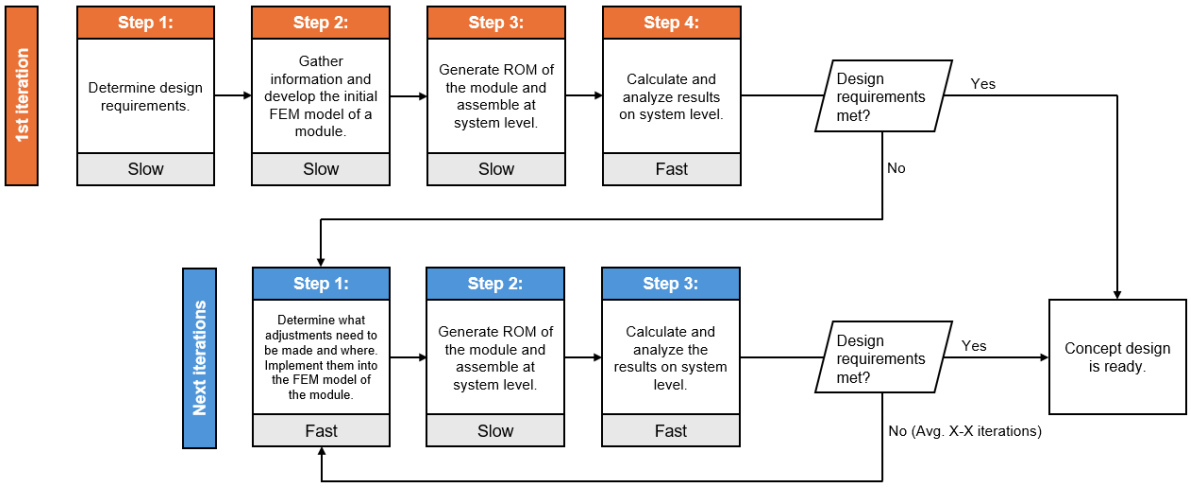

Figure 3.2: Flow chart showing the step-by-step current design process within ASML.

In the figure, it can be seen that the design process essentially consists of two parts. In the first part, shown in the top of the figure, the first iteration is depicted. Here, the conceptual design is constructed for the first time. Before building the conceptual design the first step is to determine the desired result. An example of this is ensuring that the displacements between the different modules remain within certain limit states. The second step involves gathering information from various parties to set up an initial version of the FEM model. Next, in the third step, a reduction must be applied to the module that is designed, and the reduced matrices of all modules need to be reassembled into a system that encompasses the entire machine. The final step of the first iteration is to calculate and analyze this coupled system. In this way, it can be determined whether the design requirements are met.

It is of course uncommon to meet the design requirements in one iteration and so there is a high chance that a second iteration will be necessary. Based on the knowledge and gut feeling of the ASML engineer, it is determined which adjustments should be made to the design. These typically involve small changes to just a single parameter to effectively analyze how the system responds. After this has been decided, the same steps are followed as in the first iteration. At the end of each iteration, it is re-evaluated whether the design requirements are met. If not, a new iteration takes place. This process continues until the desired design requirements of the design are reached.

The figure also shows the time duration for each step. It can be seen that doing one iteration takes about X to X (X inserted; not for public release). Since only small changes are made each time, as larger changes would provide little insight, a significant number of iterations may be required. On average around X iterations are needed. Achieving the desired concept design can therefore take months.

In the current design process, ASML uses two software applications: ANSYS and MATLAB. First, the different modules of the machine and the factory floor are set up in ANSYS. Master

DOFs are assigned and the ROMs are created in ANSYS using macros. A macro is a series of ANSYS commands written in ANSYS APDL code, stored in a library file so they can be easily called. The ROMs of the different modules are then imported into MATLAB, where the system is calculated.

Static vs. dynamic analysis considerations

Although this study focuses only on static analyses, it is important to note that a dynamic analysis is normally also performed. Therefore, the following key considerations are important to keep in mind.

The need for substructuring of the machine mainly comes from the dynamic analysis process. Calculating a machine with tens of millions of DOFs for static analysis is already complex, but for dynamic analysis this is significantly more demanding as these calculations require much more computational power and memory [8].

In addition, it is important to mention that creating a ROM for dynamic analyses takes much more time than for static analyses. While generating a ROM for a static analysis using Guyan reduction of a module with millions of DOFs can already take some time, this process takes significantly more time for dynamic analyses using the Craig-Bampton method [11]. This was also emphasized by the ASML engineers during the interviews. In Figure 3.2, a time indication is shown related to setting up a module reduction for both static and dynamic analyses. If only a static analysis needs to be performed this step will take a lot less time.

3.3. The idealized design process

In this section, the idealized design process is established first. Ten points are defined here that apply to an ideal design. The existing techniques from Chapter 2 are then tested against these ten points.

Establishing the idealized design process

In line with the principles of idealized design, the ideal version of the design process is outlined first. The idea behind idealized design, introduced by Ackoff [31], is that the existing system is assumed to no longer exist, giving the opportunity to redesign it without current limitations. However, an important guideline is that the new design should be theoretically feasible and able to function within the current environment.

The goal is not to design a process that can be implemented right away, but to create a guiding and desirable vision for the future. This ideal situation serves as a conceptual benchmark. Then, theoretical and practical limitations are added to decide which parts of the idealized process can be kept and which need to be changed or removed to make implementation possible. [3]

Based on the interviews the ten conditions below were formulated that should be met in this idealized design. These conditions are divided into the four objective categories introduced in Section 2.5.

- **Parameters to vary:**
    1. Multiple parameters can be varied simultaneously.
    2. Allows for changes in both parameters with linear and nonlinear dependencies in the structure, as well as in force locations and magnitudes.
    3. Parameter changes can be applied both globally and locally.

4. No prior knowledge is needed when setting up the technique about where and what parametric changes will later be made.

- **Accuracy:**

5. Provides nearly exact results.

- **Time reduction:**

6. Setting up the technique does not take a lot of time.
7. Calculating a new parameter set is fast.

- **Integration capability within ASML:**

8. Fits within the substructuring approach.
9. Compatible with the software used at ASML.
10. Both static and dynamic analyses can be performed.

When meeting this conditions, the design process looks like the one shown in Figure 3.3 (absolute times removed; not for public release). The figure shows that step 3 in the first iteration of the current design process has been replaced by two new steps. This is because a standard reduction is no longer performed. Instead, a reduction is used that allows parametric changes to be made easily at a later stage. In step 4, the first parameter set can now be selected at the module level and then evaluated at the system level. In the next iterations, it is no longer necessary to perform a new reduction each time. Instead, only a new parameter set needs to be calculated. This means that a new iteration now takes only a few minutes to a few hours, which allows the process of reaching a design that meets the design requirements design to be completed much faster. This idealized approach shows the great potential of using these kinds of techniques. Comparing Figure 3.2 and Figure 3.3, it can be seen that this can be up to X to X (X inserted; not for public release).

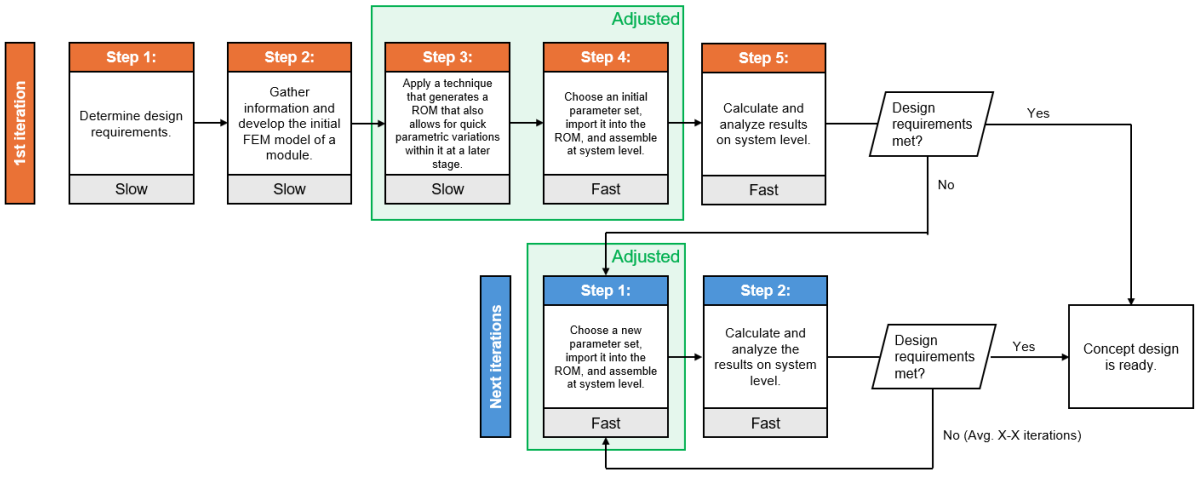

Figure 3.3: Flow chart the step-by-step idealized design process using a technique that enables parametric changes within ROMs.

Testing the idealized design process

The next step is to test the existing techniques from Chapter 2 against this ideal situation. Table 3.2 gives an overview of how each technique relates to each condition. In the figure, a green color means that the condition is fully met. On the other hand, a red color means the

condition is not met. Yellow and orange are in between, where yellow means the condition is met more than in the case of orange.

Category & Condition		MMMM	POD-RBF	MLS	Woodbury
Parameters to vary	1	✓	✓	✓	✓
	2	Low accuracy for nonlinear parameter dependencies. Supports variation in force location.	Handles nonlinear parameter dependencies. Unknown performance for varying force locations.	✓	Does not support variation in force location.
	3	Supports parameter changes across the entire structure. Unknown if the methods works for local changes.	✓	Not effective for changes in large parts of the structure.	Not effective for changes in large parts of the structure.
	4	Supported for what parameter, still unknown if method works for local changes.	Requires prior knowledge of parameter changes.	✓	✓
Accuracy	5	Relatively accurate for linear parameters. Unknown accuracy for partly nonlinear parameters.	Relatively accurate with enough interpolation points.	✓	✓
Time reduction	6	Relatively fast.	Dimensionality issues arise when the number of parameters increases.	Does not need to be set up beforehand.	Does not need to be set up beforehand.
	7	Requires many new stiffness matrices for multiple parameter changes.	No stiffness matrices required after training.	Dependent on the number of substructures that need to be modified.	Dependent on the number of elements that need to be modified.
Integration capability within ASML	8	Provides a displacement vector, it is unknown whether this is compatible with substructuring.	Provides a displacement vector, it is unknown whether this is compatible with substructuring.	✓	✓
	9	✓	✓	✓.	✓
	10	Extension to dynamics available.	No direct extension to dynamics available.	Also works for dynamic analysis.	No direct extension to dynamics available.

Table 3.2: Overview of how each technique matches the idealized design conditions.

What stands out when looking at the table is that each technique has its own advantages and disadvantages compared to other methods. It also becomes clear that several aspects are still unknown for the MMMM and POD-RBF approximation. Based on this table, the idealized design process, and the literature, the different techniques will be integrated into the current design process in the next subsection.

3.4. Feasible adaptations and implementation requirements per technique

In this subsection each of the four different techniques is discussed one by one, explaining how the technique fits into the current design process. These are essentially feasible adaptations of the idealized design process, based on the available technical possibilities according to the literature. For each implemented technique the requirements are then defined that must be met to be successfully integrated into the design process.

In the list of requirements below, only aspects have been included for which it is still uncertain whether the technique meets them, or where the practical applicability is not guaranteed. If a

technique is already proven to be capable in a certain area based on the literature, no additional requirement is formulated for that aspect. The requirements therefore focus specifically on areas that still need attention or verification to allow successful integration into ASML's design process. After the technique-specific requirements, two general requirements are added at the end that apply to all techniques.

3.4.1. MMMM

Looking at Table 3.2, this technique may initially seem like a weak alternative compared to the others, given the large number of yellow and orange-colored cells. However, there are several points that make the MMMM uniquely positioned compared to the rest, which is why this technique can definitely be valuable in the design process. For example, together with the POD-RBF approximation, the MMMM is the only technique that allows for changes in large parts of the structure. So if this is desired, these techniques seem to be the only options. An advantage compared to the POD-RBF approximation is that the MMMM is quicker to set up and has an extension towards dynamics. This extension towards dynamics is very relevant because, as also mentioned in Section 3.2, generating a ROM for static analysis takes much less time than for dynamic analysis. The significant time savings to speed up the design process can therefore be found in the application to dynamics. Although this falls outside the scope of the current study, it remains highly relevant to consider.

In Figure 3.4a, it is shown how the MMMM would fit into the current design process (absolute times removed; not for public release). An important difference that can be seen when this process is compared to the ideal design process in Figure 3.3 is that it still requires the creation of new stiffness matrices for the adjusted parameters. Based on the unknowns (see Figure 2.7 and Table 3.2) and this updated design process, the following requirements have been defined to successfully implement the MMMM into the design process:

1. **Understanding of which parameters allow accurate variation.** To make informed use of the method, it is important to identify which parameters can be varied with acceptable accuracy.
2. **The technique must support changes in localized regions of the structure.** If parameter variations can only be applied globally, the technique's use in modular designs becomes very limited.
3. **The displacement error at interface DOFs must remain below 5%.** In the modular approach it is important that the results at the interface DOFs are within a reasonable level of accuracy as this is where the module is connected to the rest of the system. An accuracy of no more than 5% is required to guarantee that the module's interaction with the rest of the system is sufficiently accurate. For static analysis, this means that displacements obtained from MMMM at the interface nodes must not differ more than 5% from the direct solution of Equation (2.1).
4. **The technique must be compatible with substructuring approaches.** It must be compatible with substructuring in order to generate the full system model and integrate it into ASML's design process.

3.4.2. POD-RBF approximation

This is the second technique that allows changes in larger parts of the structure. An advantage of this method compared to the MMMM is that it can also include nonlinear parameters with relatively good accuracy. In addition, this method is faster once it has been set up.

How the POD-RBF approximation would fit into the design process is shown in Figure 3.4b.

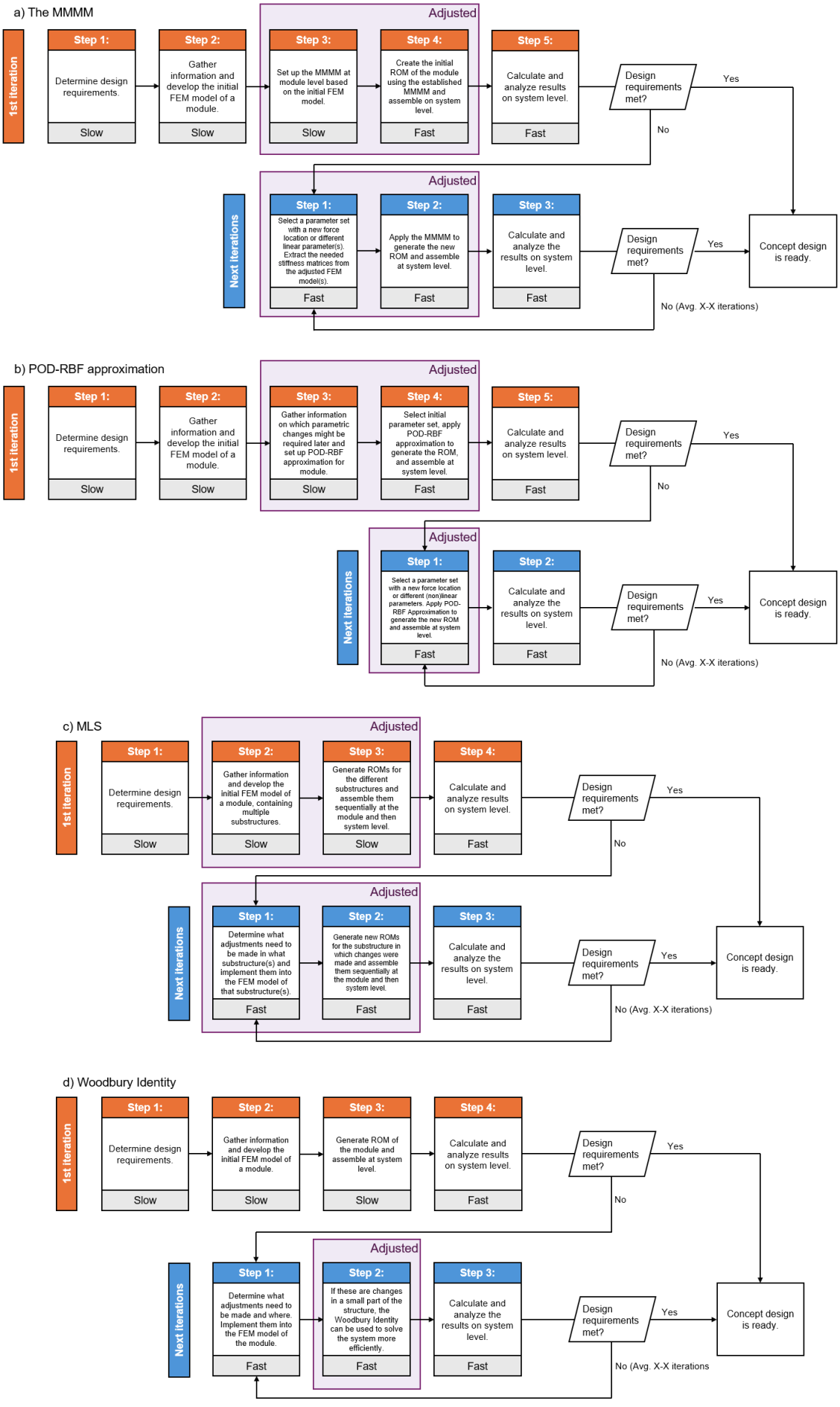

Figure 3.4: Flow chart for a) MMMM, b) POD-RBF approximation, c) MLS, and d) Woodbury Identity, showing a feasible updated step-by-step design process.

The main difference compared to the ideal design process in Figure 3.3 is that setting up the POD-RBF approximation takes significantly more time. Based on the unknowns (see Figure 2.7 and Table 3.2) and this updated design process, the following requirements have been defined to successfully implement the POD-RBF approximation into the design process:

1. **The setup time must remain within two weeks.** If generating the interpolation data (e.g., in step 3 of Figure 3.4b) requires too many training points and takes months, the benefit of faster iterations is lost. A maximum setup time of two weeks is therefore defined.
2. **Understanding of whether varying force locations can be handled.** To make informed use of the method, it must be clear whether the method can interpolate accurately for different force positions.
3. **The displacement error at interface DOFs must remain below 5%.** As with the MMMM, the displacement error at interface nodes must not exceed 5% compared to the direct solution of Equation (2.1), to ensure sufficiently accurate interaction with the rest of the system.
4. **The technique must be compatible with substructuring approaches.** As with the MMMM, it must be compatible with substructuring in order to generate the full system model and integrate it into ASML's design process.

3.4.3. MLS

If only changes in a small part of the structure are needed, MLS seems to perform well according to Table 3.2. The main difference compared to the previous two techniques is that MLS does not require a completely different way of generating ROMs. Instead, the existing ROMs become smaller because the modules are divided into more substructures. This reduces the size of the parts that need to be updated, which helps speed up the current design process.

How MLS can be implemented in the current design process is shown in Figure 3.4c. Based on this updated design process, the following requirement has been defined to successfully implement MLS into the design process:

1. **Understanding of the number of substructures that can be varied effectively.** To make use of MLS, it is important to have insight into how many substructures can be modified within a modular setup before it becomes more efficient to regenerate the entire ROM instead.

3.4.4. Woodbury Identity

Looking at Table 3.2, MLS seems to have an edge over the Woodbury Identity on all fronts. However, there is one situation in which the Woodbury Identity has an advantage. If small changes are needed in different, spread-out parts of the structure, MLS would require each substructure to be reduced again, which cancels out its effectiveness. With the Woodbury Identity, the changed elements can be updated separately, since it does not depend on substructures. In this situation, the Woodbury Identity is therefore preferred.

Also, the fact that it works independently of substructures within a module can be an advantage if that is needed in a specific case. How the Woodbury Identity can be implemented in the current design process is shown in Figure 3.4d. Based on this updated design process, the following requirement has been defined to successfully implement the Woodbury Identity into the design process:

1. **Understanding of the number of element changes that can be handled effectively.** To make use of Woodbury Identity, there must be insight into how many elements can be modified before it becomes more efficient to regenerate the entire ROM instead.

3.4.5. General requirements

In addition to the requirements that apply to each individual technique for successful implementation, there are also several general requirements that apply to all techniques which have not yet been mentioned above. These requirements are explained below:

A. **Implementation must rely solely on existing ASML software tools.** The implementation of any new method must be compatible with software already in use at ASML (ANSYS and MATLAB), to ensure practical integration without requiring additional licensing or training.

B. **Recalculation time for new parameter sets must remain below 20 minutes.** Since the goal of the method is to accelerate design iterations, the complete recalculation of a new parameter set, including stiffness matrix generation, should take no more than 20 minutes. This allows multiple parameter evaluations within a single working day and contributes significantly to overall process efficiency.

3.5. Findings and updated objectives

This chapter originated from SQ2: '*What does ASML's current design process look like, and what requirements must techniques meet to be implemented in this process?*'. Based on semi-structured interviews, the current design process was described. Then using the idea of idealized design [31] a new ideal design process was created. Next, based on this idealized design and the theoretical limitations of the different methods, a feasible updated design process was developed. From this new feasible updated design process, requirements were defined that must be met in order to successfully implement the methods thus answering SQ2. The requirements are shown per technique in Table 3.3. The requirements listed under "General" apply to all techniques.

Technique		Requirement
MMMM	1	Understanding of which parameters allow accurate variation.
	2	The technique must support changes in localized regions of the structure.
	3	The displacement error at interface DOFs must remain below 5%.
	4	The technique must be compatible with substructuring approaches.
POD-RBF	1	The setup time must remain within two weeks.
	2	Understanding of whether varying force locations can be handled.
	3	The displacement error at interface DOFs must remain below 5%.
	4	The technique must be compatible with substructuring approaches.
MLS	1	Understanding of the number of substructures that can be varied effectively.
Woodbury	1	Understanding of the number of element changes that can be handled effectively.
General	A	Implementation must rely solely on existing ASML software tools.
	B	Recalculation time for new parameter sets must remain below 20 minutes.

Table 3.3: Overview of the requirements per technique for successful implementation into the ASML design process.

An important conclusion from the interviews is that creating a ROM for dynamic analyses takes much more time than for static analyses. With this thesis focusing only on static analysis, the potential to significantly speed up the process is more limited, since the time saved by skipping each reduction step is smaller. Nevertheless, it remains relevant, as some time savings might

still be achieved, and static analysis techniques often form the basis for those used in dynamic analysis.

Based on the obtained knowledge in this chapter, the initial objectives from Section 2.5 for applying techniques that allow for parametric changes in ROMs to potentially speed up ASML's design process, can now be updated. This is a reiteration of step 2 of the DSRM. In Table 3.4 below, the updated objectives are presented, along with an explanation of how these were defined for each category.

As shown in the table, the objectives have now been significantly expanded. However, a limitation is that these objectives are currently based solely on theoretical insights from the literature on the various techniques. To assess whether these objectives can realistically be achieved in practice, the associated implementation requirements from Table 3.3 will be verified in the next chapter. Based on the results of this verification, the objectives will be revised once again at the end of that chapter.

Category	Objective	Explanation
Parameters to vary	• Multiple parameters can be varied at the same time. • Both structural parameters with linear and non-linear effects on the system, as well as the location and magnitude of the applied force, can be adjusted. • Parametric changes can be applied to both large parts of the structure and to local areas. • When working with non-linear parameters that affect large parts of the structure, the locations of the parametric changes need to be known in advance.	*This has not been changed.*
Accuracy	• The displacement vector on the interface DOFs must have a maximum deviation of 5% compared to the direct solution	From the interviews, it became clear that a maximum deviation of 5% on the interface DOFs compared to the direct solution provides sufficient accuracy.
Time reduction	• Setting up a technique should not take longer than 2 weeks. • Running a new parameter set should take no more than 20 minutes.	This comes from the interviews and the setup of the updated design processes: • The setup of the technique should not take too long as the benefit of faster follow-up iterations would otherwise be lost. This is especially important for POD-RBF approximation. The limit of 2 weeks was therefore chosen as a realistic objective. • If recalculating takes too long, it becomes impossible to quickly evaluate different parameter sets. This is especially relevant for the MMMM, MLS, and Woodbury methods, as with these techniques new stiffness matrices need to be generated for each new parameter set.
Integration capability within ASML	• Implemented techniques must fit within the substructuring approach at ASML. • The methods should only use software that is already used at ASML.	The interviews showed that these objectives are important for an updated design process at ASML, as smooth integration into existing tools and workflows is essential for successful adoption.

Table 3.4: Updated objectives for implementation of techniques enabling parametric changes within ROMs to speed up ASML's design process.

4

Verification of techniques against implementation requirements

This chapter addresses SQ3: '*Do the existing techniques, either in their existing form or adapted, meet the requirements for implementation within ASML's design process?*'. In the previous chapter, requirements were defined for the different techniques to enable successful implementation in ASML's design process. These techniques are tested on two different cases to examine whether the requirements are met. For some techniques, not all requirements will be met, and innovative adaptations will be proposed and tested in an attempt to still meet these requirements.

In Section 4.1, the cases will first be introduced. Then, in Section 4.2 to Section 4.5, the four different techniques are tested against the requirements defined in the previous chapter. In these sections, innovative adaptations will also be introduced and tested if certain requirements are not met. In this way, steps 3, 4, and 5 of the DSRM are essentially followed for each technique, as the techniques are developed (step 3 of the DSRM), demonstrated on the cases (step 4 of the DSRM), and evaluated (step 5 of the DSRM). In Section 4.6, the findings are discussed, and a final reiteration to step 2 of the DSRM is made by finalizing the objectives.

4.1. Introduction to the cases used for verification

Two cases are introduced on which different techniques will be applied and evaluated against the defined requirements. The first case, introduced in Subsection 4.1.1, uses simple Euler-Bernoulli (EB) beam elements and can be set up manually in MATLAB. The second case, introduced in Subsection 4.1.2, is slightly more complex as it uses shell elements and is therefore set up with the help of ANSYS.

4.1.1. Case 1: beam model

This first case will consist solely of EB beams. This element is suitable for modeling slender beams where transverse shear deformation is neglected. In this case, a 2D analysis is performed. A single EB beam element defined by two nodes then has a total of six degrees of freedom (DOFs). These include displacements x_1, y_1, x_2, y_2, and rotations ϕ_1 and ϕ_2, as shown in Figure 4.1.

The stiffness matrix for this element $\mathbf{K}_{el}$ is shown in Equation 4.1. Here, E represents the modulus of elasticity, I is the second moment of area of the cross-section, A is the cross-

sectional area, and L is the length of the element. The displacement vector $\mathbf{u}$ is given in Equation (4.2).

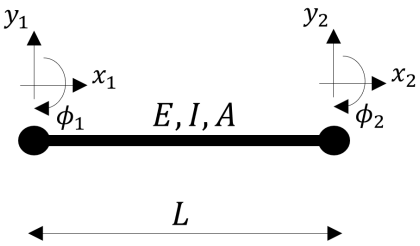

Figure 4.1: Single Euler Bernoulli beam element.

$$\mathbf{K}_{el} = \frac{EI}{L^3}\left[\begin{array}{ccc|ccc} \frac{AL^2}{I} & 0 & 0 & -\frac{AL^2}{I} & 0 & 0 \\ 0 & 12 & 6L & 0 & -12 & 6L \\ 0 & 6L & 4L^2 & 0 & -6L & 2L^2 \\ \hline -\frac{AL^2}{I} & 0 & 0 & \frac{AL^2}{I} & 0 & 0 \\ 0 & -12 & -6L & 0 & 12 & -6L \\ 0 & 6L & 2L^2 & 0 & -6L & 4L^2 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{K}_{11} & \mathbf{K}_{12} \\ \hline \mathbf{K}_{21} & \mathbf{K}_{22} \end{array}\right] \tag{4.1}$$

$$\mathbf{u} = \begin{bmatrix} x_1 & y_1 & \phi_1 & x_2 & y_2 & \phi_2 \end{bmatrix}^T \tag{4.2}$$

The case to which the techniques will be applied is a cantilever beam. A cantilever beam of only four elements in finite element form is shown in Figure 4.2. How a stiffness matrix of such a cantilever beam is formed is presented in Figure 4.3. In the figure it can be seen that the first row and column are crossed out. This is because x_1, y_1, and ϕ_1 are equal to zero for a cantilever beam as these DOFs are restricted.

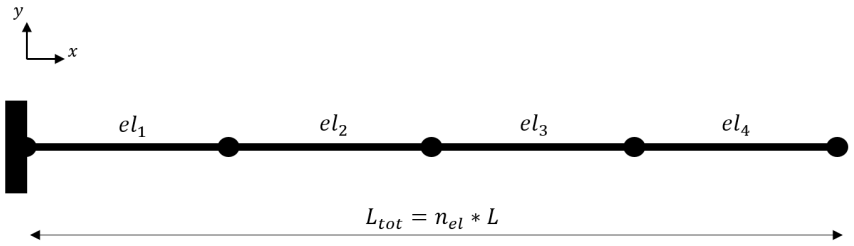

Figure 4.2: Cantilever beam model for four elements in finite element form.

In MATLAB, a code has been developed that automatically generates the stiffness matrix of this cantilever beam, dependent on the number of elements n_{el}. This code can be found under function 3 in Appendix C.1. Unless mentioned otherwise, the case is a cantilever beam with a total of 400 elements. With 3 DOFs per node, this entire system therefore has a total of 1200 DOFs. The application of forces differs for the different techniques and will therefore be explained in each corresponding section.

4.1.2. Case 2: shell model

This next case uses shell elements instead of EB beam elements. New in this case is of course the type of element, but perhaps even more important is the procedure used to determine the stiffness matrices and displacement vectors. Setting up the stiffness matrix of a shell element by hand is very complex, and therefore it is done using ANSYS in this case.

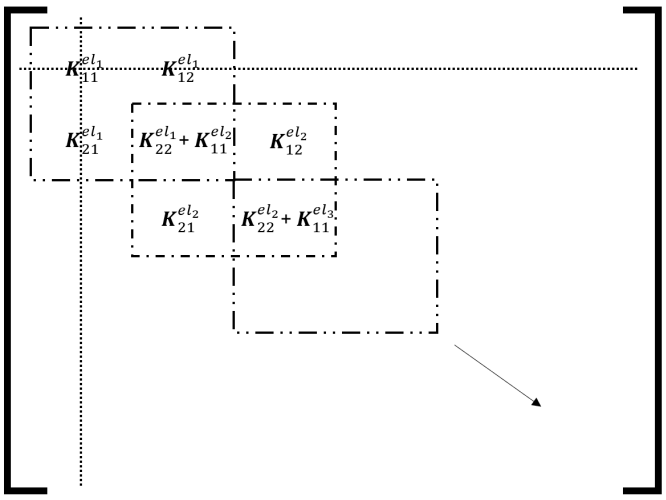

Figure 4.3: Stiffness matrix of cantilever beam superposed by single elements.

The chosen type of shell element from ANSYS is the SHELL181 element. For this element type, each node has the DOF x, y, z, r_x, r_y, and r_z [32]. In the element, E represents the modulus of elasticity, L_x and L_y the element length in the x- and y-direction, and t the thickness. Figure 4.4 shows a single SHELL181 element.

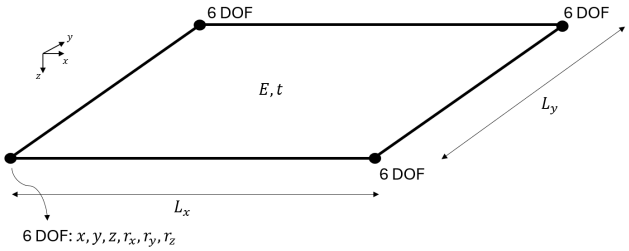

Figure 4.4: Single SHELL181 element with its DOFs and parameters.

Unless mentioned otherwise, the shell structure used for verifying the different techniques consists of 14 by 14 connected shell elements, with the corners restricted in the x, y, and z directions. The structure generated in ANSYS is shown in Figure 4.5. In total, the structure contains 196 elements and therefore, including the boundary conditions, 1338 DOFs. Similar to Case 1, the applied forces vary between techniques and are therefore explained in the respective subsections. The method for extracting the required information from ANSYS may also differ per technique and is explained in more detail in the corresponding chapters, with each referring to Appendix D.

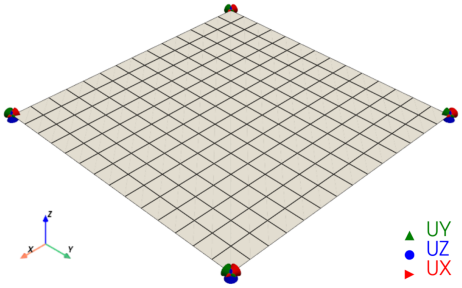

Figure 4.5: In ANSYS generated case 2 model without applied forces.

4.2. MMMM

The requirements in Table 4.1 were defined in Section 3.4 to successfully implement the MMMM into the design process. This section tests and verifies whether the MMMM can meet these requirements. Subsection 4.2.1 shows that requirement 2 cannot be met with the current use of the MMMM. Therefore, Subsection 4.2.2 proposes and tests an adaptation in this technique that may solve this issue. Subsection 4.2.3 evaluates whether the requirements are met.

Technique	Requirement	
MMMM	1	Understanding of which parameters allow accurate variation.
	2	The technique must support changes in localized regions of the structure.
	3	The displacement error at interface DOFs must remain below 5%.
	4	The technique must be compatible with substructuring approaches.
General	A	Implementation must rely solely on existing ASML software tools.
	B	Recalculation time for new parameter sets must remain below 20 minutes.

Table 4.1: Requirements that the MMMM must meet for successful implementation in ASML's design process

4.2.1. Current technique

The MMMM was first applied to Case 1 and then to Case 2. The setup and the results are discussed in the sections below.

Application on case 1

The essential idea behind the MMMM is to construct the projector matrix $\mathbf{V} \in \mathbb{R}^{n \times m}$, which reduces $\mathbf{K} \in \mathbb{R}^{n \times n}$ to $\mathbf{K}_r(p) \in \mathbb{R}^{m \times m}$, where $m << n$. The input for constructing the projector $\mathbf{V}$, as described in Algorithm 2, consists of the system's stiffness matrix and selected force vectors. As seen in the algorithm, the more force vectors are chosen, the more iterations are required, and the larger the number of columns in $\mathbf{V}$. This means that it takes longer to construct $\mathbf{V}$ and the reduction is less effective. Therefore only a limited number of force vectors is preferred. This helps reduce computation time for a new reduction of a parameter set and so supports meeting requirement B. In Figure 4.6 below Case 1 is shown along with its properties and the ten applied force vectors.

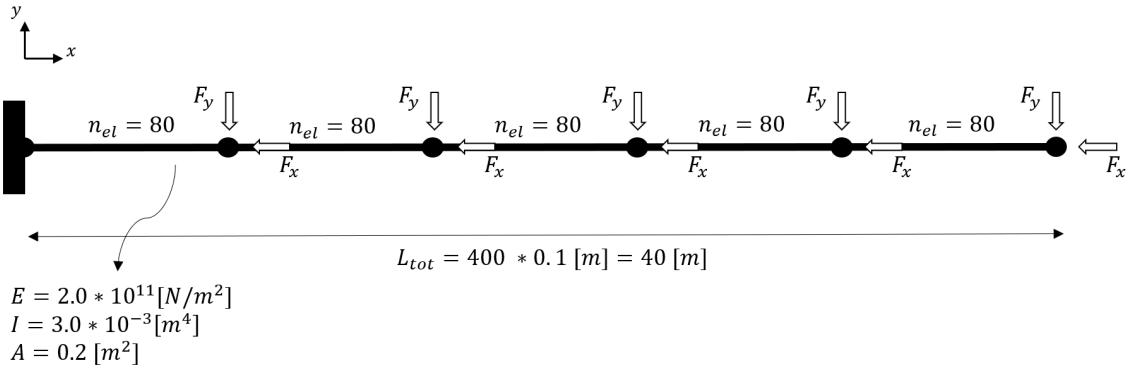

Figure 4.6: Case 1 model for the MMMM.

In step 1 of the MMMM algorithm (Appendix A.2), the first columns of $\mathbf{V}$ are obtained. For each force vector, two columns are generated, except for the first force vector which generates only one column. In this case this results in 19 columns. In step 2, the non-orthogonal columns are filtered out. In this case all columns are orthogonal, so no columns are removed. In step 3,

additional columns are added to $\mathbf{V}$ until the chosen limit is reached. If the limit is set to zero, $\mathbf{V}$ remains at 19 columns. If the limit is set higher, the number of columns in $\mathbf{V}$ increases accordingly. In step 4, non-orthogonal columns are filtered out again.

Now that the projector $\mathbf{V}$ has been constructed for this case, the accuracy of the technique can be tested. The result of Equation (2.24) is compared with the direct solution of Equation (2.1). To give an idea of how this is carried out, the MATLAB code for the MMMM with a variation in the E-modulus is shown in Appendix C.1.

In Table 4.2 the errors are shown for different changing parameters, various placements and magnitudes of F_x and F_y, and different number of columns in projector matrix $\mathbf{V}$. The table is split into four sections where in each section the part that is varied is shown in blue. The first column shows how much each parameter has been varied. The second column indicates the nodes where the forces have been applied. Keep in mind that when setting up $\mathbf{V}$, the forces were applied to nodes 80, 160, 240, 320, and 400 (see Figure 4.6). The third and fourth column show the magnitude of the forces applied to these nodes. The fifth column shows the number of columns in $\mathbf{V}$, and the last two columns show the errors compared to the direct solution. The median error shows the average deviation of each value in the resulting displacement vector $\mathbf{u}$ compared to the displacement vector from the direct solution, while the maximum error is the highest deviation of any error on a specific DOF. The most important one is the maximum error, because requirement 3 states that the maximum error on the interface DOFs must not go above 5%, and in theory, every DOF can be an interface DOF in a connected structure. Still, the median error gives a useful indication of whether the displacements are generally well estimated.

Modified Parameters	Force Application Nodes	F_x on Application Nodes [kN]	F_y on Application Nodes [kN]	Number of Columns in V	Median Error [%]	Max Error [%]
E *2	[160, 400]	[10, 10]	[5, 5]	19	*3.30E-05*	*3.60E-05*
E *½	[160, 400]	[10, 10]	[5, 5]	19	*6.80E-05*	*6.90E-05*
A *2	[160, 400]	[10, 10]	[5, 5]	19	*2.30E-05*	*2.80E-05*
A *½	[160, 400]	[10, 10]	[5, 5]	19	*2.30E-05*	*2.80E-05*
I *2	[160, 400]	[10, 10]	[5, 5]	19	*3.80E-05*	*4.40E-05*
I *½	[160, 400]	[10, 10]	[5, 5]	19	*6.20E-05*	*7.10E-05*
L *1.1	[160, 400]	[10, 10]	[5, 5]	19	*99.94*	*99.99*
L *1.01	[160, 400]	[10, 10]	[5, 5]	19	*95.74*	*99.11*
E *2, A *2, I *2	[160, 400]	[10, 10]	[5, 5]	19	*4.00E-06*	*7.00E-06*
E *½, A *½, I *½	[160, 400]	[10, 10]	[5, 5]	19	*8.60E-05*	*9.90E-05*
E *½, A *½, I *½	[80]	[10]	[5]	19	*4.70E-05*	*1.39E-04*
E *½, A *½, I *½	[90]	[10]	[5]	19	*3.42E-01*	*7.12*
E *½, A *½, I *½	[100]	[10]	[5]	19	*5.42E-01*	*10.04*
E *½, A *½, I *½	[120]	[10]	[5]	19	*5.23E-01*	*11.52*
E *½, A *½, I *½	[140]	[10]	[5]	19	*2.24E-01*	*8.99*
E *½, A *½, I *½	[150]	[10]	[5]	19	*1.02E-01*	*5.78*
E *½, A *½, I *½	[160]	[10]	[5]	19	*6.00E-05*	*1.28E-04*
E *½, A *½, I *½	[120, 200]	[10, 10]	[5, 5]	19	*2.13E-01*	*7.24*
E *½, A *½, I *½	[120, 200, 280]	[10, 10, 10]	[5, 5, 5]	19	*1.05E-01*	*6.18*
E *½, A *½, I *½	[120, 200, 280, 360]	[10, 10, 10, 10]	[5, 5, 5, 5]	19	*5.12E-02*	*4.57*
E *½, A *½, I *½	[80, 160, 240, 320]	[10, 10, 10, 10]	[5, 5, 5, 5]	19	*7.50E-05*	*1.13E-04*
E *½, A *½, I *½	[120]	[10]	[5]	19	*5.23E-01*	*11.52*
E *½, A *½, I *½	[120]	[-100]	[-50]	19	*5.23E-01*	*11.52*
E *½, A *½, I *½	[120]	[100]	[20]	19	*5.23E-01*	*11.52*
E *½, A *½, I *½	[120]	[1]	[5]	19	*5.23E-01*	*11.52*
L *1.1	[120, 200]	[10, 10]	[5, 5]	19	*99.92*	*100.01*
L *1.1	[120, 200]	[10, 10]	[5, 5]	81	*99.89*	*100.01*
L *1.1	[120, 200]	[10, 10]	[5, 5]	301	*29.72*	*97.61*
L *1.1	[120, 200]	[10, 10]	[5, 5]	801	*24.81*	*87.26*
E *½, A *½, I *½	[120, 200]	[10, 10]	[5, 5]	19	*2.13E-01*	*7.24*
E *½, A *½, I *½	[120, 200]	[10, 10]	[5, 5]	81	*6.49E-04*	*7.24*
E *½, A *½, I *½	[120, 200]	[10, 10]	[5, 5]	301	*9.90E-05*	*7.24*

Table 4.2: Results of the MMMM on the case 1 model for varying different parameters, force locations and magnituses, and number of columns in the projection matrix.

Below, the results of each section of the table are explained.

- The first section of table shows that the parameters E, A, and I can vary significantly without causing large deviations. In contrast, even a small variation in L results in a large error. The reason why E, A, and I can be varied without issues, while L cannot, is related to how L appears in the stiffness matrix. As seen in Equation (4.1), the terms E, A, and I only appear to the first power, whereas L appears to the first, second, and third power. This aligns with the literature stating that, since the MMMM is based on a Taylor approximation, nonlinear parameters (such as L) are not well accounted for [14, 15].
- The second section of the table shows that the error is minimal when the force is applied to nodes 80 and 160. This is because, in this case, the forces were placed on both node 80 and 160 when setting up projector matrix $\mathbf{V}$. When the force is applied further away from these nodes, the deviation becomes significantly larger. Additionally, it shows that the number of applied forces does not have a significant impact on accuracy.
- The third section shows that when varying the magnitudes of the forces, the errors remain the same. This is not surprising since the force vectors are normalized in the MMMM Algorithm (Appendix A.2).
- The fourth section shows that when increasing the number of columns for a variation in L, the median error gradually decreases. However, even with more than 800 columns in $\mathbf{V}$, the errors remains significant. The total system contains 1200 DOFs, meaning that the stiffness matrix $\mathbf{K} \in \mathbb{R}^{1200\times1200}$ would only be reduced to $\mathbf{K}_r(p) \in \mathbb{R}^{801\times801}$. Therefore, it can be concluded that adding extra columns to $\mathbf{V}$ slightly improves accuracy, but it is not worthwhile for including L. The section also shows that for a parameter set that already had no significant deviation, the maximum error does not decrease much when adding more columns to $\mathbf{V}$.

In the context of requirement 1, an additional test was performed. The question was raised whether it would also be possible to choose, for example, p^2 as a parameter instead of p, so that the method is not limited to only linear parameters. What is meant here is that in Equation (2.24), Δp_i does not necessarily have to be calculated with $p_1 - p_0$, but can also be calculated as $p_1^2 - p_0^2$ or $p_1^3 - p_0^3$ if the parameter appears in the stiffness matrix in that form. To test this, the results of the three tests below have been compared.

- **Test 1:** Case 1 without modifications.
- **Test 2:** Case 1, but in all entries of the stiffness matrix where I appears (I only appears in this form in the matrix, see Equation (4.1)), it is now replaced by I^2. In Equation (2.24), Δp_i is now calculated using $I_1^2 - I_0^2$ instead of $I_1 - I_0$.
- **Test 3:** Case 1, but only in 2 entries of the stiffness matrix, I is replaced by I^2.

The results are as follows: For test 1 and 2, the median and maximum error are 0%, , and for test 3 the median error is over 23%. Since test 1 and 2 show similar errors and test 3 shows a significantly higher error, it can be concluded that it is indeed possible to choose parameters such as p^2 or another form, as long as they appear in that exact form consistently in the stiffness matrix. Because of the requirement that the parameters must all appear in a consistent form throughout the stiffness matrix, varying L in an Euler-Bernoulli beam does not work.

Since the main interest is in the maximum error, it is useful to explain where this maximum error occurs and how it compares to the median error. The tests show that the error at a

force application node is always zero. The maximum error often appears at the point between two application nodes where a new force is applied. This happens because the columns in $\mathbf{V}$ are based on the responses to the application forces. If a DOF is not well represented in this basis, the error at that DOF becomes larger. This means that, in a substructuring context, an application force can always be placed on the interface DOF, and the error will meet requirement 3 since the error at that point is zero. This behavior only holds when the parameter being varied appears consistently in the same form throughout the stiffness matrix.

Application on case 2

The difference in this case compared to Case 1 is the element type and the way the stiffness matrices are retrieved. Requirement A states that software within ASML must be used for this, which made it important to test whether the MMMM could be applied using only ANSYS and MATLAB. During these tests, it was found that it is indeed possible to extract the matrices using ANSYS and import them into MATLAB. This was done using the Python API of ANSYS, as shown in Appendix D.1.The application of the MMMM is then carried out in the same way as described in Appendix C.1.

During the tests, it was noticed that the MMMM is highly sensitive to rounding errors in the stiffness matrix. Therefore, it is important to retrieve the matrix with enough significant digits. When exporting a sparse matrix in ANSYS, this is automatically done with 14 significant digits, which is sufficient in this case. For full matrices, however, only 4 significant digits are used, which does not provide enough accuracy.

For this case, a similar table was created as Table 4.2. This table and how it was set up can be found in Appendix E.1. The same conclusions are drawn from this as were made for the application on Case 1. For example, parameters that appear consistently in the stiffness matrix can again be varied without causing large errors, while parameters that appear highly inconsistently result in large errors when varied. It was also found again that the error is near zero when forces are applied only to the application nodes, the force magnitude has no influence on the error, and the maximum error does not decrease much when adding more columns to $\mathbf{V}$.

It is interesting to mention the results of varying parameter t. This parameter appears mostly consistent in linear form in the stiffness matrix, but also has some terms up to the third power. Details on how this was determined can be found in Appendix F. The errors from varying t fall between the errors from varying parameters that occur constantly in the stiffness matrix of a SHELL181 element (E), and parameters that occur highly inconsistently in the stiffness matrix (L_x, L_y). The error seems acceptable in this specific case with 1338 DOFs, but tests show that when the number of elements and so the number of DOFs increases, the error also becomes significantly larger. The conclusion remains that only parameters which occur constantly in the stiffness matrix can be varied.

In the context of requirement 2, additional tests were performed in which only a local part of the structure was modified. An example of such a test is shown in Figure 4.7. These tests show that when a small part of the structure is varied, the median and maximum errors increase significantly. Moreover, the analysis shows that the error at a force application node is no longer zero. This has important implication because in a substructuring context, an application force can no longer simply be placed on an interface DOF without anticipated error. This may therefore violate requirement 3. This suggests that requirement 2 cannot be met in this way.

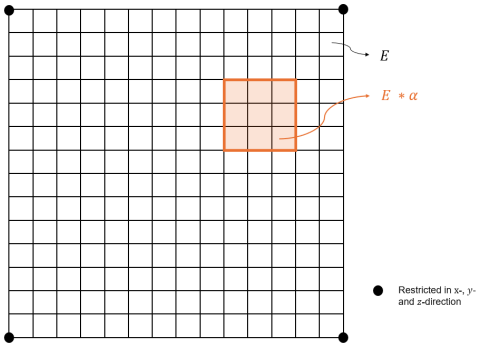

Figure 4.7: Illustration of a local adjustment to $\mathbf{E}$ in the case 2 model

4.2.2. Proposed adaptations

When local changes are not possible without anticipated errors on the locations of the application forces, the MMMM can only be applied if the entire structure is modified. Combined with the fact that only parameters which occur constantly in the stiffness matrix can be adjusted, the practical applicability of the MMMM becomes very limited. For this reason, further research into possible adaptions in the MMMM is done to see whether there is still a way to meet requirement 2.

Varying application forces

The choice of the DOFs on which the application forces are placed when constructing the projection matrix $\mathbf{V}$ strongly influences how well the response at certain DOFs is approximated. Based on this insight, it was investigated whether it is possible to apply the application forces to DOFs around the area where a change occurs. In this adjusted setup, forces were applied to all six degrees of freedom (six DOFs) surrounding the modified area. This is visualized in Figure 4.8. The procedure for retrieving the adjusted stiffness matrices and force vectors with help of the ANSYS Python API is shown in Appendix D.3.

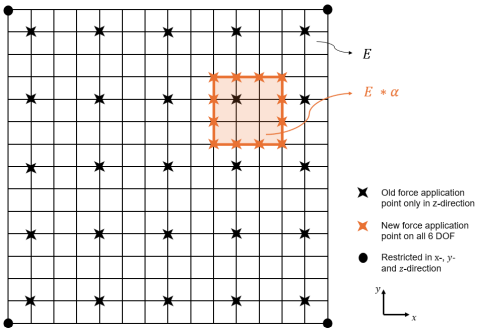

Figure 4.8: Illustration of a local adjustment to $\mathbf{E}$ in the case 2 model with enclosing application forces.

It was found that as long as the force is applied to a DOF that was also used as an application node when constructing $\mathbf{V}$, both the median and maximum error are almost zero. This can be explained theoretically by the fact that the projection matrix $\mathbf{V}$ is based on the responses to the applied forces. If a change occurs within the area already covered by the application forces, this change is fully represented in the basis, which keeps the error minimal.

At first, this seems like a promising finding, but in essence, it is not very different from the reduction achieved through substructuring. In substructuring, the interface DOFs are retained, which means the system is reduced to the number of interface DOFs. In the case of the MMMM, two columns are generated in the projection matrix for each application force, making

the reduction only half as effective compared to substructuring. Moreover, substructuring has the advantage that it is not limited to parameters that occur consistently in the stiffness matrix.

For this reason, a test was also done where only a limited number of DOFs in the modified area were chosen as application force points in the setup of $\mathbf{V}$. It is important to realize that this only becomes beneficial compared to substructuring if less than half (preferably fewer) of the surrounding DOFs are selected as application forces. Significant increases in both median and maximum error were found again. This confirms that an application force can no longer simply be placed on an interface DOF without anticipated error, meaning that requirement 2 cannot be met without violating requirement 3.

Generating one large projection matrix

Based on the finding that both the median and maximum error are near zero when the application forces are placed on DOFs around the area where a change occurs, the idea was developed to create a projection matrix with application nodes placed on all DOFs in the entire structure. From this large matrix the vectors corresponding to the surrounding nodes where a modification is made can then be selected. The main advantage over substructuring would be that there is no dependency on predefined substructures.

When testing this, it turned out that this idea does not work in practice. This is because when constructing $\mathbf{V}$ in the MMMM algorithm (Appendix A.2), each new column depends on the previous one and they only function together as a projection matrix. Therefore, selecting individual columns from this matrix is not possible.

4.2.3. Evaluation against the requirements

It is concluded that it not possible to meet requirement 2 and at the same time also meet requirement 3, without ending up with a weakened version of substructuring. This leads to the conclusion that this technique cannot be successfully implemented in ASML's design process. In Table 4.3 below, each requirement is briefly evaluated. It is important to mention that although this technique does not appear suitable for ASML's design process, a great deal of valuable knowledge was gained by exploring it in depth.

Technique	Requirement		Evaluation	
MMMM	1	Understanding of which parameters allow accurate variation.	This requirement is now fulfilled. The tests above show that variation is possible for parameters that appear consistently throughout the stiffness matrix. Force locations and force magnitudes can also be varied.	✓
	2	The technique must support changes in localized regions of the structure.	In theory, this requirement can be met if the force application point used to set up the projection matrix surround the region where the local change is applied. However, this is now basically a weakened version of substructuring, and therefore it is concluded that this requirement is not fulfilled.	x
	3	The displacement error at interface DOFs must remain below 5%.	This can be achieved for global changes in parameters that appear consistently throughout the stiffness matrix, by ensuring that the force application points are placed on the potential interface DOFs.	✓
	4	The technique must be compatible with substructuring approaches.	*After finding that requirement 2 could not be fulfilled, no further research was done for this requirement.*	?
General	A	Implementation must rely solely on existing ASML software tools.	This is feasible. It is important to extract the stiffness matrix from ANSYS with enough significant digits. This can be done by exporting the matrix as a sparse matrix.	✓
	B	Recalculation time for new parameter sets must remain below 20 minutes.	*After finding that requirement 2 could not be fulfilled, no further research was done for this requirement.*	?

Table 4.3: Evaluation of the MMMM against requirements

4.3. POD-RBF approximation

The requirements in Table 4.4 were defined in Section 3.4 to successfully implement POD-RBF approximation into the design process. This section tests and verifies whether POD-RBF approximation can meet these requirements. Subsection 4.3.1 shows that requirement 1 is hard to meet and therefore Subsection 4.3.2 proposes and tests an adaptation in this technique that may solve this issue. Finally, Subsection 4.3.3 evaluates whether the requirements are met.

Technique	Requirement	
POD-RBF	1	The setup time must remain within two weeks.
	2	Understanding of whether varying force locations can be handled.
	3	The displacement error at interface DOFs must remain below 5%.
	4	The technique must be compatible with substructuring approaches.
General	A	Implementation must rely solely on existing ASML software tools.
	B	Recalculation time for new parameter sets must remain below 20 minutes.

Table 4.4: Requirements that POD-RBF approximation must meet for successful implementation in ASML's design process

4.3.1. Current technique

POD-RBF application was first applied to case 1 and then to case 2. The setup and the results are discussed in the sections below.

Application on case 1

The case is again a cantilever beam with 400 elements. The only difference from the MMMM case is that during each calculation of $\mathbf{u}$ as a column in displacement matrix $\mathbf{U}$, only 2 force vectors were placed on the beam instead of 10 in the MMMM. In Figure 4.9 the standard case with the forces F_x and F_y placed only at three-quarters of the beam can be seen.

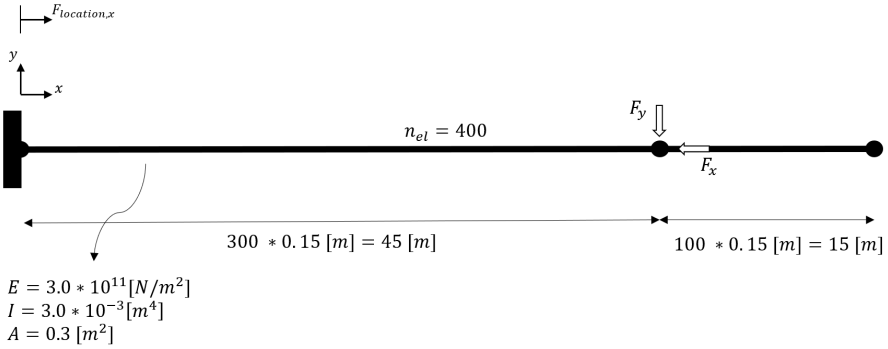

Figure 4.9: Case 1 model for POD-RBF approximation.

In Table 4.5 below, the median and maximum errors are shown for different changing parameters, various numbers of interpolation points, different cut-off values, and varying test values. To give an idea of how these tests were carried out, the MATLAB code for the POD-RBF approximation with a variation in the E-modulus is shown in Appendix C.2.

In the table each test is separated by a horizontal line. The first and second columns show the parameter and its corresponding unit. Compared to the MMMM, what stands out is that the force magnitude F_{size} and the force location $F_{location,x}$ are also included as parameters. This was done in line with testing requirement 2, as this is still unknown at this point.

In the third column, the range of the interpolation values for the different parameters is shown. These are essentially the first and last values of the interpolation points. For $F_{location,x}$, the values 30 $[m]$ and 60 $[m]$ can be seen, which represent the location of the force vector on the beam, where 30 $[m]$ corresponds to the middle of the cantilever beam and 60 $[m]$ to the end. In the fourth column, the number of interpolation points per parameter is shown. A visualization of the interpolation points is presented in Figure 4.10.

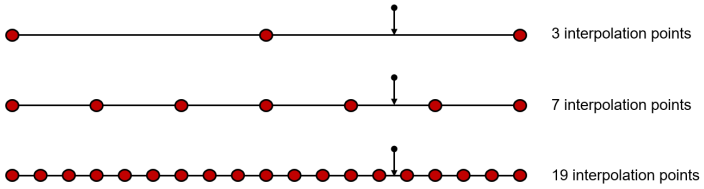

Figure 4.10: Location of the test value for 3, 7, and 19 interpolation points.

In the fifth column, the total number of interpolation points is shown. This is essentially the number of displacement vectors $\mathbf{u}$ that need to be calculated. In the sixth column, the cut-off value K is shown. Only K modes are considered to reduce the model (see Subsection 2.3.2).In the seventh column, the final value of the parameter to be tested is chosen. This value is selected for each parameter so that it falls exactly between two interpolation points, as shown in Figure 4.10. The median and maximum error compared to the direct solution is shown in the last two columns.

Below, the results of each section of the table are explained.

- In the first section of the table it can be seen that for each parameter, both linear and non-linear, relatively small errors are found. This aligns with the literature, which states that structural parameters appearing both linearly and non-linearly in the stiffness matrix yield accurate results [23]. As the number of parameters increases, this error becomes larger. Something important to note about variation in the parameter $F_{location,x}$ is that in this case, only one applied force is used. For situations with two or more applied forces, no interpolation points are available. The error will therefore be high if multiple forces are used and this is not included when training the model. If this is needed, more interpolation points will have to be generated and stored. Another thing that stands out in the table is that the total number of interpolation points increases exponentially with the number of parameters. For example, with 6 parameters and 3 interpolation points per parameter, a total of $3^6 = 729$ interpolation points are required.
- In the second section, errors are shown for varying numbers of interpolation points for the case with parameters E and A. This was done for 3, 7, and 19 interpolation points. For this number of interpolation points, the test value always remains halfway between two interpolation points, as shown in Figure 4.10. It shows that with more interpolation points, the relative error is significantly reduced.
- In the third section, errors are shown for different cut-off values K. It shows that this value should not be chosen too small, as the error increases if not enough modes are taken into account. In this case, the first 3 modes need to be considered to achieve the smallest errors. It is important to note that a higher cut-off value K only increases the training time of the POD-RBF approximation slightly. Most of the training time is actually spent on calculating the displacement vectors $\mathbf{u}$ in the displacement vector $\mathbf{U}$. Therefore, it makes more sense to choose a larger value for K.

Parameter	Unit	Range	Interpolation points per parameter	Total number of interpolation points	K (cut-off)	Test value	Median Error [%]	Max Error [%]
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	3	3	3.50E+11	*2.08*	*2.08*
A	[m^2]	0.20 ~ 0.40	3	3	3	0.35	*1.09E-13*	*2.08*
I	[m^4]	2.00E-03 ~ 4.00E-03	3	3	3	3.50E-03	*2.08*	*2.08*
L	[m]	0.10 ~ 0.20	3	3	3	0.175	*2.04*	*6.12*
F_{size}	[kN]	10.00 ~20.00	3	3	3	17.50	*1.33E-13*	*1.45E-13*
$F_{location,\ x}$	[m]	30.00 ~ 60.00	3	3	3	52.50	*1.63E-08*	*7.14*
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	9	3	3.50E+11	*6.11*	*8.67*
A	[m^2]	0.20 ~ 0.40	3			0.35		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	81	3	3.50E+11	*7.93*	*7.97*
A	[m^2]	0.20 ~ 0.40	3			0.35		
I	[m^4]	2.00E-03 ~ 4.00E-03	3			3.50E-03		
L	[m]	0.10 ~ 0.20	3			0.175		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	729	3	3.50E+11	*6.73*	*16.28*
A	[m^2]	0.20 ~ 0.40	3			0.35		
I	[m^4]	2.00E-03 ~ 4.00E-03	3			3.50E-03		
L	[m]	0.10 ~ 0.20	3			0.175		
F_{size}	[kN]	10.00 ~20.00	3			17.50		
$F_{location,x}$	[m]	30.00 ~ 60.00	3			52.50		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	9	3	3.50E+11	*6.11*	*8.67*
A	[m^2]	0.20 ~ 0.40	3			0.35		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	7	49	3	3.50E+11	*0.23*	*0.32*
A	[m^2]	0.20 ~ 0.40	7			0.35		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	19	361	3	3.50E+11	*7.20E-03*	*1.10E-02*
A	[m^2]	0.20 ~ 0.40	19			0.35		
L	[m]	0.10 ~ 0.20	19	19	1	0.175	*0.16*	*2.38*
L	[m]	0.10 ~ 0.20	19	19	2	0.175	*7.50E-02*	*2.06*
L	[m]	0.10 ~ 0.20	19	19	3	0.175	*2.50E-02*	*8.00E-02*
L	[m]	0.10 ~ 0.20	19	19	4	0.175	*2.50E-02*	*7.60E-02*
L	[m]	0.10 ~ 0.20	19	19	5	0.175	*2.50E-02*	*7.60E-02*
L	[m]	0.10 ~ 0.20	3	3	3	0.15	*5.77E-05*	*5.00E-04*
L	[m]	0.10 ~ 0.20	3	3	3	0.16	*1.56*	*4.98*
L	[m]	0.10 ~ 0.20	3	3	3	0.175	*2.04*	*6.12*
L	[m]	0.10 ~ 0.20	3	3	3	0.19	*1.11*	*3.14*
L	[m]	0.10 ~ 0.20	3	3	3	0.20	*2.82E-10*	*4.40E-03*

Table 4.5: Results of POD-RBF approximation on the case 1 model for varying different parameters, varying number of interpolation points, varying K-value, and varying test values.

- The last section shows the errors for different test values. The table shows that the closer the test value is to an interpolation point, the smaller the relative error is. On the interpolation point itself, the error is near zero.

Application on case 2

The difference in this case compared to case 1 is the element type and the way the stiffness matrices are retrieved. Requirement A states that software within ASML must be used for this, which made it important to test whether POD-RBF approximation could be applied using only ANSYS and MATLAB. Just like with the MMMM, this is possible. However, unlike the MMMM, this technique is not sensitive to rounding of the stiffness matrix, so having enough significant digits is less important. The ANSYS Python API was used again, this time with a slightly different approach as shown in Appendix D.2, after which the POD-RBF approximation was applied as described in Appendix C.2.

For this case, a similar table was created as Table 4.5. This table and how this table was set up can be found in Appendix E.2. Some same conclusions are drawn from this application test as were made for the application on case 1. Both linear and non-linear parameters can again be approximated with fairly good accuracy. Also more varied parameters lead to larger errors, more interpolation points reduce the error, and the error is smaller near the interpolation point.

What does stand out compared to case 1 is that adding more interpolation points for $F_{location,x}$

still results in a relatively large error. This can be explained by the way POD-RBF approximation works. The interpolation is done between the displacement values for each DOF. For parameters like E, t, L_x, L_y, and F_{size}, the relative relation between the parameters does not change much. For example, with a higher stiffness in the shell structure, the displacement of each DOF simply becomes larger or smaller by a certain factor. This is different for the location of the force. In that case, the displacement of the DOFs changes completely in relation to each other, as can also be seen in Figure 4.11. This is the reason why POD-RBF approximation does not work very well for a changing force location.

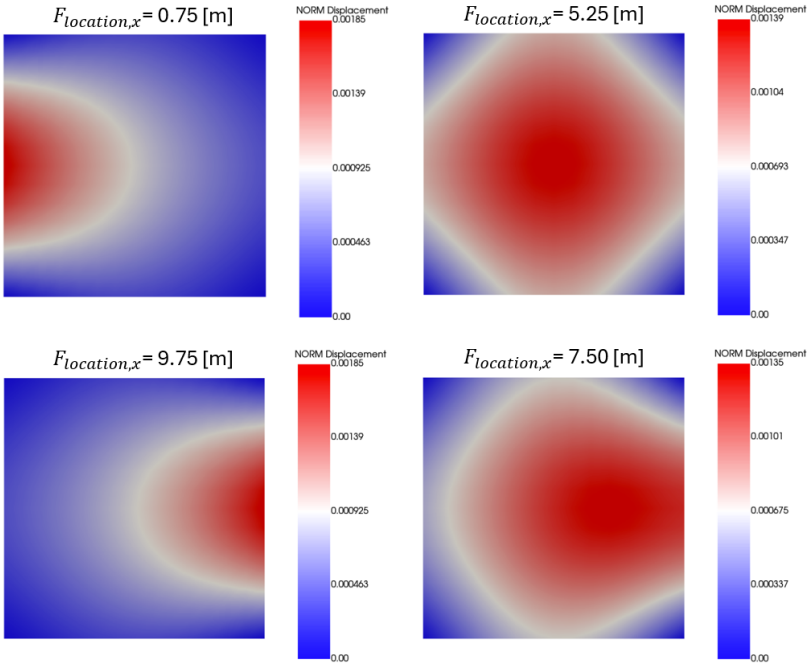

Figure 4.11: Displacements for different locations of the applied force.

In the context of requirement B, some tests were also carried out with a much larger number of elements and therefore more DOFs. This increases the computation time for each interpolation point, and thus the total training time. However, the increase in computation time for calculating a new parameter set is negligible. It still only requires a few matrix multiplications (see Subsection 2.3.2), which are computationally fast. Meeting requirement B is therefore not an issue.

For both case 1 and case 2, it was found that the number of interpolation points increases significantly with the number of parameters. In addition, the error also increases significantly when more parameters are used, meaning that even more interpolation points are needed to keep the error low. Including many different parameters can therefore make it difficult to meet requirement 1, because too many interpolation points have to be calculated. The section below explores whether this can be done in a smarter way.

4.3.2. Proposed adaptations

The current way of setting up the POD-RBF approximation is to calculate each interpolation point directly. Because each calculation takes a certain amount of time, the total time needed to set up the POD-RBF approximation is equal to the time per calculation multiplied by the number of interpolation points. It is proposed to use a smarter training approach by combining the POD-RBF approximation with substructuring.

The literature shows that changes in a small part of the structure can be calculated more quickly by using substructuring. Therefore, it is proposed to calculate all interpolation points that only involve a small change compared to a previously calculated structure using substructuring. This is illustrated in Figure 4.12.

Substructuring gives exact results when applied to static calculations, so in theory this approach should work. However, substructuring has not yet been applied to these practical cases, so it is still unclear how much faster this training method can be. In Section 4.4, substructuring is tested, and more can be said about this afterwards.

A similar approach can be applied to global changes in the structure for parameters that appear consistently throughout the stiffness matrix, but in that case using the MMMM technique instead of substructuring.

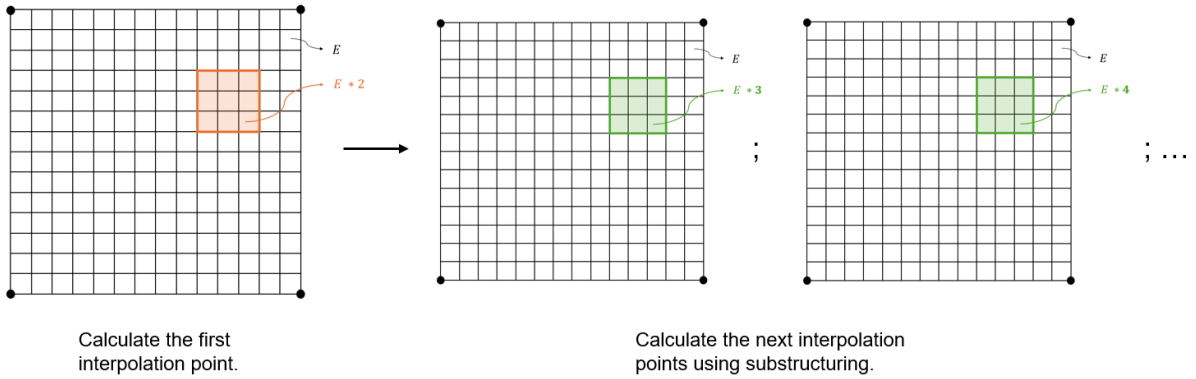

Figure 4.12: Improved training approach by combining POD-RBF with substructuring.

In the context of requirement 4, an adaptation is also being explored. The POD-RBF approximation ultimately provides a displacement field, while the substructuring approach works by coupling reduced stiffness matrices at the interface DOFs. For static problems, this is done using Guyan reduction, which was introduced in Subsection 2.2.1.

From the displacement vector approximated by the POD-RBF method, the interface displacements $\mathbf{u}_2$ from Equation (2.10) (shown again below) can be extracted by selecting only the interface DOFs. The corresponding force vector $\mathbf{f}_2$ is also known, as it contains the forces applied at the interface DOFs.

$$\overline{\mathbf{K}}_G \mathbf{u}_2 = \mathbf{f}_2$$

From this, $\overline{\mathbf{K}}_G$ can be estimated. However, since it only involves a single displacement force pair, the system is underdetermined and can only be directly calculated as a diagonal matrix. This means that off diagonal terms, which represent the coupling between interface DOFs, are not included. As a result, the reconstructed $\overline{\mathbf{K}}_G$ only reflects the local stiffness at each DOF and not the interaction between them.

If there is strong coupling between the interface DOFs in place, then the POD-RBF approximation does not provide a good fit within the substructuring context. However, if the coupling is minimal, the POD-RBF approximation can offer a reasonable estimate.

4.3.3. Evaluation against the requirements

In Table 4.6 each requirement is briefly evaluated. It is concluded that all requirements can be met, except for requirement 4, which is only partly fulfilled. The fact that the method can

only be used when the coupling between the interface DOFs is minimal limits its applicability. However, since all other requirements are met, the method is still usable in ASML's design process.

In the previous section, where the MMMM was evaluated, requirement 4 (which is the same in this case) was not examined. However, since the MMMM also only produces new displacement vectors, the same conclusion could be drawn here.

Technique	Requirement		Evaluation	
POD-RBF	1	The setup time must remain within two weeks.	This requirement is satisfied because the number of interpolation points can be chosen freely and can therefore stay within this time limit. It may be challenging, however, to also meet requirement 3 at the same time if many different parameters need to be changed. In this situation, it is suggested to combine the technique with substructuring or the MMMM.	✓
	2	Understanding of whether varying force locations can be handled.	This requirement is now fulfilled. The tests above show that it is difficult to interpolate between different force locations because the displacement field changes completely with a different force location. As a result, interpolation is not well supported in this case. However, varying the force magnitude is supported.	✓
	3	The displacement error at interface DOFs must remain below 5%.	The tests show that this level of accuracy can be reached if enough interpolation points are used. This requirement is therefore satisfied.	✓
	4	The technique must be compatible with substructuring approaches.	It turns out that only a reduced diagonal matrix can be retrieved. Therefore, only a rough estimation can be made, and only if the coupling between the different interface DOFs is minimal. This requirement is therefore partly fulfilled.	-
General	A	Implementation must rely solely on existing ASML software tools.	This is feasible, based on the test results.	✓
	B	Recalculation time for new parameter sets must remain below 20 minutes.	This is achievable according to the test results, even for many degrees of freedom. This is achievable according to the test results, even for many degrees of freedom.	✓

Table 4.6: Evaluation of POD-RBF approximation against requirements

4.4. MLS

The requirements in Table 4.7 were defined in Section 3.4 to successfully implement MLS into the design process. This section tests and verifies whether MLS can meet these requirements.

Technique	Requirement	
MLS	1	Understanding of the number of substructures that can be varied effectively.
General	A	Implementation must rely solely on existing ASML software tools.
	B	Recalculation time for new parameter sets must remain below 20 minutes.

Table 4.7: Requirements that MLS must meet for successful implementation in ASML's design process

4.4.1. Current technique

Requirement 1 was defined to create clarity about how much substructures can be changed before directly generating a new ROM becomes faster. To test this, a system with a large number of DOFs must be used so that computation times can be compared. For small systems, the computation times will be very low, which makes a proper comparison difficult. It is also important to work with the same software used in the actual process to make a valid comparison. For these reasons, MLS is directly applied to case 2. In this case, the number of DOFs can easily be increased by adding more elements, and the stiffness matrices are generated directly in ANSYS.

Application on case 2

The test case is shown in Figure 4.13, and the steps of the test are explained below. The corresponding MATLAB code is provided in Appendix C.3.

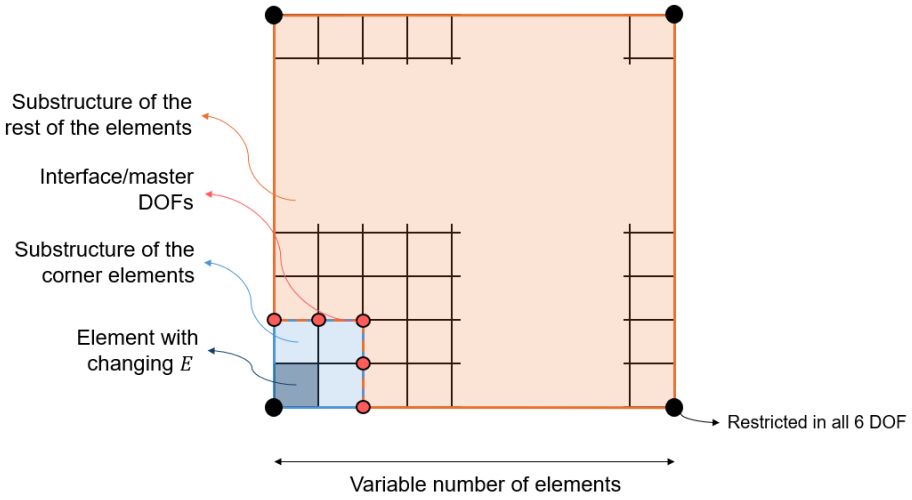

Figure 4.13: Case 2 model for MLS.

1. **ANSYS: Set up structure and export stiffness matrix and coupling vectors.** The first step is to set up case 2 in ANSYS with the desired number of DOFs. Then, the stiffness matrix and a number of coupling vectors need to be extracted from this model. ANSYS can only export the stiffness matrix in *solver ordering*. This means ANSYS does not provide the stiffness matrix in the order that matches the node or element numbers (*user ordering*), but instead reorders the entire matrix to allow for faster solving. To later select the interface DOFs in this stiffness matrix, the relation between the node numbers of these interface DOFs and their position in the stiffness matrix must be retrieved. Extracting different coupling vectors makes it possible to obtain this relation. The ANSYS APDL snippet used to extract the stiffness matrix and these coupling vectors is shown in Appendix G.1. After extraction, it is not necessary to solve the system in ANSYS, as this will later be done using substructuring in MATLAB.
2. **MATLAB: Import stiffness matrices and coupling vectors and generate ROMs for Substructure 1 and 2.** In the second step, all exported matrices from ANSYS are imported into MATLAB. Two ROMs are then generated using Guyan reduction. One ROM is for Substructure 1, which includes the four elements in the bottom-left corner. The other is for Substructure 2, which includes the remaining elements. After this, the reduced total system can be solved.
3. **ANSYS: Modify structure and export stiffness matrix.** The next step is to make a change in the elasticity modulus of the corner element in Substructure 1 in the ANSYS model. The new stiffness matrix needs to be exported. Coupling vectors do not need to be extracted again since the ordering has not changed.
4. **MATLAB: Import stiffness matrix and regenerate ROM for Substructure 1.** Finally, this new stiffness matrix must be imported into MATLAB, and only the ROM for Substructure 1 needs to be regenerated, as the ROM for Substructure 2 remains the same. The total reduced system can now be solved again.

The results for solving directly in ANSYS or MATLAB and for using substructuring match exactly, as expected based on the literature. However, the focus for Requirement 1 is on the

computation times. Although computation times can of course vary between computers, a fair comparison is possible because all tests were performed on the same computer. For these tests, an HP ZBook Fury 16 G10 Mobile Workstation PC was used.

Figure 4.14 shows the computation times in ANSYS, where the crosses in the figure represent the collected data. The time needed to set up and export the stiffness matrix and coupling vectors is compared to the time required to solve the system directly in ANSYS. The difference in time is caused by the fact that, in the direct solution, ANSYS also needs to calculate $\mathbf{u}$ in addition to setting up the system. It is important to mention that these calculations were done using only one processor core. This is because ANSYS does not allow exporting the stiffness matrix when the computation is distributed over multiple cores [33].

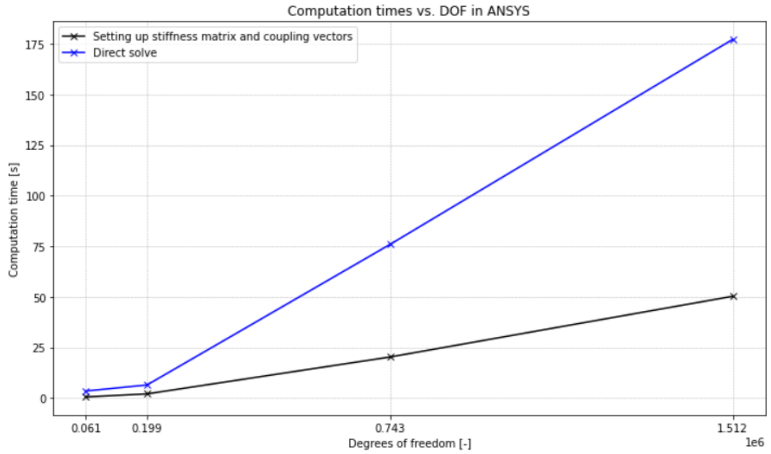

Figure 4.14: Computation times vs. DOFs in ANSYS

The computation times of the different steps in MATLAB are shown in Figure 4.15. The red line shows how much time it takes to regenerate only Substructure 1 and then solve the system (part of step 4 above). It shows that this is much faster than solving the full system directly in MATLAB, shown by the orange line. This confirms that substructuring can indeed provide significant time savings when only one substructure is modified.

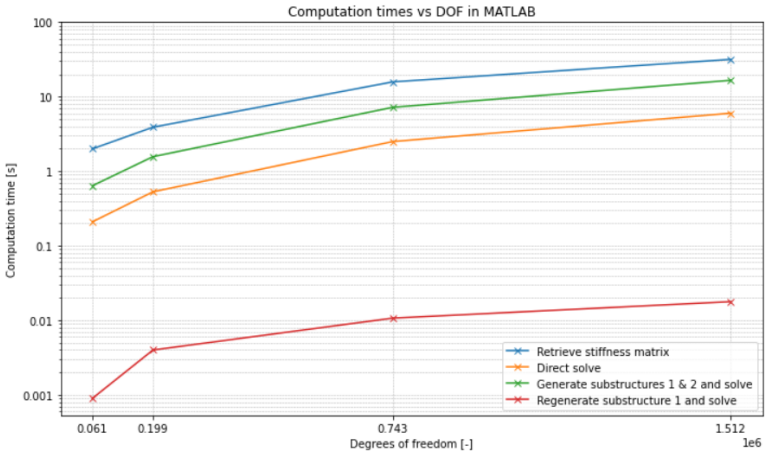

Figure 4.15: Computation times vs. DOFs in MATLAB

Although the relative time savings may seem significant at first, in practice they are much lower due to the time it takes to load a stiffness matrix, shown by the blue line. When using this method, the new stiffness matrix must be imported each time, and this turns out to take

a lot of time. In ANSYS, it is only possible to export the stiffness matrix in Matrix Market Format with 14 significant digits, which is read in MATLAB using the existing `mmread` function. Therefore, it is not possible to speed up this step. However, ANSYS has indicated that in the newest version (ANSYS 2026 R1), it might become possible to export the matrices with fewer significant digits, which would make importing into MATLAB faster.

An important result shown in the figure is that the time needed to set up the substructures is very limited. With Guyan reduction, creating a ROM for a substructure with 1,512,000 DOFs takes just over 6 seconds. This supports what was also found in the interviews in Section 3.2, namely that generating ROMs for static analysis is quite fast. Of course, some modules within the ASML machine have more than 1,512,000 DOFs. Even in those cases, the absolute possible time savings with a new design approach for static analysis will remain limited, because setting up a substructure with Guyan is already very fast.

When comparing the two figures side by side, it stands out that direct solving in MATLAB is much faster than direct solving in ANSYS. For 1,512,000 DOFs, direct solving in MATLAB takes about 6 seconds, while solving the entire system in ANSYS (excluding matrix export) takes around 125 seconds. It is expected that part of this difference is because ANSYS uses only 1 core, while MATLAB uses up to 8 cores on the computer used for testing [34]. To check this, the calculations in ANSYS were also performed again using 4 and 8 cores. The results are shown in Figure 4.16.

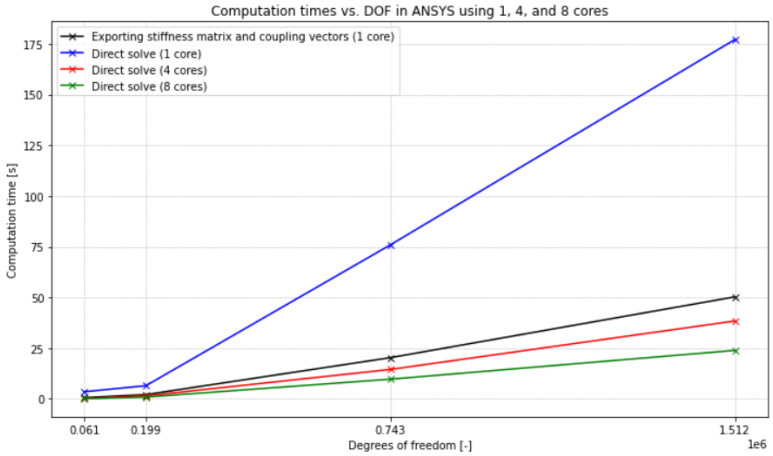

Figure 4.16: Computation times vs. DOFs in ANSYS using 1, 4, and 8 cores.

It can be seen that the computation times are now much closer to those of MATLAB, but the difference is still significant. After contacting ANSYS, it appears that the difference is likely caused by certain calculation checks performed within ANSYS that MATLAB may not do. Investigating this difference further falls outside the scope of this research.

It can also be seen that solving the full system in ANSYS with 4 cores is already faster than just exporting the matrices. Since most modern computers have 4 cores or more, it can be concluded that methods which require exporting the stiffness matrix and coupling vectors are better solved directly in ANSYS.

4.4.2. Evaluation against the requirements

This section explains that using MLS in MATLAB is not worthwhile, because retrieving matrices in ANSYS, which only works with one core, takes more time than solving the full static system using four cores. As a result, Requirement 1 states that the number of substructures that can

effectively be changed is zero, if Requirement A must also be satisfied.

4.5. Woodbury Identity

As described in Subsection 2.4.2, it is also important for the Woodbury Identity to retrieve the new stiffness matrix. Based on the previous section, it can therefore be concluded that the number of elements that can effectively be varied is 0 (Requirement 1 in Table 4.8) if Requirement A must also be satisfied.

Technique	Requirement	
Woodbury	**1**	Understanding of the number of element changes that can be handled effectively.
General	**A**	Implementation must rely solely on existing ASML software tools.
	B	Recalculation time for new parameter sets must remain below 20 minutes.

Table 4.8: Requirements that Woodbury Identity must meet for successful implementation in ASML's design process

4.6. Findings and final objectives

This chapter originated from SQ3: '*Do the existing techniques, either in their existing form or adapted, meet the requirements for implementation within ASML's design process?*'. This question was answered using tests in MATLAB and ANSYS. Below, a short explanation is given for each technique about why it does or does not meet the requirements.

- **MMMM:** This technique does not meet the requirement that states it must support changes in localized regions of the structure. It is not possible to make local changes without causing significant median and maximum errors, or without the method starting to resemble a weakened version of substructuring.
- **POD-RBF approximation:** This technique meets all requirements. A limitation is that in the context of substructuring it is only possible to retrieve the reduced stiffness matrix as a diagonal matrix. This means that an accurate estimate can only be made when the coupling between the interface DOFs is minimal.
- **MLS:** It turns out that when the stiffness matrix must be retrieved from ANSYS, it is limited to the use of one processor core. Solving the full system using four cores is already faster than exporting the stiffness matrix using one core. Therefore, it is not possible to speed up the process by implementing this method.
- **Woodbury Identity:** For this method, it is also necessary to retrieve the new stiffness matrices for each new parameter set. As a result, this method is not suitable for implementation.

Based on the conclusion for MLS, the MMMM is also ruled out again as it also requires new stiffness matrices for each new parameter set. POD-RBF meets all requirements, does not require new stiffness matrices for every parameter set, and is therefore the only technique that could be successfully implemented.

An important conclusion from the tests for MLS is that generating a ROM with Guyan is relatively fast. For example, a ROM of a substructure with more than 1.5 million DOFs was generated in about 6 seconds. This supports what was found in the interviews in Chapter 3, that since this thesis focuses only on static analysis, the potential to significantly speed up the process is more limited. Knowing that the modules within ASML will not have significantly more DOFs than 1.5 million, it can be concluded that the absolute time savings from skipping

the reduction step will be minimal.

This does not mean that no improvements can be made. The only technique that meets the requirements, POD-RBF approximation, is the only one that also skips the step of modifying the FEM model in addition to skipping the reduction step (see Figure 3.4). This makes it possible to calculate multiple parameter sets faster and still offers a speed-up in absolute time. Moreover, POD-RBF approximation also has the advantage that, once the model is trained, no expensive ANSYS license is needed for the user.

Now that all techniques from the literature have been verified by developing them (step 3 of the DSRM), demonstrating them on specific cases (step 4 of the DSRM), and evaluating the results (step 5 of the DSRM), one final iteration can be made back to step 2 of the DSRM. In Table 4.9 below, the finalized objectives are presented, along with an explanation of how these were defined for each category.

Category	Objective	Explanation
Parameters to vary	For all the objectives in this category, it is important to consider during the training setup whether this parameter change may potentially need to be made later. • Multiple parameters can be varied at the same time. • Both structural parameters with linear and non-linear effects on the system, as well as the magnitude of the applied force, can be adjusted. • Parametric changes can be applied to both large parts of the structure and to local areas.	Based on the literature and now also the test results, it appears that the following objectives are achievable using POD-RBF approximation, which is the only technique that meets the requirements. One objective not included is variation in force location, because interpolating this with POD-RBF approximation does not yield good results. However, keep in mind that if the data already includes these locations, it can still produce good results (as no interpolation is needed in that case).
Accuracy	• The displacement vector on the interface DOFs must have a maximum deviation of 5% compared to the direct solution	This has not been changed. The tests show that this is feasible for POD-RBF approximation with enough interpolation points.
Time reduction	• Setting up the technique should not take longer than 2 weeks. • Running a new parameter set should take no more than 1 minute.	• This has not been changed. The tests show that it should be feasible to set up the model within two weeks. • This objective has been lowered from 20 minutes to 1 minute because the strength of the POD-RBF technique lies in the fact that it not only skips the reduction step but also bypasses the need to modify the FEM model. As a result, it becomes possible to evaluate different parameter sets very quickly.
Integration capability within ASML	• The technique only need to be compatible with ASML's substructuring approach for modules where the coupling between interface DOFs is minimal. • The technique should only use software that is already used at ASML.	• POD-RBF can only be implemented within the substructuring approach if the coupling between the interface DOFs is minimal. • This has not been changed.

Table 4.9: Final objectives for implementation of techniques enabling parametric changes within ROMs to speed up ASML's design process.

Since now the final objectives have been defined, it is possible to investigate how the only technique that meets all the requirements, POD-RBF approximation, can actually be applied within ASML. Although the tests in this chapter were carried out on generic models, they do not yet provide clear insight into the applicability within ASML's specific context. Therefore, the next chapter presents a case study based on a realistic ASML application. By testing this case against the defined objectives, it can be evaluated whether the expected benefits of the POD-RBF technique can also be achieved in a realistic setting. This is essential for answering the main research question.

5

Case study on factory floor

In Chapter 3, requirements were defined that the different techniques must meet to be successfully implemented in ASML's design process. In Chapter 4, these requirements were tested, and it was found that only the POD-RBF approximation meets the defined requirements. To demonstrate the practical value of the POD-RBF approximation at ASML, this chapter presents a case study based on an example from within the company. The findings from this case study are then used to answer SQ4: '*How do the techniques that meet the implementation requirements perform when applied to a designed ASML case study*?'.

This chapter starts by introducing the case study in Section 5.1. The following subchapters are structured according to steps 3, 4, and 5 of the DSRM (development, demonstration, and evaluation). In Section 5.2 step 3 of the DSRM is addressed by setting up the case study model, which is then validated in Section 5.3. Next, Section 5.4 covers step 4 of the DSRM by demonstrating which insight can be retrieved with the developed model. Finally, in Section 5.5 step 5 of the DSRM is carried out by evaluating how well the POD-RBF approximation performs in this case. This evaluation is based both on the final objectives from Section 4.6 and on feedback from a focus group.

Note that in this public version, absolute numbers have been replaced with an X for ASML security reasons.

5.1. Introduction

As explained in Section 3.2, substructuring is used when performing system-level calculations for the ASML machine. In this approach, the machine is divided into around twenty modules. The underlying factory floor is one of these modules. This case study focuses on this factory floor. First, some background information about the factory floor is provided in Subsection 5.1.1, followed by a description of the problem that is experienced with this module in Subsection 5.1.2.

5.1.1. Background

All factories look very similar but differ slightly from one another. For ASML security reasons, this study does not depict a real factory floor. Instead, a fictitious factory floor was created. This fictitious, standardized schematic version of a factory is shown in Figure 5.1. The figure shows a factory that consists of three floors built with columns, beams, and floors. The ground floor is used for utilities, the first floor is for the support cabinets of the ASML machines, and

the second floor is where the ASML machines, shown in orange, are placed. The roof is left out here because the focus is only on the relationship between the machine and the factory, and the roof has minimal influence on this. The most important parameters that can vary in the fictitious factory are also shown in the figure.

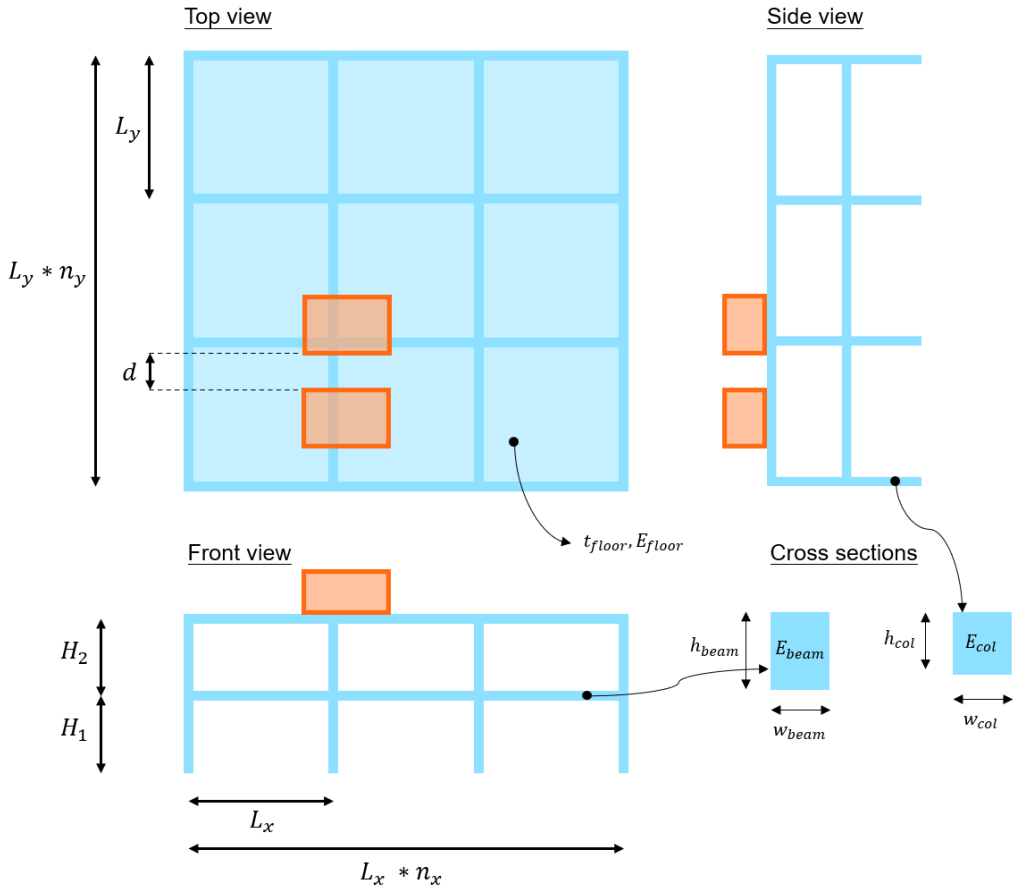

Figure 5.1: Schematic representation of a standardized fictitious factory including an ASML machine, showing the most important structural parameters.

The fact that most factories have such a standardized structure is not because ASML is directly responsible for building the machines, but because of the requirements ASML sets for the factories that customers need to build. There are only limited options, which are also economically feasible, to construct a factory that meets these requirements. As a result, most factories only vary in aspects like column spacing or floor thickness. The most important requirements that ASML sets for the factory are:

1. $L_x, L_y > \mathrm{X}\,[\mathrm{m}]$
2. $H_1 < \mathrm{X}\,[\mathrm{m}]$
3. $H_2 < \mathrm{X}\,[\mathrm{m}]$
4. $d = d_{\text{small}} = \mathrm{X}\,[\mathrm{m}]$ or $d = d_{\text{wide}} = \mathrm{X}\,[\mathrm{m}]$
5. $K_{\text{legs}} > \mathrm{X}\,[\mathrm{N/m^2}]$

The first requirement comes from the size of the support cabinets of the ASML machines, which are placed on the ground floor and the first floor. The second and third requirements are based on the maximum cable length from these cabinets to the machine on the second floor. The fourth requirement states that there are two possible distances between machines:

the small and the wide corridor. These distances are set to ensure enough space around the machine for maintenance. It is not of importance for this research where each distance is used.

The fifth and final requirement sets a minimum stiffness under the legs of the machine. This requirement is based on two things. First, it comes from the rotation limits of the machine. To function properly, the machine is not allowed to rotate more than X $[\mathrm{urad}]$ in the x- and y-direction, and the maximum out-of-plane torsion must not exceed X $[\mathrm{um}]$. Second, there is a dynamic requirement where the frequency of the moving parts in the machine must not match the natural frequency of the factory floor.

5.1.2. Problem statement

When a machine is placed, it is leveled using special feet to make sure it stays within the rotation limits specified above. It often happens that after a few months, when the first machine has already been operating for some time, a second machine is placed. The placement team currently has limited insight into how the addition of a second machine affects the first one. As ASML's machines continue to grow in size and weight, there is growing concern that even when the current stiffness requirements are met, the first machine may still experience excessive rotation in certain scenarios. This situation is shown schematically in Figure 5.2. In such a case, the first machine would need to be re-leveled, which means it would be out of operation for a week. This leads to high costs for the customer and must be avoided at all costs.

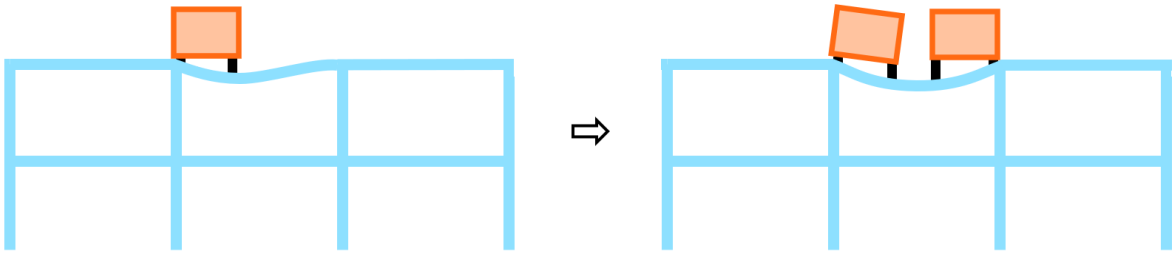

Figure 5.2: Schematization of the first machine rotating because of the second machine.

Better insight could of course be gained using ANSYS models, but this team does not have ANSYS licenses, and their knowledge of the software is also limited. For questions about this, the internal FEM team at ASML has to be contacted each time. There is a demand from this team for an accessible method that allows the placement team to understand how the machines interact without always relying on the FEM team.

5.2. Setting up the model

In this section, the case study model is set up using the POD-RBF approximation. In Subsection 5.2.1, the motivation for using the POD-RBF approximation is explained, including the added value it can offer in relation to the problem described in the previous section. Next, the first step in setting up the case study model is presented in Subsection 5.2.2, which involves creating a parametric FEM model. Finally, Subsection 5.2.3 explains how this model can be used efficiently to generate the required data for training the POD-RBF approximation.

5.2.1. Motivation for using POD-RBF approximation

In Chapter 4 it was established that the POD-RBF approximation is the only technique that meets all the requirements to be successfully implemented in ASML's design process. This section further explains why this technique is especially suitable for this specific case study.

As discussed in Subsection 5.1.1 almost all factories have a standardized structure, where variation is limited to a small number of structural parameters. This results in a limited and well-defined parameter domain in which the effect of these parameters on the structural behavior can be simulated. This makes it possible to create a training dataset using the POD-RBF approximation, which can then be used to accurately interpolate to new parameter combinations within this domain, without needing additional FEM calculations.

POD-RBF approximation offers the advantage that after the training phase, there is no longer any need to use ANSYS or other FEM software. The calculations can be fully performed within a simple tool, such as Excel or MATLAB. This means that team members without FEM knowledge or an ANSYS license can independently perform analyses. In this way, it directly addresses the problem described in Subsection 5.1.2.

There are two important points to consider when using this technique. The first is that it is difficult to interpolate between different machine locations with the POD-RBF approximation. This requires making specific choices for the machine locations during the training procedure. The second point is that with POD-RBF approximation the number of required interpolation points for accurate results increases exponentially with the number of parameters to be varied. In Figure 5.1, the 14 most important varying structural parameters are shown. As explained in Section 4.3, with this many parameters, it might be that more than 10 interpolation points per parameter are needed for accurate results. This would bring the total number of interpolation points to over 10^{14}, even without varying the placement of the ASML machine. Such a number of interpolation points is impossible to calculate in terms of time and storage capacity. Therefore, it is important to carefully decide which parameters are essential to vary in this case study.

5.2.2. Parametric FEM model setup

To generate the training data needed for the POD-RBF approximation, the FEM model must be set up and solved for each parameter set. It is not time-efficient to manually create a new FEM model for every interpolation point. Therefore, a parametric FEM model has been developed.

The developed parametric model can modify all 14 structural parameters shown in Figure 5.1. In addition, the machine placement is also parameterized. The structural parameters are parameterized using the parametrization functions in ANSYS DesignModeler. The machine placement is parameterized using an APDL Command script, which is added to the model in ANSYS Mechanical. This APDL script is shown in Appendix G.2.

Due to the large number of elements, the entire input APDL code file for this parametric FEM model consists of approximately 50,000 lines and is therefore not included directly in the appendix. The APDL input file can be downloaded by double-clicking the pushpin below. Note that to access the files a PDF viewer is needed that supports embedded links (i.e. Adobe Acrobat).

APDL input file: 

In reality, the legs of the ASML machine are not placed directly on the factory floor but on two *pedestals*. These are blocks that distribute the weight of the machine over the floor. The relation between the machine, the legs, and the pedestals is shown in Figure 5.3. It is important to mention that these two pedestals do not carry the full weight of the machine, but only the central, heaviest part. In reality, there are also pedestals located on the left and right sides of the machine outline as shown in the figure. However, since these pedestals carry significantly less weight, their influence on the displacements is minimal. Therefore, they are not included

in this study. When referring to the ASML machine in the remainder of this study, it specifically refers to this central module.

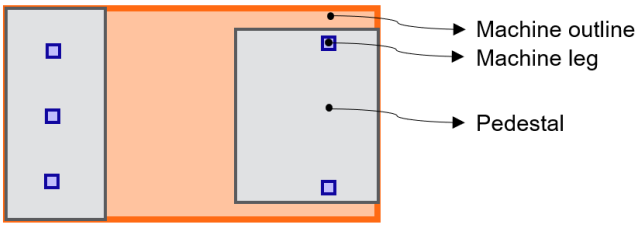

Figure 5.3: Machine legs positioned on pedestals that distribute the load onto the factory floor.

Unfortunately, ANSYS does not support parametric modeling of the position of the pedestals. However, it is possible to parametrically define the position of a surface load. To avoid manually assigning the positions of the pedestals for thousands of interpolation points, the pedestals were excluded from the model and replaced with a surface load. It is assumed that the force exerted by the legs on the pedestals is evenly distributed across the surface of the pedestal.

The pedestals are made of concrete and, due to their thickness, often have a higher stiffness than the floor itself. As a result, excluding the pedestals may lead to higher displacement values than would occur in reality. This also means that the stiffness and rotation in the developed model may be overestimated. Therefore, if the model meets the stiffness and rotation requirements, these requirements are highly likely to also be met in the actual physical structure.

An example of the total floor displacement and the rest of the fictitious factory is shown in Figure 5.4. In the figure, the pedestal footprints can be seen over which the forces from the machine legs are equally distributed.

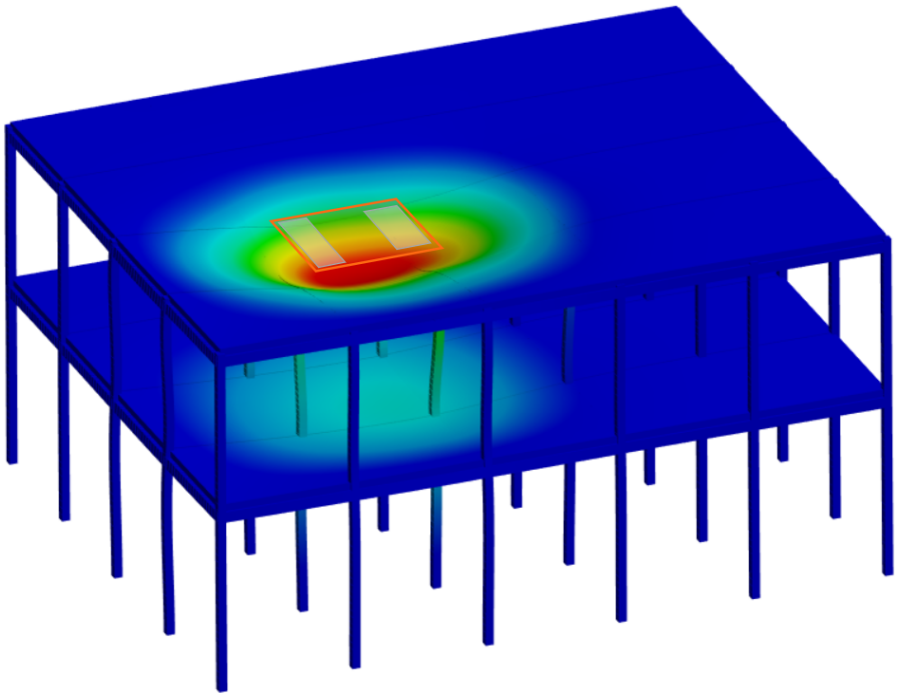

Figure 5.4: Total deformation of the parametric FEM model. Geometrical parameters: $H_1 = \mathrm{X}\,[\mathrm{m}]$, $H_2 = \mathrm{X}\,[\mathrm{m}]$, $L_x = L_y = \mathrm{X}\,[\mathrm{m}]$, $h_{col} = w_{col} = w_{beam} = t_{floor} = \mathrm{X}\,[\mathrm{m}]$, $h_{beam} = \mathrm{X}\,[\mathrm{m}]$, $n_x = \mathrm{X}\,[-]$, and $n_y = \mathrm{X}\,[-]$

5.2.3. Generation of the training data

As explained in the previous subsection, the first step is to create a parametric FEM model. In this subsection, it is explained how the training data for POD-RBF approximation is generated using this parametric FEM model. First, the parameters that will be varied are described. Then, the variation ranges for these parameters are defined, and the fixed values for the parameters that remain constant are specified. Finally, it is explained how the training data can be generated as efficiently as possible using the parametric FEM model.

Selection of parameters to vary

As discussed in Subsection 5.2.1, the number of required interpolation points increases exponentially with the number of parameters that are varied. Therefore, it is important to carefully select which parameters are essential to vary in this case study. Below, the selected parameter to be varied are displayed. In the brackets, the corresponding parameter names from Figure 5.1 are shown for each selected parameter. It can be seen that columns and beams are combined. This combination is called the *frame*.

1. Floor thickness (t_{floor})
2. Frame width ($w_{frame} = h_{beam} = w_{beam} = h_{col} = w_{col}$)
3. Distance between columns ($L = L_x = L_y$)
4. Column height on the second floor (H_2)
5. Machine location

The first four parameters are structural parameters. Three considerations led to the selection of these parameters.

The first consideration is the extent to which parameters vary between different factory floors. If certain parameters only change in a few factory floors, they are less relevant to include in this model. This information was obtained in consultation with the placement and FEM teams. For example, it was found that the distances between the columns in the x- and y-direction are almost always equal to each another.

The second consideration is the influence of a parameter on the displacement differences under the legs of the machine. In the end, it is important to check whether the rotation requirements of the machine are exceeded. Therefore, if a parameter has little to no effect on increasing or decreasing the relative displacements between different machine legs, it is less relevant to include. To determine this influence, some tests were performed with the parametric FEM model using different parameter variations. For example, it was found that the heights of the columns on the first floor has limited influence on this.

The third consideration is the placement team's preference to adjust a single parameter in a different part of the factory, rather than having to adjust two parameters in the same area. For this reason, for example, it was decided not to vary both t_{floor} and E_{floor}, because the preference is to be able to vary something like H_2 instead.

The fifth and final parameter concerns the placement of the forces. For the problem described in Subsection 5.1.2, it is important to vary these locations. Note that no interpolation will be done between these locations (see Subsection 5.2.1). Displacements will only be calculated for these specific locations. Additionally, it is necessary to place a second machine next to each selected location.

Definition of parameter ranges and fixed values

It is now important to choose the ranges and the number of interpolation points for the variable structural parameters, and to set fixed values for the structural parameters that will stay constant. The machine locations also need to be selected.

Table 5.1 gives an overview of the parameter ranges and the fixed values of the structural elements (X inserted; not for public release). These values are chosen based on the wishes from the placement team within ASML.

Parameter	Unit	Start/fixed value	End value	# Interpolation points
Floor thickness (t_{floor})	[m]	X	X	5
Frame width (w_{frame})	[m]	X	X	5
Distance between columns (L)	[m]	X	X	7
Height of first floor (H_2)	[m]	X	X	2
Floor E-modulus (E_{floor})	[N/m^2]	X	-	-
Frame E-modulus (E_{frame})	[N/m^2]	X	-	-
Height of ground floor (H_1)	[m]	X	-	-
Number of spans in x-direction (n_x)	[-]	X	-	-
Number of spans in y-direction (n_y)	[-]	X	-	-

Table 5.1: Selected parameter ranges and fixed values for the structural elements of the fictitious factory floor model.

Between the parameters t_{floor} and w_{frame}, the POD RBF approximation is used for interpolation, which means any value can be chosen for these parameters in the model. For L and H_2, this is not the case, and only one of the predefined interpolation points can be selected in the model. This decision originates from Section 4.3, where it was found that when more parameters are varied, more interpolation points per parameter are needed. To limit the number of interpolation points while still providing the same level of accuracy, this approach was chosen. Another reason not to interpolate between values of L is that tests showed the displacement changes significantly, which makes it difficult for the POD-RBF approximation to maintain accuracy (see Section 4.3).

Figure 5.5 shows the selected machine locations for $L = \mathrm{X}\,[\mathrm{m}]$ and $L = \mathrm{X}\,[\mathrm{m}]$ (X inserted, not for public release). To keep the figure clear, the possible machine locations are shown as as grid, where on each grid point, a machine can be placed. A grid point represents the bottom left corner of the machine.

The machine locations were chosen with the idea that within a single *span*, the machine position can be varied with an accuracy of X. A span refers to a portal structure (a single blue square in the figure). Tests showed that when the machine is placed outside the first row of spans, the displacement fields are very similar as long as the machine is positioned at the same relative distance from the beams.

To address the problem in Subsection 5.1.2, it is also important to place a second machine next to the first one. Therefore, the machine location is placed next to this first span, so that at least one machine can be placed at a distance of d_{small}. Since displacements are linear, the displacements can then be added together.

In addition, the placement team is also interested in the displacement behavior at the edge of the factory floor. For this reason, machines are also placed up to X from the edge. In reality, machines will not be placed closer to the edge than this. Depending on the value of L, the total number of machine locations is between 52 and 133.

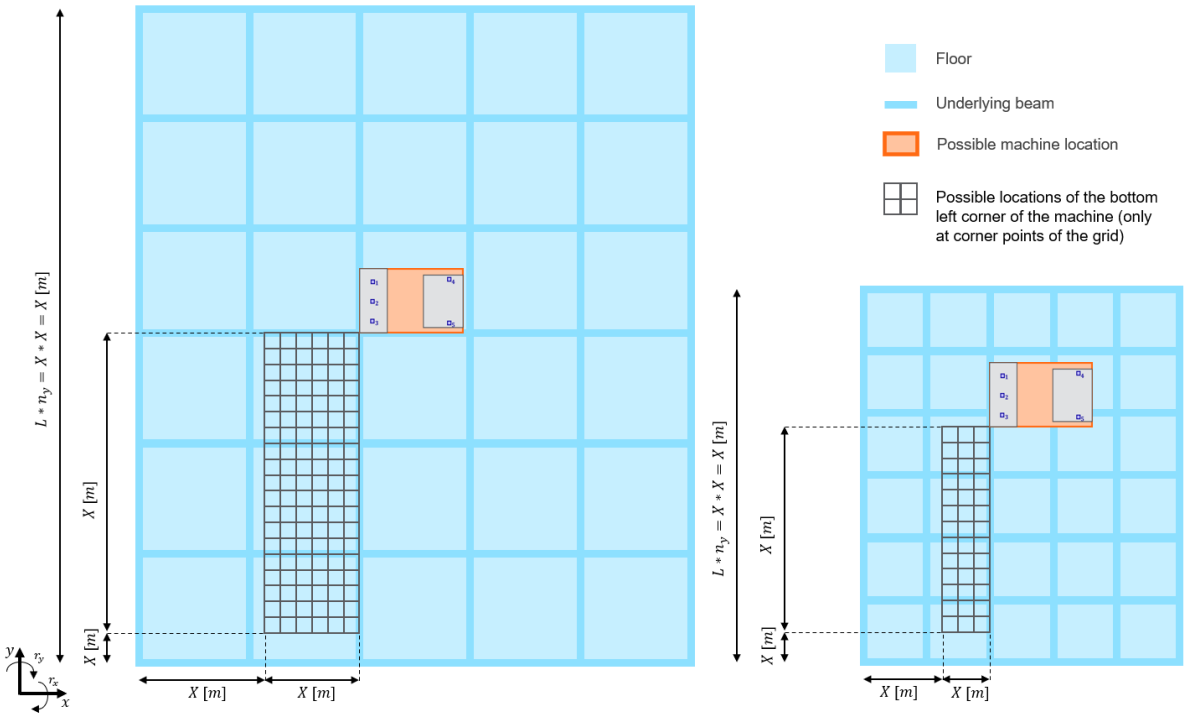

Figure 5.5: Selected machine positions used in the case study model.

Speeding up the generation of training data using load steps

If an average of 100 machine locations is assumed, the total number of interpolation points is $5 * 5 * 7 * 2 * 100 = 35,000$. The next step is to determine how to efficiently compute the displacements for all these interpolation points.

A mesh of X $[\mathrm{m}]$ is used in the parametric FEM model, which results in a number of DOFs between approximately 150,000 and 375,000, depending on L. Looking back at Section 4.4, calculating one interpolation point with 8 cores seems possible within a few seconds. Assuming 4 seconds per interpolation point, all interpolation points could be calculated in about 1.5 day $(35,000 * 4/(3600 * 24))$.

Unfortunately, calculating multiple systems sequentially in ANSYS is not this straightforward. In ANSYS Workbench, the 'Table of Design Points' is filled in under the 'Parameter Set' tab. For each row in this table with a different parameter combination, ANSYS performs the following steps: start ANSYS Mechanical, generate the mesh, start the solver, set up and execute the calculation, close the solver, and finally close ANSYS Mechanical. Although the actual calculation only takes a few seconds, all these additional steps result in a total time of almost two minutes per interpolation point. Consequently, calculating all interpolation points would take about 50 days $(43,218 * 120/(3600 * 24))$.

After some research into ways to speed up the process, it was found that using load steps can make this significantly faster. With the load step function in ANSYS Mechanical, different loads can be calculated within a single computation. This means that for the 52~133 interpolation points for the machine locations, only the calculation itself needs to be performed each time, without restarting and closing ANSYS Mechanical, the solver, or generating the mesh. This has been integrated into the ANSYS model using the ANSYS APDL code shown in Appendix G.2. As a result, the number of times these additional steps need to be performed is reduced from approximately 35000 to $5 * 5 * 7 * 2 = 350$. One interpolation point now takes longer, because

52~133 load steps need to be calculated per point, but still the total training time has been reduced to 4 days.

5.2.4. Completion of the model

In the way described above, all training data is collected. From this data, the displacement matrix $\mathbf{U}$ can now be constructed. This allows the POD-RBF approximation to be applied. A tool has been developed in which the input consists of the variables w_{frame}, t_{floor}, L, H_2, and the machine locations of two machines. The output includes the displacements, stiffnesses, and rotations of both machines, both as a result of their own placement and as a result of the other machine being placed. This MATLAB code of this tool is shown in Appendix H.

5.3. Validation of the model

Now that the model is finished, it is important to validate it. There are two things that need to be validated. First, the displacements from the ANSYS model need to be checked. The question is whether these are in line with the expected displacements. This validation is done in Subsection 5.3.1. After that, the displacements retrieved using the POD-RBF approximation, need to be validated. This is done in Subsection 5.3.2.

5.3.1. Displacements obtained with ANSYS

Appendix H shows the MATLAB code of the developed tool. In this code, the displacements obtained from ANSYS are loaded and organized. Figure 5.6 presents the displacements retrieved from ANSYS (using the APDL code from Appendix G.2) for the parameters $t_{floor} = \mathrm{X}\,[m]$, $w_{frame} = \mathrm{X}\,[m]$, $L = \mathrm{X}\,[m]$, and $H_2 = \mathrm{X}\,[m]$. The x-axis of the figure shows the *loadstep*, which refers to the ANSYS model. Each loadstep corresponds to a different machine placement. As shown in Figure 5.5, there are $4 * 14 = 56$ different possible machine locations for $L = \mathrm{X}$. The loadsteps represent the machine locations from that figure, ordered from bottom to top and from left to right.

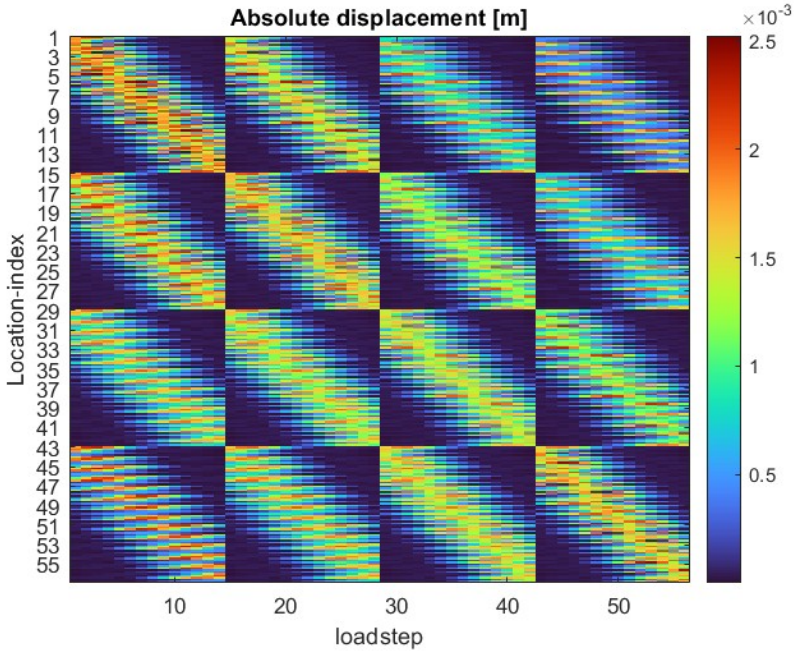

Figure 5.6: Displacements obtained from ANSYS for parameters $t_{floor} = \mathrm{X}\,[m]$, $w_{frame} = \mathrm{X}\,[m]$, $L = \mathrm{X}\,[m]$, and $H_2 = \mathrm{X}\,[m]$.

The y-axis shows the *location index*. Each location index contains the displacements under the legs of the machine. Since the machine has 5 legs, there are $5 * 56 = 280$ values in this column. The diagonal of the figure shows the displacements caused by the forces of the machine placed at its own location. The cross terms indicate the displacement at other potential machine locations. In this way, the displacements from arbitrary machine locations can be easily added together.

There are several indications that the correct displacements are being retrieved here. The first is the block structure visible in the figure. This structure appears because, after 13 shifts in the y-direction, the machine location returns to the lowest position but shifted to the right. The second indication is the dominant diagonal. On the diagonal, the displacements of the 5 legs are shown as a result of the forces from the machine at its own location. These displacements are, of course, larger than those at other potential machine positions.

The figure only shows the displacements for the values $t_{floor} = \mathrm{X}\,[m]$ and $, w_{frame} = \mathrm{X}, [m]$. Figure 5.7 shows the displacements for $t_{floor} = \mathrm{X}$~X $[m]$ and $w_{frame} = \mathrm{X}$~X $[m]$ displayed one after another. It is essentially Figure 5.6 placed 25 times next to each other. On the far left, you can see $t_{floor} = \mathrm{X}\,[m]$ and $, w_{frame} = \mathrm{X}\,[m]$. Then, t_{floor} is gradually increased step by step to X $[m]$. After that, w_{frame} is set to X $[m]$, and this continues accordingly.

What now indicates that the retrieved displacements seem to be correct is that the displacements become smaller as w_{frame} and t_{floor} increase. For loadstep 1, this is also shown separately in Figure 5.8. This is also the displacement matrix $\mathbf{U}$ for machine position 1, which is used in the POD-RBF approximation.

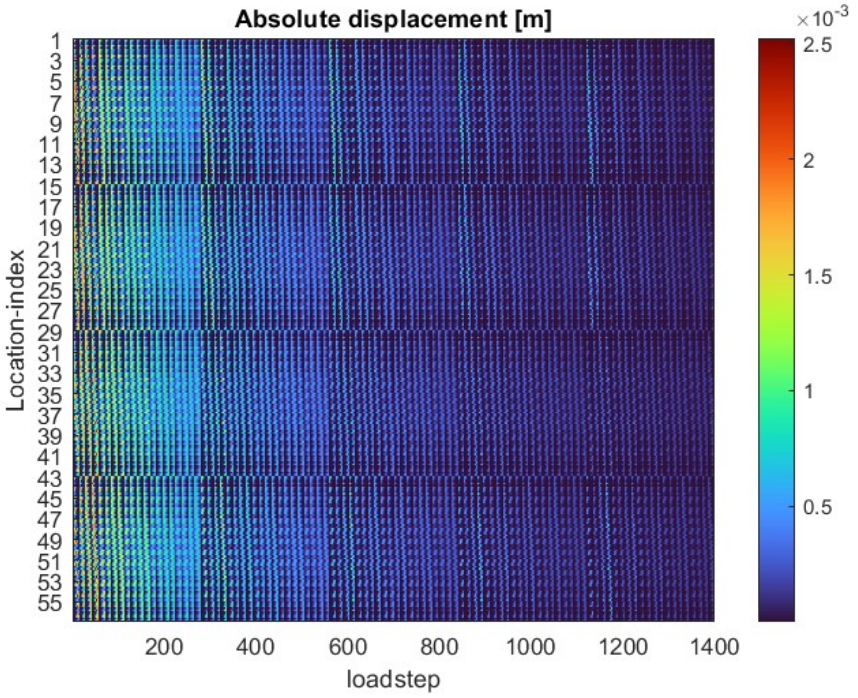

Figure 5.7: Displacements obtained from ANSYS for parameters $t_{floor} = \mathrm{X}$ ~X $[m]$, $w_{frame} = \mathrm{X}$ ~X $[m]$, $L = \mathrm{X}\,[m]$, and $H_2 = \mathrm{X}\,[m]$

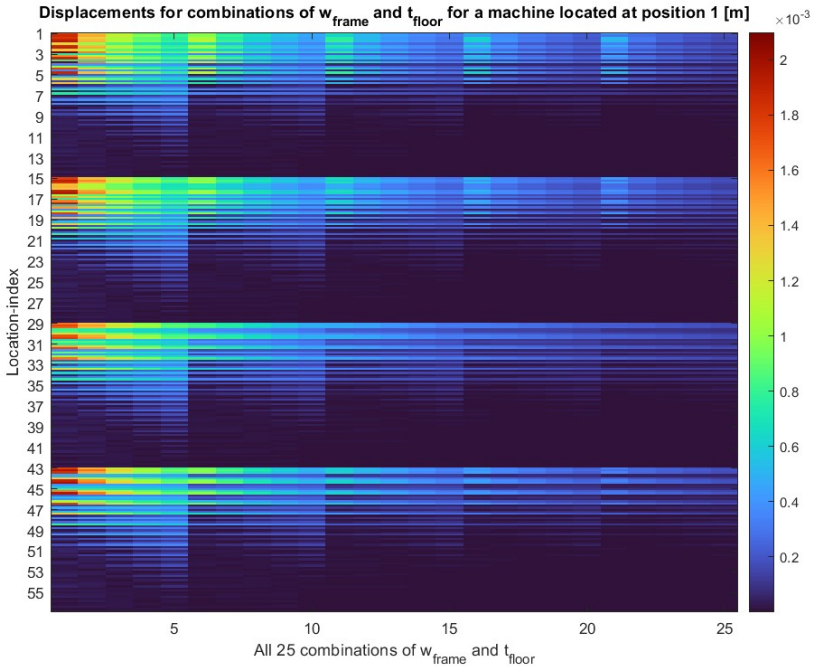

Figure 5.8: Displacements obtained from ANSYS for parameters t_{floor} = X ~X $[m]$, w_{frame} = X ~X $[m]$, L = X $[m]$, and H_2 = X $[m]$ for a machine located at position 1.

5.3.2. Displacements obtained with POD-RBF approximation

Based on the figures above, the displacements retrieved from ANSYS match expectations and therefore appear to be correct. The next step is to validate the displacements obtained using the POD-RBF approximation, which are based on these ANSYS displacements. This is done by comparing the displacement vector obtained from the POD-RBF approximation (Appendix H) with the displacement vector directly retrieved from the ANSYS model (attachments Subsection 5.2.2).

From Section 4.3, it logically follows that the error is near zero when the dataset is exactly at a training point. For this case study, the training points are X, X, X, X, and X $[m]$ for both t_{floor} and w_{frame}. The results for the dataset at these exact training points can be seen in Table 5.2 (X inserted; not for public release). It can be seen that for this case study, both the median and the maximum error are also near zero.

t_{floor} [m]	w_{frame} [m]	L [m]	H_2 [m]	x_{loc} [m]	y_{loc} [m]	Median Error [%]	Max Error [%]
X	X	X	X	X	X	*6.68E-07*	*9.99E-06*
X	X	X	X	X	X	*3.93E-07*	*9.72E-06*
X	X	X	X	X	X	*6.06E-07*	*6.88E-03*
X	X	X	X	X	X	*3.93E-07*	*3.13E-05*
X	X	X	X	X	X	*1.17E-07*	*1.65E-03*

Table 5.2: Validation results for POD RBF approximation at training values

What also becomes clear from Section 4.3 is that the error is relatively large when the chosen data points from the dataset fall exactly between the training points. Therefore, for this case study, it was also tested what the error is when t_{floor} and w_{frame} individually or both fall exactly between training points. The results of this can be seen in Table 5.3 (X inserted; not for public release).

To clarify: the values for t_{floor} and w_{frame} were used as input for both the developed POD-RBF model and the parametric FEM model. Each model produces displacement results for all potential machine locations. However, unlike in Table 5.2, the displacements from the parametric FEM model were not directly included in the training data. Instead, they are approximated by the POD-RBF model, which leads to differences between the displacement results. Comparing these results yields a median and a maximum error.

t_{floor} [m]	w_{frame} [m]	L [m]	H_2 [m]	x_{loc} [m]	y_{loc} [m]	Median Error [%]	Max Error [%]
X	X	X	X	X	X	4.22	144.56
X	X	X	X	X	X	16.43	4497.71
X	X	X	X	X	X	12.87	316.27
X	X	X	X	X	X	13.34	848.21
X	X	X	X	X	X	0.88	721.24
X	X	X	X	X	X	2.33	2774.43
X	X	X	X	X	X	10.33	1278.31
X	X	X	X	X	X	10.77	17.41
X	X	X	X	X	X	10.65	2653.22
X	X	X	X	X	X	11.71	813.24
X	X	X	X	X	X	13.58	596.31
X	X	X	X	X	X	14.66	497.05
X	X	X	X	X	X	15.71	1186.19
X	X	X	X	X	X	17.91	1917.38

Table 5.3: Validation results for POD RBF approximation in between training values

In the first six rows, only the parameters were varied, and the machine location remained the same relative to the beams. The median error stays limited, but it can be seen that the maximum error can be very large. It therefore seems that the POD-RBF approximation gives a good overall idea of the new displacement field, but that large outliers can occur. In the next eight rows, the machine locations were also varied, and similar results were observed.

To gain more insight into this maximum error, the errors have been plotted against the absolute errors. For the last row in Table 5.3, this is shown in Figure 5.9. It can be seen in this figure that the absolute error for the larger relative errors is very small compared to other absolute errors. Similar patterns can be observed for the other cases. This can be seen in Appendix I.

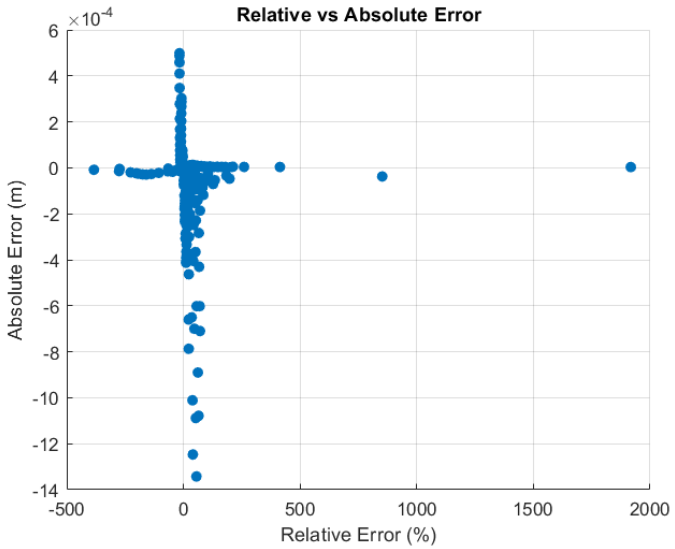

Figure 5.9: Relative vs absolute error for $t_{floor} = w_{frame} = \mathrm{X}\,[m]$, $L = \mathrm{X}\,[m]$, $H_2 = \mathrm{X}\,[m]$, $x_{loc} = \mathrm{X}\,[m]$, $y_{loc} = \mathrm{X}\,[m]$.

These maximum errors have a minimal absolute error and therefore do not have a significant

impact on the final rotations and torsion of the machine. As a result, these maximum errors are not very problematic. However, the figure (and the figures in Appendix I) also show that there are cases where both the relative and absolute error are quite large. For example at around 100% relative error in Figure 5.9. There are not many data points where this occurs, but in these situations, it does have a substantial influence on the displacements and so on the rotations and torsion. Therefore, the POD-RBF approximation does not provide a reliable estimate of the displacements in those cases.

The reason why the errors can still become quite large is due the displacements of the DOFs in the structure not varying linearly in relation to one another when t_{floor} or w_{frame} is modified. As shown in Figure 5.10, the shape of the displacement field changes substantially when only t_{floor} is increased (X inserted, not for public release). The beams now play a relatively smaller role, causing the displacements beneath the beams to become relatively larger. It is difficult to interpolate between such a changed displacement field. This is similar to what was explained in Section 4.3, where interpolation was also difficult due to the displacements of the DOFs in the structure not varying linearly in relation to one another when the force location changed (see Figure 4.11).

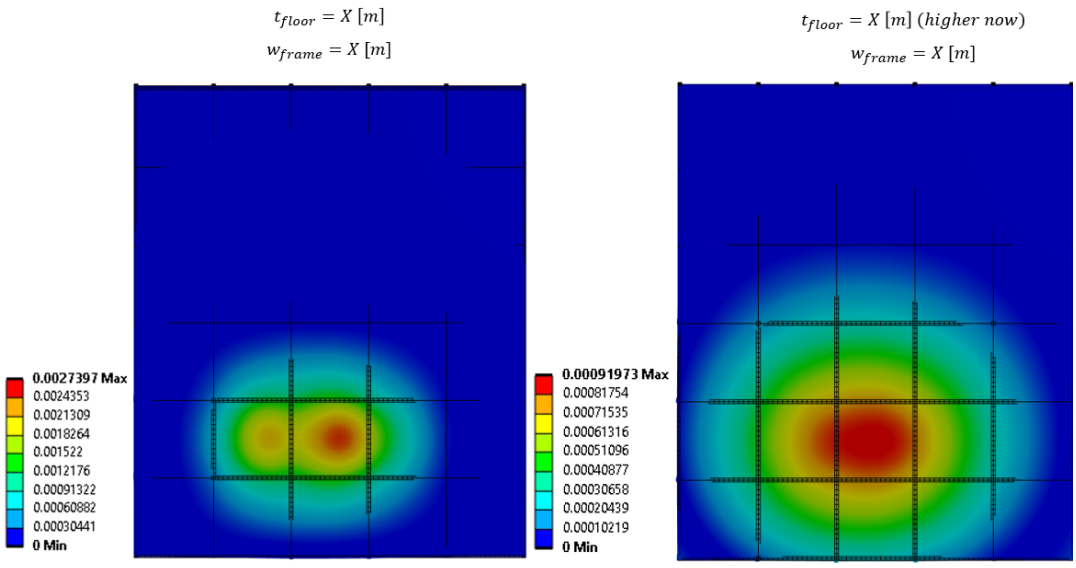

Figure 5.10: Displacement of two factories with different values for t_{floor} and $w_{frame} = \text{X}\,[m]$. For both: $L = \text{X}\,[m]$, $H_2 = \text{X}\,[m]$, $x_{loc} = \text{X}\,[m]$, $y_{loc} = \text{X}\,[m]$.

From these results it can be concluded that most of the displacements are approximated accurately (see median error), and that the largest outliers often have minimal impact. This indicates that the global displacement of the factory floor is well interpolated. However, due to the displacements of the DOFs not varying linearly in relation to one another, which makes interpolating difficult, larger outliers may still occur that cannot be guaranteed to stay within a low error margin. Since ASML wants to be able to determine the rotations and torsion for each factory floor and potential machine location within a relatively low error, the POD-RBF approximation is ultimately not very suitable in this case.

For this reason a final model is created which does not use POD-RBF approximation, but only the generated data. The only difference this makes is that it is now no longer possible to choose any value for t_{floor} and w_{frame} between X $[m]$ and X $[m]$. One of the specific values X, X, X, X or X $[m]$ must be selected. This final model is the MATLAB code shown in Appendix H, but without the POD-RBF approximation steps.

5.4. Insights derived from the model

Although the POD-RBF approximation does not provide the consistent level of accuracy that may have been hoped for, the now created final model, which provides the displacements based solely on the data, can still offer valuable insights. One of these insights is explained in this section.

As already mentioned in Subsection 5.1.1, ASML states that the stiffness under the legs (K_{legs}) must be at least X $[N/m^2]$. This requirement is based on two conditions. First, the dynamic condition, which states that the frequency of the machine should not coincide with the natural frequency of the floor. Second, the machine is not allowed to rotate more than X $[\mathrm{urad}]$ in the x- and y-direction, and the maximum out-of-plane torsion must not exceed X $[\mathrm{um}]$. The retrieved data does not give insight into the dynamic requirement, but it can definitely give insight into the static rotation requirements.

Since all data have already been retrieved, it can now be quickly processed using MATLAB (loop through Appendix H, without the POD-RBF approximation steps). This allows to state that the stiffness requirement under a single placed machine must be at least X $[N/m^2]$ and identify the situation in which a second machine results in values that are as close as possible to $r_x = r_y =$ X $[\mathrm{urad}]$. Figure 5.11 shows this situation (X inserted, not for public release). When only one of the two machines is placed, it holds that $r_x =$ X $[\mathrm{urad}]$ ($K_{leg,min} =$ X $[N/m^2]$), while placing both machines results in $r_x =$ X $[\mathrm{urad}]$. This means that a maximum of 47% of the limit is reached. Based on this, it can be concluded that if for each individual machine $K_{legs} >$ X $[N/m^2]$, the requirements $r_x <$ X $[\mathrm{urad}]$ or $r_y <$ X $[\mathrm{urad}]$ are met, even when a second machine is placed.

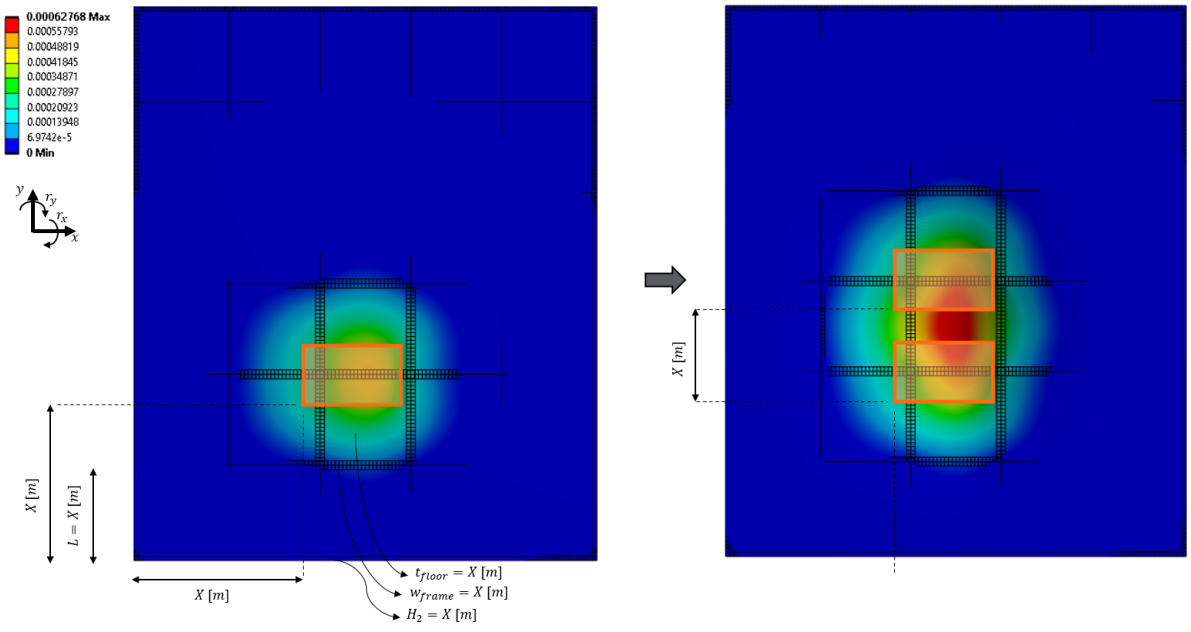

Figure 5.11: Situation in which rotation is critical with two machines for $K_{legs} >$ X $[N/m^2]$.

Since a maximum of 47% is reached with the requirement $K_{legs} >$ X $[N/m^2]$, the question arises whether the rotation requirements can still be met with a lower stiffness requirement. Therefore, tests were conducted for $K_{legs} >$ X $[N/m^2]$ and $K_{legs} >$ X $[N/m^2]$. Figure 5.12 shows the situations for these K_{legs}-requirements in which a second machine results in values that are as close as possible to $r_x = r_y =$ X $[\mathrm{urad}]$ (X inserted, not for public release).

The left figure shows the situation for $K_{legs} >$ X $[N/m^2]$. For the upper machine, the rotation

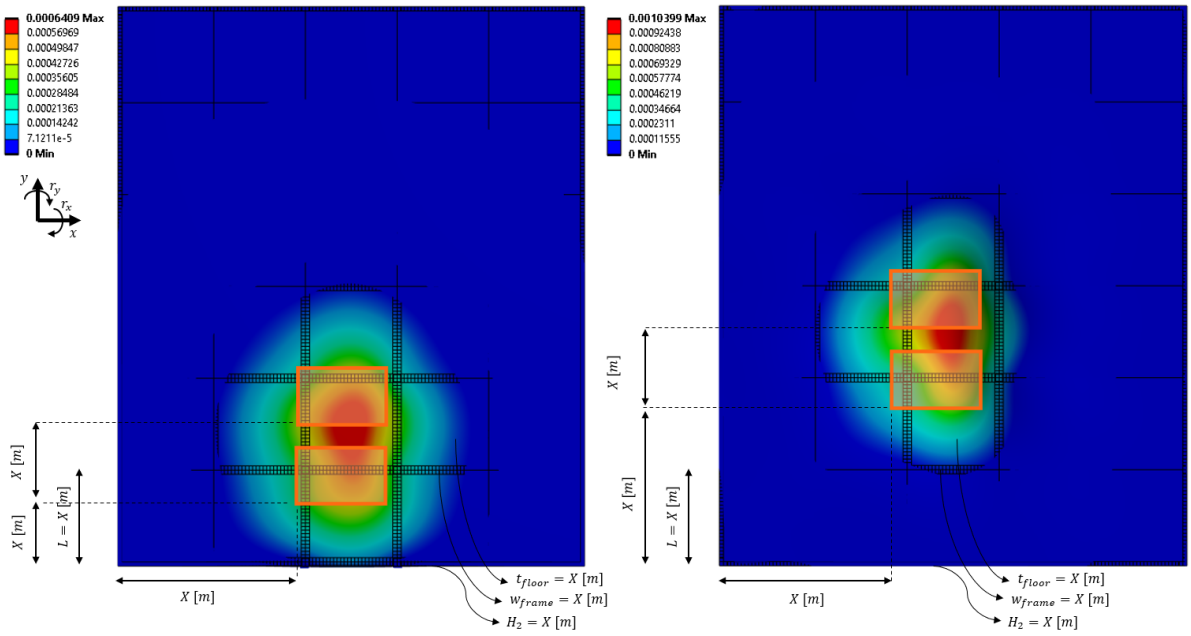

Figure 5.12: Situation in which rotation is critical with two machines for $K_{legs} >$ X (left) and X $[N/m^2]$ (right).

in x-direction caused by its own weight is $r_x =$ X $[\mathrm{urad}]$. However, due to the influence of the lower machine, this increases to $r_x =$ X $[\mathrm{urad}]$ (65%).

The right figure shows the situation for $K_{legs} >$ X $[N/m^2]$. For the upper machine, the rotation in x-direction caused by its own weight is $r_x =$ X $[\mathrm{urad}]$. However, due to the lower machine, this increases to $r_x =$ X $[\mathrm{urad}]$ (113%), which means the rotation requirement is no longer met.

These insights obtained from the model data indicate that, given the current requirement $K_{legs} >$ X $[N/m^2]$, a second machine would not violate the rotation limits. At the same time, it appears that even with a reduction to $K_{legs} >$ X $[N/m^2]$, the requirements would still be met. This suggests that a reduction of the stiffness requirement may be feasible, potentially allowing factories to be built with less material. However, no analysis was performed on r_t and the dynamic requirements, which also influence this requirement.

It is important to realize that this model is only a fictitious simplification of what an actual factory floor looks like. Additionally, not all parameter variations have been explored (see selection in Table 5.1), as this would require an excessive amount of training data. Therefore, further research is certainly needed to confidently determine whether the stiffness requirement can be reduced while still meeting the rotation constraints.

5.5. Findings and evaluation

This chapter originates from SQ4: '*How do the techniques that meet the implementation requirements perform when applied to a designed ASML case study?*'. To answer this question, it was necessary to develop a case study, apply the technique that satisfies the implementation requirements to it, and derive results from this process. This has now been accomplished by using MATLAB and ANSYS to set up a model that applies the POD-RBF approximation to obtain the displacements of a machine placed on a factory floor.

To now fully answer this question an evaluation of how well the technique actually performs is required. This is addressed in this section by assessing the results obtained. In Subsection 5.5.1 this is done by evaluating the results against the final objectives for successful imple-

mentation in ASML's design process. Following this, Subsection 5.2.2 presents an evaluation based on a focus group, which evaluates the extent to which the research meets ASML's needs and to identifies opportunities for improvement.

5.5.1. Assessment of final objectives

Throughout this research, the Design Science Research Methodology (DSRM) was used as the overarching approach. At the end of each chapter, specific objectives were formulated that the technique(s) needed to meet in order to be successfully implemented in ASML's design process. These objectives were initially introduced after Chapter 2, based on insights from the literature. In Chapter 3, the objectives were updated according to the current status and requirements of ASML's design process. Finally, following Chapter 4, the objectives were finalized based on the outcomes of verification tests with the identified and adapted techniques. These final objectives are presented in Section 4.6.

After completing this case study, it can be evaluated whether the application of the POD-RBF approximation in a realistic setup within ASML also meets these final objectives. The evaluation is presented in Table 5.4. To provide a clearer argumentation, the evaluations for *parameters to vary* and *accuracy* have been combined.

Category	Final objectives	Evaluation
Parameters to vary	For all the objectives in this category, it is important to consider during the training setup whether this parameter change may potentially need to be made later. • Multiple parameters can be varied at the same time. • Both structural parameters with linear and non-linear effects on the system, as well as the magnitude of the applied force, can be adjusted. • Parametric changes can be applied to both large parts of the structure and to local areas.	Based on the literature and the test results from Chapter 4, all objectives seem achievable. However, the case study has demonstrated that meeting the strict 5% accuracy requirement is not feasible with a realistic number of interpolation points when the displacement field changes significantly due to parameter variation. In such cases, the POD-RBF approximation struggles to interpolate certain degrees of freedom accurately, leading to a considerable increase in both median and maximum error. It should be noted, however, that these high maximum errors typically correspond to very small absolute errors and therefore have minimal impact on the final displacement of the machine. Still, there are also cases in which both the relative and absolute errors are quite high, meaning that the POD-RBF approximation can indeed result in critical displacement inaccuracies. As such, this requirement can only be fulfilled for simple structures where the displacement field does not change, such as those presented in Chapter 4. For these simple cases, it does not matter whether multiple parameters vary simultaneously, whether the parameter influence is linear or nonlinear, or whether the changes occur globally or locally.
Accuracy	• The displacement vector on the interface DOFs must have a maximum deviation of 5% compared to the direct solution	
Time reduction	• Setting up the technique should not take longer than 2 weeks. • Running a new parameter set should take no more than 1 minute.	Setting up the model with the POD-RBF approximation takes approximately four days for this case study and calculating a new parameter set takes about one second. Since the model in this case study is already quite large (up to 375,000 DOFs) and still performs well below the objectives, it can be concluded that these objectives can be met for most models within ASML.
Integration capability within ASML	• The technique only need to be compatible with ASML's substructuring approach for modules where the coupling between interface DOFs is minimal. • The technique should only use software that is already used at ASML.	In the case study, no issues were encountered that suggest these objectives cannot be achieved. Combined with the findings from the theory and Chapter 4, this leads to the conclusion that these objectives can indeed be met.

Table 5.4: Evaluation of the final objectives for implementation of techniques enabling parametric changes within ROMs to speed up ASML's design process.

Now that step 5 of the DSRM (evaluation) has been completed, a final conclusion can be drawn regarding the applicability of techniques that enable parametric changes within ROMs to speed up ASML's design process. As also written in the table, it is not possible to achieve deviations close to the target if the displacements of the DOFs in the structure do not vary linearly in relation to one another (see Figure 5.10). As a result, the objectives are only met when the displacements of the DOFs vary linearly in relation to one another. In this case

study, however, this is not the case, and therefore the answer to SQ4 is that the POD-RBF approximation performs poorly.

Although this case study shows that it is not possible to maintain a low displacement error, there may be other cases where this is achievable. There are likely other situations within ASML where the issue of displacements of DOFs not varying linearly in relation to one another does not occur, such as the simpler cases presented in Chapter 4. In these cases the POD-RBF approximation would be suitable for successfully speeding up ASML's design process.

This study did not identify such a scenario, but it remains possible that even in cases where the displacements of the DOFs do vary quite linearly in relation to one another, the maximum relative error still exceeds the defined 5 percent requirement. Therefore, it is important to mention that this case study shows that even when a high relative error occurs, the absolute error may still be negligible. As a result, the POD-RBF approximation can still offer added value. This suggests that the strictness of the 5 percent maximum relative error requirement may not always be justified in practice.

The fact that POD-RBF could not be directly used in this case study does not mean that the data collected to train the model was without value. On the contrary, the data provided relevant insights into a practical design concern at ASML: the concern that placing a second machine next to the first one might cause the first machine to rotate excessively. The results suggest that if the stiffness requirements defined by ASML for its customers are met, this excessive rotation is unlikely to occur. In fact, based purely on the rotational requirements, there even appears to be room to slightly relax the current stiffness constraint from $K_{legs} > \mathrm{X}\,[\mathrm{N/m^2}]$ to $K_{legs} > \mathrm{X}\,[\mathrm{N/m^2}]$, purely looking at the rotation requirements. It is important to note that this conclusion is based on a simplified model and considers rotation only. Therefore, further research is needed to validate these findings.

5.5.2. Evaluation through focus group

To evaluate whether the results and conclusions from the case study and the rest of the research align with ASML's needs, a focus group was organized. A focus group is a research method in which a small group of people, guided by a moderator, engages in a structured discussion on a specific topic [35]. The main goal here was to evaluate the extent to which the research meets ASML's needs and to identify opportunities for improvement.

Focus group setup

The focus group was conducted with the same ASML employees who participated in the semi-structured interviews described in Chapter 3 (see Table 3.1). These participants had been selected based on their experience with ASML's design process, their representation of different teams, and their willingness to contribute. These same criteria also made them suitable for participation in the focus group.

Another important reason for selecting these participants again is that this way, two members of the FEM team and two from the placement team (one directly and one indirectly involved) are taking part. It is important that both teams are well represented, as they benefit most from the results of this research. The FEM team benefits mainly because of the potential to speed up the design process, which plays an important role in their daily work. The placement team benefits from the model developed in this chapter, as it can help them gain more insight into the displacements of machines on different factory floors.

As noted above, the focus group aimed to evaluate the extent to which the research meets ASML's needs and to identify opportunities for improvement. To achieve this, three sub-goals

were defined when organizing the session. The first two were more relevant for the FEM team, while the last one was mainly aimed at the placement team.

The first was to re-evaluate the previously defined objectives for successful implementation of the techniques to speed up ASML's design process, which were partly based on the semi-structured interviews. The second sub-goal was to present the results related to whether these objectives had been met and to check if any important aspect had been overlooked. The third and final sub-goal was to demonstrate that, although the POD-RBF approximation did not perform well in the case study, the collected data could still offer valuable insights. Additionally, it aimed to verify whether any critical points had been missed in the case study.

To achieve these main and sub-goals, the following structure was used for the focus group:

1. **Introduction:** In this part, the participants were welcomed, the structure of the focus group was explained, and the goals of the session were briefly communicated to the participants.
2. **Presentation:** Since not all participants were up to date with the research, a presentation was given to share the defined objectives and the results of the research and the case study.
3. **Discussion:** This was the most important part of the focus group where an open discussion took place. To guide the conversation, various questions and follow-up questions were prepared.
4. **Closing:** In this part, the objectives stated in the introduction were revisited, and the focus group was formally concluded.

Focus group outcomes

Regarding the first sub-goal, which was to re-evaluate the defined objectives, the participants indicated that the objectives formulated during the study were realistic and comprehensive. There was broad agreement that these requirements are essential for the successful implementation of a technique within ASML. No additional or conflicting criteria were mentioned by the participants.

The fact that the requirement of a maximum relative error of 5% was questioned due to the possibility of very small absolute errors (Subsection 5.1.1) was not discussed during the focus group, as this only became clear afterwards. However, after individual discussions with two ASML colleagues, this was acknowledged as a valid point of concern. As a result, the requirement of a maximum relative error of 5% was no longer considered necessary in cases where the absolute error is insignificant.

The second sub-goal was to present the results regarding whether these objectives were met and to check if any important aspects had been overlooked. Within the focus group, the results were largely accepted and considered valid. Regarding the possibility of overlooking important aspects, it was suggested that it could be interesting to further analyze the existing dataset using neural networks. This might offer an alternative approach to the interpolation problem, especially considering the limitations of the POD-RBF method. This is elaborated further in Chapter 6.

The third sub-goal was to show that some valuable insights could still be drawn from the collected data and to check whether any important aspects had been overlooked. During the open discussion, participants agreed that, even though POD-RBF is not applicable in this case, the data collected for it is still certainly useful. However, several remarks were made about the data. For example, it was suggested that the focus should have been more on

varying machine locations rather than increasing the number of parameters. Additionally, it was pointed out that the model should offer more flexibility in setting the distance between machines, allowing more realistic placement scenarios to be explored. Some participants also felt that other parameters might be more relevant than the ones currently included. Lastly, one of the placement team members mentioned that using a MATLAB tool requires a MATLAB license, which not everyone in the team has access to.

Based on this feedback, the model was rebuilt. In the new version, the focus was placed on machine locations and the placement of a possible second machine. One parameter was removed to compensate for the increasing data generation time, and another parameter was replaced with one considered to be more relevant. These remarks have already been incorporated into the research presented above, but it is important to emphasize that they only emerged thanks to the focus group. Without it, these valuable changes would not have been made, simply because the awareness of these possible improvements was lacking. Finally, an Excel tool was developed in addition to the MATLAB that is easier to use and avoids the need for a license. The interface of this Excel tool is shown in Appendix J. Note that this tool does not allow for quickly looping through the data, as was done in Section 5.5 using the MATLAB tool. With both the Excel and MATLAB models now available, the placement team has an accessible way to examine the interaction between machines across different factories and machine positions.

The main goal of the focus group was to evaluate the extent to which the research meets ASML's needs and to identify opportunities for improvement. By addressing the sub-goals, this main objective was ultimately achieved. It became clear that, regarding the acceleration of the design process, the research aligned well with ASML's needs, with only a few improvement points concerning the maximum error requirement and the exploration of neural networks. For the placement team's case study, more improvement points were identified that needed to be incorporated into the research. With these adjustments now implemented, the case study also aligns well with ASML's needs.

Organizing this focus group enabled an interactive exchange of viewpoints. By discussing the research and its outcomes collectively, participants were able to build on each other's ideas and collectively arrive at practical feedback points that everyone could agree on. This type of input could not have been obtained through interviews alone, which made the focus group a crucial final step in evaluating the research and ensuring that it was of clear value to ASML.

6

Conclusion and recommendations

This chapter presents an answer to the research question and provides recommendations. The goal of this study was to explore techniques that allow for direct parametric changes in ROMs for static analysis, assess how these techniques could be applied within ASML's design process, and evaluate to what extent they could speed up this process. This resulted in the formulation of the following main research question:

RQ: *To what extent can techniques that enable parametric changes within Reduced Order Models for static analysis contribute to speeding up ASML's design process?*

To answer the main research question, four sub-questions were formulated. These are first addressed individually in Section 6.1, followed by the answer to the main question in Section 6.2. Additionally, Section 6.3 presents conclusions that are not directly related to the research questions but are still worth mentioning. Section 6.4 then describes the implications for ASML, and finally, Section 6.5 provides recommendations for future research.

6.1. Answering the sub-questions

In this section, each of the formulated sub-questions is answered.

SQ1: *What is the current state of the art in techniques that enable parametric changes within Reduced Order Models for static analysis?*

Based on an extensive literature review, four techniques were identified that enable direct or faster parametric changes in ROMs for static analysis. These are the Multiparameter Moment Matching Method (MMMM), Proper Orthogonal Decomposition Radial Basis Function (POD-RBF) Approximation, Multilevel Substructuring (MLS), and the Woodbury Identity. The first two are Parametric Reduced Order Modeling techniques, meaning they retain parametric effects after the reduction. The latter two do not preserve parametric effects after the reduction but still offer the potential to apply parametric changes to reduced models in a fast way. While all techniques seem promising, the literature reveals that important details regarding their application are still unknown (see Figure 2.7).

SQ2: *What does ASML's current design process look like, and what requirements must techniques meet to be implemented in this process?*

To provide a well-founded answer to both parts of this question, semi-structured interviews were conducted with several employees familiar with the design process. These interviews

revealed that, for the first part of the question, ASML uses a modular approach in which the machine and the supporting factory floor are divided into roughly 20 modules. A substructuring method is applied, where a ROM is created for each of these modules. These ROMs are then linked together to calculate the behavior of the entire system. It was found that typically X to X design iterations take place (X inserted; not for public release), where each time changes must be made to the FEM model and a new ROM needs to be generated (see Figure 3.2).

Making adjustments to the FEM model typically takes only a few hours, but generating a new ROM each time requires approximately one to three days. This shows that potential time savings of X to X could be achieved if this process were made faster by setting it up parametrically. However, the interviews revealed that creating a ROM for static analysis is already significantly faster than for dynamic analysis. Since this study focuses on static analysis only, the potential to speed up ASML's design process appears to be limited. Still, the research is considered relevant, as some time savings are expected, and techniques used for static analysis often form the basis for techniques for dynamic analysis.

The answer to the second part of the question is presented in Table 6.1. The requirements listed under "General" apply to all techniques. These requirements refer to aspects that are still unknown based on the current literature.

Technique	Requirement	
MMMM	1	Understanding of which parameters allow accurate variation.
	2	The technique must support changes in localized regions of the structure.
	3	The displacement error at interface DOFs must remain below 5%.
	4	The technique must be compatible with substructuring approaches.
POD-RBF	1	The setup time must remain within two weeks.
	2	Understanding of whether varying force locations can be handled.
	3	The displacement error at interface DOFs must remain below 5%.
	4	The technique must be compatible with substructuring approaches.
MLS	1	Understanding of the number of substructures that can be varied effectively.
Woodbury	1	Understanding of the number of element changes that can be handled effectively.
General	A	Implementation must rely solely on existing ASML software tools.
	B	Recalculation time for new parameter sets must remain below 20 minutes.

Table 6.1: Overview of the requirements per technique for successful implementation into the ASML design process.

SQ3: *Do the existing techniques, either in their existing form or adapted, meet the requirements for implementation within ASML's design process?*

The implementation requirements formulated in Table 6.1 were tested using simple structures in MATLAB and ANSYS. For the MMMM, it was found that Requirement 2 cannot be met without introducing large errors, which means that in this case Requirement 3 cannot be fulfilled (see Table 4.3). The tests showed that the POD-RBF approximation meets all requirements. However, with regard to Requirement 4, it was found that an accurate estimate can only be made when the coupling between the interface DOFs is minimal (see Table 4.6).

While performing the tests for MLS it turned out that when the stiffness matrix needs to be extracted from ANSYS, it is limited to a single processor core. Solving the full system using four cores is already faster than exporting the stiffness matrix using one core (see Figure 4.16). Therefore, in relation to Requirement A, this method does not lead to a faster process and is thus not suitable for implementation. As for the Woodbury Identity, it also requires retrieving new stiffness matrices for each new parameter set. As a result, this method is not suitable for

implementation either. The MMMM is also ruled out again for the same reason.

Another important conclusion from the tests for MLS is that generating a ROM for static analysis is relatively fast. For example, a ROM of a substructure with more than 1.5 million DOFs was generated in about 6 seconds (see Figure 4.15). This supports what stated in the answer of the previous sub-question, that since this thesis focuses only on static analysis, the potential to significantly speed up the process is limited. Knowing that the modules within ASML will not have significantly more than 1.5 million DOFs, it can be concluded that the absolute time savings from skipping the reduction step will be minimal.

This does not directly mean that no improvements can be made. The only technique that meets the requirements, POD-RBF approximation, is the only one that also skips the step of modifying the FEM model in addition to skipping the reduction step (see Figure 3.4). This makes it possible to calculate multiple parameter sets faster and still offers a speed-up in absolute time. Moreover, POD-RBF approximation also has the advantage that, once the model is trained, no expensive ANSYS license is needed for the user.

SQ4: *How do the techniques that meet the implementation requirements perform when applied to a designed ASML case study?*

To answer this question, a case study was conducted on a current issue within ASML, to which the techniques that meet the implementation requirements (only POD-RBF approximation) could be applied. The problem in this case study involved a concern within the placement team, which is responsible for positioning the ASML machines on the factory floor. They feared that, in certain situations, placing a second machine next to an existing one could cause the first machine to rotate excessively. Better insights could of course be obtained using ANSYS models, but this team does not have access to ANSYS licenses and has limited knowledge of the software. The use of the POD-RBF approximation is therefore highly suitable in this context, as it offers a fast way to retrieve displacements without the need for ANSYS licenses. However, the test results show that it is not possible to achieve an accuracy close to Requirement 3 in this case study (see Table 5.3). This is because POD-RBF approximation has difficulty achieving accurate interpolation when the displacements of the DOFs in the structure do not vary linearly in relation to one another when certain parameters are changed (see Figure 5.10).

The fact that the POD-RBF approximation does not work in this case study due to this issue does not mean that it could not work in any case within ASML. There will most likely be cases where this issue does not occur, in which case the POD-RBF approximation could be suitable for successfully speeding up ASML's design process.

It could potentially still be difficult in such cases to meet the strict Requirement 3. That is why it is important to note that this case study shows that this requirement may not always be justified in practice, as even when a high relative error occurs, the absolute error can still be negligible (see Figure 5.9). In such situations, not meeting this requirement is essentially inconsequential, as displacements are still approximated accurately in absolute terms. Therefore, even if this requirement is not met, POD-RBF approximation could still help speed up the design proces. Note that in the context of this case study this was irrelevant, as the lack of a linear relationship between DOFs displacements led to a considerable absolute error.

Because POD-RBF approximation did not work in this case study, a model was created using only the data required for setting up POD-RBF approximation (see Appendix H and J). This model therefore does not provide interpolation as in the POD-RBF approximation, but instead allows direct selection of the values from the generated data. In this way, a solution to the

problem is still provided, although with less flexibility in choosing the parameters. This demonstrates that even without using any of the techniques identified in the literature, but purely based on data, support can still be provided.

6.2. Answering the main research question

Now that all sub-questions have been addressed, the main research question '*To what extent can techniques that enable parametric changes within Reduced Order Models for static analysis contribute to speeding up ASML's design process?*' can be answered. The findings suggest that one technique can indeed help speed up ASML's design process, although there are some very important limitations to take into account.

This technique is the POD RBF approximation, which is the only method found to meet ASML's implementation requirements. It allows both the step of generating a ROM and the step of regenerating an FEM model to be skipped. While it was the idea that the potential time savings would come from skipping the ROM generation step, the research shows that generating a ROM for static analysis is already very fast. This contrasts with dynamic analysis, where this step is significantly more time-consuming, but that falls outside the scope of this study. Skipping this second step is therefore the reason why this technique can speed up the design process. In each of the X to X iterations (X inserted; not for public release), the POD RBF approximation eliminates the need to make changes to the FEM model (see Figure 3.4b). This does provide some time savings, although far less than the potential savings from skipping the reduction step in dynamic analysis.

There are also several important constraints for applying this technique in ASML's substructuring context. Accurate estimates can only be made when the coupling between the interface DOFs in a substructuring setup is minimal, as otherwise it is not possible to work backwards from displacements to the reduced stiffness matrix. Furthermore, the technique can only produce accurate results if the DOFs in the structure respond fairly linearly in relation to one another when parameters are adjusted, as this allows for proper interpolation.

In summary, while the technique can provide some time savings for static analysis, its constraints make it only suitable for certain scenarios. Far greater potential lies in its application to dynamic analysis, where the expected time gains are significantly higher.

6.3. Additional findings

The study uncovered several findings that, while not directly linked to the research questions, provide valuable insights. These are listed below.

- **State of the art review**: This is the first time that a comprehensive state of the art study has been conducted on techniques that allow for parametric changes in ROMs for static analysis. The strengths, weaknesses, and unknowns of these techniques, identified here for the first time (see Figure 2.7), offer a valuable reference point for anyone aiming to apply these techniques in practice.
- **Findings for the MMMM**: The identified unknowns in the state of the art study for the MMMM are now clarified. It was not known which parameters could be accurately varied or whether the method would also work for local changes. Regarding the first point, the research shows that variation is possible for parameters that appear consistently throughout the stiffness matrix, and that force locations and force magnitudes can also be varied (see Table 4.2 and Table 4.3). Regarding the second point, the research shows that it is not possible to make local changes without causing significant median and

maximum errors, or without the method starting to resemble a weakened version of substructuring (see Figure 4.8 and Table 4.3).

- **Findings for POD-RBF approximation**: The identified unknowns in the state of the art study for POD-RBF approximation are now also clarified. The performance for varying force location and magnitude was not known. This research found that the displacement of the DOFs changes completely in relation to each other when the force location changes (see Figure 4.11 and Table E.2), meaning the method does not work in that case. For a varying force magnitude, this does not occur, and POD-RBF approximation does work (see Table 4.5). This research also identified a potential way to set up the POD-RBF approximation more efficiently by combining it with substructuring (see Figure 4.12). In theory, this approach should work, but it would only be beneficial if substructuring is faster than calculating the entire system directly.
- **Findings from the case study**: As already mentioned in the answer to SQ4 in Section 5.1, because the POD-RBF approximation did not work, a model was created in the case study using only the data required for setting up the POD-RBF approximation (see Appendix H and J). This model provided relevant insights into the practical design concern at ASML that placing a second machine next to the first one might cause the first machine to rotate excessively. The results suggest that if the stiffness requirements defined by ASML for its customers are met, this excessive rotation is unlikely to occur (see Figure 5.11). In fact, based purely on the rotational requirements, there even appears to be room to slightly relax the current stiffness constraint from $K_{legs} > \mathrm{X}\,[\mathrm{N/m^2}]$ to $K_{legs} > \mathrm{X}\,[\mathrm{N/m^2}]$ (X inserted; not for public release), purely looking at the rotation requirements (see Figure 5.12). It is important to note that this conclusion is based on a simplified model and considers rotation only. Therefore, further research is needed to validate these findings.

6.4. Implications for ASML

Now that the conclusions of this research have been presented, the implications for ASML can be outlined. This refers to the concrete consequences of this research for ASML, for example how they can apply the results and what impact these results may have. First, the implications of the research into speeding up the design process will be discussed, followed by the implications of the specific case study.

Regarding the design process, the findings show that while techniques enabling parametric changes within ROMs for static analysis can provide some time savings, their constraints make them suitable only for scenarios where the coupling between interface DOFs is minimal and the DOFs in a structure respond fairly linearly in relation to one another. Far greater potential lies in their application to dynamic analysis, where the expected time savings are significantly higher. For ASML, the task is therefore to identify scenarios where implementing the POD-RBF approximation is both possible given these constraints and worthwhile considering the limited time savings. If such scenarios are identified, the next step is naturally to implement the method. The case study in Chapter 5 provides a good example of how this could be done.

A more important next step for ASML in the context of speeding the design process is to initiate research into the use of techniques enabling parametric changes within ROMs for dynamic analysis, as this is where substantial time savings could potentially be achieved. More on this can be found in Section 6.5.

Regarding the case study, there are also several implications. The developed model provides the placement team with an accessible way to examine the interaction between machines for

different factories and machine positions. This model can be used directly within the team as a quick indicator of the displacements, rotations, and torsion of machines in the factory. It can also be used immediately to validate certain assumptions currently made about the relationships between stiffness values on beams or in the middle of the floor.

The concern within the placement team that the machines would rotate excessively under the current specified stiffness requirement appears unnecessary when using this model, as it shows this will not occur. In the future, the placement team could also use the model to investigate the potential for excessive torsion of the machine. Since the research found that even with a lower stiffness requirement no excessive rotation occurs, it is worth further investigating the potential for reducing this requirement. More on this can be found in Section 6.5.

6.5. Recommendations for future research

The study shows, based on both the interviews and the tests, that generating a ROM for static analysis is relatively fast. In contrast, the literature and interviews indicate that generating a ROM for dynamic analysis can be much more time-consuming. This insight leads to the recommendation to focus future research on dynamic analysis. The literature study also showed that techniques suitable for static analysis are limited, while more options appear to be available for dynamic analysis. This is plausible because there is simply more to gain in dynamic analysis.

The literature also showed that MLS is well-suited for dynamic analysis. In the case of static analysis, it turned out that retrieving the stiffness matrix using the limited single-core export was not worthwhile, as solving the full system with four cores was already faster. However, ANSYS allows the use of up to eight cores, and since dynamic calculations are significantly more demanding, it would be interesting to test whether retrieving the required matrices in this context could lead to potential time savings. If this proves to be the case, MLS could offer opportunities to speed up ASML's design process.

In this case, there might also be opportunities for the MMMM. An extension of this method for dynamic analysis is available in the literature [15]. Although this study found that the method leads to large errors in static analysis when local changes are made, it is still uncertain whether this will also apply to dynamic analysis. The result could be different. It is important to note that in this research, several typos were found in Yoo's algorithm for static analysis, which were difficult to detect. Therefore, it is possible that the dynamic analysis algorithms presented in the same paper also contain one or more typos. This should be taken into consideration when trying to rebuild the algorithm.

For both future research on parametric flexibility in ROMs for dynamic analysis and for this study focused on static analysis, it is also interesting to explore other ways to retrieve the updated matrices besides using ANSYS. Although ASML currently works with ANSYS, there may be other software packages or methods that allow to retrieve the updated matrices much faster. This could create some room for applying MLS and possibly the Woodbury Identity for faster parametric changes in ROMs in static analysis. However, it is important to keep in mind that the absolute time savings in static analysis will remain limited. For dynamic analysis, on the other hand, the potential time savings could be much greater.

The literature shows that there are many, often quite complex, techniques available for achieving parametric flexibility in ROMs for dynamic analysis [15, 16, 17, 18, 24, 25]. Just as there was no clear overview of available techniques for static analysis at the start of this study, such an overview is currently also lacking for dynamic analysis. Existing literature typically focuses on a single technique or limits itself to specific categories, such as interpolation-based methods.

Therefore, future research would benefit from creating a structured overview of the available techniques. This could also make it possible to combine different methods to overcome the weaknesses of individual techniques.

Also, it is recommended that future research explores how neural networks could play a role in enabling parametric flexibility in ROMs. For example, the POD-RBF approximation requires a large amount of data to perform interpolation, but the method struggles when the DOFs in the structure do not respond fairly linearly to parameter changes. Neural networks could potentially offer a solution in such cases. The same applies to other interpolation-based methods for dynamic analysis, where large datasets are also generated.

The final recommendation arises from a finding in the case study. Based solely on the rotation requirements, the condition $K_{legs} > \mathrm{X}\,[\mathrm{N/m^2}]$ may be somewhat strict and could potentially be lowered to $K_{legs} > \mathrm{X}\,[\mathrm{N/m^2}]$ (X inserted; not for public release). This would allow factories to be built with less material and therefore at lower cost. Using the developed model, it can be checked whether, for specific machine placements, this requirement could be lowered even further. This might be possible by avoiding machine placement in certain locations. However, it should be kept in mind that this is a simplified model, and further research into this static situation is needed. In addition, this stiffness requirement also comes from dynamic considerations, so research will also be needed in that area before the requirement can actually be reduced. The main message is therefore that, instead of simply giving the customer this high stiffness requirement, it may be worth looking more specifically at potential factories and machine layouts to possibly allow the customer to build a cheaper factory.

References

[1] D. Rixen. “Substructuring Concepts and Component Mode Synthesis”. In: *Handbook of Experimental Structural Dynamics*. Ed. by R. Allemang and P. Avitabile. New York, NY: Springer New York, 2022, pp. 833–856. ISBN: 978-1-4614-4547-0. DOI: 10.1007/978-1-4614-4547-0_16. URL: https://doi.org/10.1007/978-1-4614-4547-0_16.

[2] E. Lappano et al. “A Parametric Model Order Reduction for simulations of beam-based structures”. In: *Virtual Vehicle Research Center and KU Leuven* (2021).

[3] Arnon Levy. “Idealization and abstraction: refining the distinction”. In: *Synthese* 198.Suppl 24 (2021), S5855–S5872. DOI: 10.1007/s11229-018-1721-z. URL: https://doi.org/10.1007/s11229-018-1721-z.

[4] K. Peffers et al. “A design science research methodology for information systems research”. In: *Journal of Management Information Systems* 24 (2007), pp. 45–77.

[5] J. Holmström, M. Ketokivi, and A.P. Hameri. “Bridging Practice and Theory: A Design Science Approach”. In: *Decision Sciences* 40 (2009), pp. 65–87. DOI: 10.1111/j.1540-5915.2008.00221.x.

[6] S. Gregor and O. Zwikael. “Design science research and the co-creation of project management knowledge”. In: *International Journal of Project Management* 42.3 (2024), p. 102584. ISSN: 0263-7863. DOI: 10.1016/j.ijproman.2024.102584. URL: https://www.sciencedirect.com/science/article/pii/S0263786324000267.

[7] R. D. Cook, D. S. Malkus, and M. E. Plesha. *Concepts and Applications of Finite Element Analysis*. 4th Edition. Hoboken, NJ: John Wiley & Sons, 2007.

[8] L. Wu. “Model Order Reduction and Substructuring Methods for Nonlinear Structural Dynamics”. Ph.D. dissertation. Delft University of Technology, July 2018. DOI: 10.4233/uuid:f9736e8b-d00f-4e25-b456-48bd65b43788. URL: https://doi.org/10.4233/uuid:f9736e8b-d00f-4e25-b456-48bd65b43788.

[9] R. J. Guyan. “Reduction of Stiffness and Mass Matrices”. In: *AIAA Journal* 3.2 (1965), p. 380. DOI: 10.2514/3.2874. URL: https://hal.archives-ouvertes.fr/hal-01711552.

[10] S. Weng et al. “Construction of Stiffness and Flexibility for Substructure-Based Model Updating”. In: *Mathematical Problems in Engineering* 2013.1 (2013), p. 706798. DOI: https://doi.org/10.1155/2013/706798. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2013/706798. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2013/706798.

[11] C. Papadimitriou and D.-C. Papadioti. “Component mode synthesis techniques for finite element model updating”. In: *Computers & Structures* 126 (2013). Uncertainty Quantification in structural analysis and design: To commemorate Professor Gerhart I. Schueller for his life-time contribution in the area of computational stochastic mechanics, pp. 15–28. ISSN: 0045-7949. DOI: https://doi.org/10.1016/j.compstruc.2012.10.018. URL: https://www.sciencedirect.com/science/article/pii/S0045794912002520.

[12] W. Swart. “Methods for Improving the Computational Performance of Sequentially Linear Analysis”. Available at http://repository.tudelft.nl/. Master’s Thesis. Delft, Netherlands: Delft University of Technology, Aug. 2018.

[13] J. Lee. “A parametric reduced-order model using substructural mode selections and interpolation”. In: *Computers and Structures* 212 (2019), pp. 199–214. DOI: 10.1016/j.compstruc.2018.10.018. URL: https://doi.org/10.1016/j.compstruc.2018.10.018.

[14] L. Daniel et al. “A Multiparameter Moment-Matching Model-Reduction Approach for Generating Geometrically Parameterized Interconnect Performance Models”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23.5 (2004), pp. 678–693. DOI: 10.1109/TCAD.2004.826583.

[15] E. J. Yoo. “Parametric Model Order Reduction for Structural Analysis and Control”. Advisors: Univ.-Prof. Dr.-Ing. H. Baier and Univ.-Prof. Dr.-Ing. habil. B. Lohmann. Doctoral Dissertation. Munich, Germany: Technische Universität München, Nov. 2010.

[16] G. Zhang, M. P. Castanier, and C. Pierre. “Integration of component-based and parametric reduced-order modeling methods for probabilistic vibration analysis and design”. In: *Proceedings of EURODYN 2005: 6th International Conference on Structural Dynamics*. University of Michigan, Ann Arbor, MI, USA. 2005, pp. 993–998.

[17] S. K. Hong, B. I. Epureanu, and M. P. Castanier. “Next-generation parametric reduced-order models”. In: *Mechanical Systems and Signal Processing* 37.1 (2013), pp. 403–421. DOI: 10.1016/j.ymssp.2012.12.012.

[18] J. Speet. “Parametric Reduced Order Modeling of Structural Models by Manifold Interpolation Techniques: Application on a Jacket Foundation of an Offshore Wind Turbine”. Supervisors: Dr. ir. M. Langelaar, Prof. dr. ir. A. van Keulen, Ir. B. Nortier, Ir. B. Verheugt. Master’s Thesis. Delft, The Netherlands: Delft University of Technology, Nov. 2017.

[19] J. Lee and M. Cho. “An interpolation-based parametric reduced order model combined with component mode synthesis”. In: *Computer Methods in Applied Mechanics and Engineering* 319 (2017), pp. 258–286. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2017.02.010. URL: https://www.sciencedirect.com/science/article/pii/S0045782516307551.

[20] B. Lohmann and R. Eid. “Efficient Order Reduction of Parametric and Nonlinear Models by Superposition of Locally Reduced Models”. In: *Lehrstuhl für Regelungstechnik, Technische Universität München* (2009). Presented at a conference on model order reduction techniques. URL: https://www.rt.mw.tum.de/.

[21] D. Amsallem. “Interpolation on manifolds of CFD-based fluid and finite element-based structural reduced-order models for on-line aeroelastic predictions”. Doctoral dissertation. Stanford University, 2010.

[22] P. Benner, S. Gugercin, and K. Willcox. “A survey of projection-based model reduction methods for parametric dynamical systems”. In: *SIAM Review* 57.4 (2015), pp. 483–531.

[23] V. Buljak. *Inverse Analyses with Model Reduction*. Computational Fluid and Solid Mechanics. Springer-Verlag Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-22703-5_3. URL: https://doi.org/10.1007/978-3-642-22703-5_3.

[24] D. Amsallem and C. Farhat. “Interpolation method for adapting reduced-order models and application to aeroelasticity”. In: *AIAA Journal* 46.7 (2008), p. 1803.

[25] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*. Vol. 120. Academic Press, 1986.

[26] E. Balmes. “Parametric Families of Reduced Finite Element Models”. In: *Mechanical Systems and Signal Processing* 10.4 (1996), pp. 381–394. DOI: 10.1006/mssp.1996.0027.

[27] S. Abhinav, D. Ghosh, and C. S. Manohar. “Substructuring Methods for Finite Element Analysis”. In: *Encyclopedia of Earthquake Engineering*. Berlin, Germany: Springer-Verlag Berlin Heidelberg, 2014, pp. 1–15. DOI: 10.1007/978-3-642-36197-5_267-1.

[28] C. Hyun and P. S. Lee. “A load balancing algorithm for the parallel automated multilevel substructuring method”. In: *Computers & Structures* 257 (2021), p. 106649. ISSN: 0045-7949. DOI: https://doi.org/10.1016/j.compstruc.2021.106649. URL: https://www.sciencedirect.com/science/article/pii/S0045794921001711.

[29] J. K. Bennighof and R. B. Lehoucq. “An automated multilevel substructuring method for eigenspace computation in linear elastodynamics”. In: *SIAM Journal on Scientific Computing* 25.6 (2004), pp. 2084–2106. DOI: 10.1137/S1064827502400650. URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-10244221212&doi=10.1137%2fS1064827502400650&partnerID=40&md5=1eb1d34e4b256f2d65fe949afa2a9786.

[30] OpenAI. *ChatGPT (GPT-4)*. https://chat.openai.com. Accessed via https://chat.openai.com. 2025.

[31] Russell L. Ackoff. *A Brief Guide to Interactive Planning and Idealized Design*. Tech. rep. Unpublished manuscript. Bryn Mawr, PA: Ackoff Center for Advancement of Systems Approaches, 2001. URL: https://thesystemsthinker.com/wp-content/uploads/pdfs/BRIEF%20GUIDE%20TO%20INTERACTIVE%20PLANNING%20AND%20IDEALIZED%20DESIGN.pdf.

[32] ANSYS, Inc. *ANSYS Mechanical APDL Element Reference*. ANSYS, Inc. Canonsburg, PA, USA, 2011. URL: http://www.ansys.com.

[33] ANSYS, Inc. *Extracting Stiffness and Mass Matrix in Workbench*. Accessed from internal documentation. ANSYS. 2015.

[34] MathWorks. *Multicore Programming with MATLAB*. https://nl.mathworks.com/discovery/matlab-multicore.html. 2024.

[35] Peter van Eeuwijk and Zuzanna Angehrn. *How to … Conduct a Focus Group Discussion (FGD). Methodological Manual*. Published at the Zurich Open Repository and Archive, University of Zurich. University of Basel, Swiss Tropical and Public Health Institute (Swiss TPH). 2017. URL: https://doi.org/10.5167/uzh-150640.

A

Algorithms

A.1. Arnoldi algorithm

Algorithm 1 Arnoldi Algorithm

$\mathbf{v}_1 = \frac{\mathbf{b}}{\|\mathbf{b}\|}$
For $k = 1, \ldots, m$:
    $\mathbf{z} = \mathbf{A}\mathbf{v}_k$
    For $i = 1, \ldots, k$:
        $h_{i,k} = \mathbf{v}_i^T \mathbf{z}$
        $\mathbf{z} = \mathbf{z} - h_{i,k}\mathbf{v}_i$
    end
    $h_{k+1,k} = \|\mathbf{z}\|_2$
    $\mathbf{v}_{k+1} = \frac{\mathbf{z}}{h_{k+1,k}}$
end

A.2. MMMM algorithm

Algorithm 2 The MMMM algorithm including two-step Arnoldi method and deflation [15]

Step 1: Initiation

$\mathbf{w}_1^\circ = \frac{\mathbf{b}_1}{\|\mathbf{b}_1\|}$
For $i = 1, \ldots, p-1$:
  $\mathbf{z} = \mathbf{A}^{-1}\mathbf{w}_{2i-1}^\circ$
  For $j = 1, \ldots, 2i-1$:
    $h'_{j,2i-1} = \mathbf{w}_j^{\circ T}\mathbf{z}$
    $\mathbf{z} = \mathbf{z} - h'_{j,2i-1}\mathbf{w}_j^\circ$
  end
  $h'_{2i,2i-1} = \|\mathbf{z}\|$
  $\mathbf{w}_{2i}^\circ = \frac{\mathbf{z}}{h'_{2i,2i-1}}$
  $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}_{i+1}$
  $\mathbf{z} = \frac{\mathbf{x}}{\|\mathbf{x}\|}$
  For $j = 1, \ldots, 2i$:
    $h'_{j,2i} = \mathbf{w}_j^{\circ T}\mathbf{z}$
    $\mathbf{z} = \mathbf{z} - h'_{j,2i}\mathbf{w}_j^\circ$
  end
  $h'_{2i+1,2i} = \|\mathbf{z}\|$
  $\mathbf{w}_{2i+1}^\circ = \frac{\mathbf{z}}{h'_{2i+1,2i}}$
end

Step 2: Deflation

$j = 1$
$\mathbf{W}^\circ = \left[\mathbf{w}_1^\circ\ \mathbf{w}_2^\circ\ \ldots\ \mathbf{w}_{2p}^\circ\right]$
$\mathbf{ch} = \mathbf{W}^{\circ T}\mathbf{W}^\circ$
For $i = 2, \ldots, 2p$:
  If $|\mathbf{ch}_{i,1:i-1}|$ *small enough*:
    $\mathbf{w}_j = \mathbf{w}_i^\circ$
    $j = j + 1$
  end
end
$sz = j - 1$

Step 3: Block Arnoldi

For $i = sz, sz+1, \ldots,$ *chosen limit*:
  $\mathbf{z} = \mathbf{A}^{-1}\mathbf{w}_i$
  For $j = 1, \ldots, i$:
    $h_{j,i} = \mathbf{w}_j^T\mathbf{z}$
    $\mathbf{z} = \mathbf{z} - h_{j,i}\mathbf{w}_j$
  end
  $h_{i+1,i} = \|\mathbf{z}\|$
  $\mathbf{w}_{i+1} = \frac{\mathbf{z}}{h_{i+1,i}}$
end

Step 4: Deflation

$j = 1$
$\mathbf{W} = [\mathbf{w}_1\ \mathbf{w}_2\ \ldots]$
$\mathbf{ch} = \mathbf{W}^T\mathbf{W}$
For $i = 2, \ldots$:
  If $|\mathbf{ch}_{i,1:i-1}|$ *small enough*:
    $\mathbf{v}_j = \mathbf{w}_i$
    $j = j + 1$
  end
end

B

Interview guide

Introduction

1. **Introduction of the research project:** The goal of my master's thesis is to explore techniques that enable parametric changes within ROMs, and how these can be used into ASML's structural analysis design process to potentially speed this up. Do you have any questions regarding the research topic or the interview itself?
2. **Consent reminder:** I would like to remind you of the consent form you signed. This interview will be recorded purely for transcription purposes. Shared information will be kept strictly confidential. The audio file will be deleted after transcription. Also, the transcripts and resulting outcomes will be anonymized to respect your privacy. You can choose not to answer any question or end the interview at any time.
3. **Interviewee introduction:** Before we start with the interview, I would like to gain some background information about you and your experience. Could you please briefly state your role you fulfill within the organization?

Main body

Theme 1: ASML's design process

Question 1: To what extent are you familiar with Reduced Order Models?

Follow-up questions:

- Which techniques are used within ASML for static analyses? (think of, for example, Guyan)
- Which techniques are used within ASML for dynamic analyses? (think of Craig-Bampton)

Question 2: How long does it currently take to create such a ROM based on your experience?

Follow-up questions:

- What steps do you follow in this process?
- Which of these steps takes the most time?

Question 3: What does the current design process look like?

Follow-up questions:

- Is a new ROM created for every modification?
- How long does one iteration take?

Theme 2: Implementation of techniques that enable parametric changes within ROMs

Question 4: How do you see a useful application of techniques that enable parametric changes within ROMs in ASML's design process?

Follow-up questions:

- Which steps could be eliminated in the current process?
- How much time would this save in comparison to the current process?

Question 5: What are the requirements that these techniques should meet to be useful?

Follow-up questions:

- What parameters would you like to be able to vary? (Think of geometry/material properties)
- What reduction times would make these techniques useful?
- What is an acceptable error margin?

Closure

1. **Additional information:** Is there anything else you would like to add or something we haven't discussed yet?
2. **Proposed documents or interviewees:** Are there any documents or other information that could be valuable for my research? Are there any colleagues with extensive knowledge on this topic whom I could interview?
3. **Thank you:** Now we can complete this interview. I will now transcribe this interview and share this with you for verification. Thank you for your time and valuable insights.

C

MATLAB implementations of different techniques

C.1. MMMM

```
%%%%%%%%%%%%%%%%%%%
%% GENERAL CODE %%
%%%%%%%%%%%%%%%%%%%

%% 1. Define Parameters
n_beams = 400;              % Number of beam elements
L = 0.1;                    % Length of each beam element (m)
I = 3.0e-3;                 % Moment of inertia of the beam cross-section (m^4)
A = 0.2;                    % Cross-sectional area of the beam (m^2)
F_MMMM = -1000;             % Force used in constructing projector V (N)
E_0 = 2.0e11;               % E-modulus 1 (N/m^2)
E_1 = 1.0e11;               % E-modulus 2 (N/m^2)
E_2 = 4.0e11;               % E-modulus 3 (N/m^2)
F_distance = 80;            % Number of elements between forces used for V
known_dofs = [1, 2, 3];     % A list of DOFs that are fixed

error_marge1 = 1e-8;        % Error margin deflation 1
error_marge2 = 1e-8;        % Error margin deflation 2
limit = 0;                  % Maximum number of additional columns in V

F_position = [160, 400];    % Element positions of F_x and F_y in tested forces
F_x = [-5000, -5000];       % Magnitudes of F_x in force vectors to test (N)
F_y = [-10000, -10000];     % Magnitudes of F_y in force vectors to test (N)

%% 2. Calculate the K-matrices and F-matrix
K_0 = generate_K_matrix(n_beams, L, E_0, I, A, known_dofs);
K_1 = generate_K_matrix(n_beams, L, E_1, I, A, known_dofs);
K_2 = generate_K_matrix(n_beams, L, E_2, I, A, known_dofs);
F_matrix = generate_F_matrix(n_beams, F_MMMM, F_distance, known_dofs);

%% 3. Generate the projector V
[V, sz, W_circ, W, H_apst, H] = generate_V(K_0, F_matrix, error_marge1,
    error_marge2, limit);

%% 4. Generate results
[x_0, x_0_exact, rel_error_0, rel_error_0_median, time_x0_mmmm, time_x0_exact] =
    compute_x(K_0, K_0, E_0, E_0, V, F_position, F_y, F_x);
[x_1, x_1_exact, rel_error_1, rel_error_1_median, time_x1_mmmm, time_x1_exact] =
```

```
    compute_x(K_0, K_1, E_0, E_1, V, F_position, F_y, F_x);
[x_2, x_2_exact, rel_error_2, rel_error_2_median, time_x2_mmmm, time_x2_exact] =
    compute_x(K_0, K_2, E_0, E_2, V, F_position, F_y, F_x);

%%%%%%%%%%%%%%%
%% FUNCTIONS %%
%%%%%%%%%%%%%%%

%% 1. Setting up V
function [V, sz, W_circ, W, H_apst, H] = generate_V(K_0, F_matrix, error_marge1,
    error_marge2, limit)
    % This function generates the projector matrix V using a series of steps.
    % Input:
    %   K_0: Initial stiffness matrix
    %   F_matrix: Force matrix
    %   error_marge1: Error margin for deflation in Step 2
    %   error_marge2: Error margin for deflation in Step 4
    %   limit: Maximum number of additional columns in V

    % Output:
    %   V: The projector matrix
    %   sz: The size of V after step 2
    %   W_circ, W, H_apst, H: Intermediate matrices used in the process

    W_circ = []; W = []; V = []; H_apst = []; H = [];

    b_1 = F_matrix(:, 1);
    p = size(F_matrix, 2);
    K_0_inv = inv(K_0);

    % Step 1: Initiation
    w_circ_1 = b_1 / norm(b_1);
    W_circ = [W_circ, w_circ_1];
    for i = 1 : (p-1)
        z = K_0_inv * W_circ(:, 2*i - 1);
        for j = 1 : (2*i - 1)
            h_apst_j_2i_min_1 = W_circ(:, j)' * z;
            H_apst(j, 2*i - 1) = h_apst_j_2i_min_1;
            z = z - h_apst_j_2i_min_1 * W_circ(:, j);
        end
        h_apst_2i_2i_min_1 = norm(z);
        H_apst(2*i, 2*i-1) = h_apst_2i_2i_min_1;
        w_circ_2i = z / h_apst_2i_2i_min_1;
        W_circ = [W_circ, w_circ_2i];
        b_i_plus_1 = F_matrix(:, i + 1);
        x = K_0_inv * b_i_plus_1;
        z2 = x / norm(x);
        for j = 1 : (2*i)
            W_circ(:, j);
            h_apst_j_2i = W_circ(:, j)' * z2;
            H_apst(j, 2*i) = h_apst_j_2i;
            z2 = z2 - h_apst_j_2i * W_circ(:, j);
        end
        h_apst_2i_plus_1_2i = norm(z2);
        H_apst(2*i + 1, 2*i) = h_apst_2i_plus_1_2i;
        w_circ_2i_plus_1 = z2 / h_apst_2i_plus_1_2i;
        W_circ = [W_circ, w_circ_2i_plus_1];
    end

    % Step 2: Deflation
    W_circ = [zeros(size(W_circ, 1), 1), W_circ];
```

```
95      j = 1;
96      ch = W_circ' * W_circ;
97      for i = 2 : (2*p)
98          if norm(abs(ch(i, 1 : (i - 1)))) < error_marge1
99              W(:, j) = W_circ(:, i);
100             j = j + 1;
101         end
102     end
103     sz = j - 1;
104
105     % Step 3: Block Arnoldi
106     for i = sz : limit
107         z = K_0_inv * W(:, i);
108         for j = 1 : i
109             h_j_i = W(:, j)' * z;
110             H(j, i) = h_j_i;
111             z = z - h_j_i * W(:, j);
112         end
113         h_i_plus_1_i = norm(z);
114         H(i+1, i) = h_i_plus_1_i;
115         w_i_plus_1 = z / h_i_plus_1_i;
116         W = [W, w_i_plus_1];
117     end
118
119     % Step 4: Deflation
120     W = [zeros(size(W, 1), 1), W];
121     j = 1;
122     ch = W' * W;
123     for i = 2 : size(ch, 1)
124         if norm(abs(ch(i, 1 : (i - 1)))) < error_marge2
125             V(:, j) = W(:, i);
126             j = j + 1;
127         end
128     end
129 end
130
131 %% 2. Compute x with MMMM and exact x
132 function [x, x_exact, rel_error, rel_error_median, time_mmmm, time_exact] =
        compute_x(K_0, K_new, p_0, p_new, V, F_position, F_y, F_x)
133     % This function computes the displacement vector x using both MMMM and the
            exact method.
134     % Input:
135     %   K_0: Initial stiffness matrix
136     %   K_new: Updated stiffness matrix
137     %   p_0, p_new: Initial and updated bending stiffness values
138     %   V: Projector matrix
139     %   F_position, F_x, F_y: Force positions and magnitudes
140
141     % Output:
142     %   x: Displacement vector using MMMM
143     %   x_exact: Exact displacement vector
144     %   rel_error: Relative error between the MMMM and exact solutions
145     %   rel_error_median: Median of the relative error
146
147     f = zeros(size(K_0, 1), 1);
148     f(F_position * 3 - 2) = F_x;
149     f(F_position * 3 - 1) = F_y;
150
151     % MMMM Calculation
152     tic
153     delta_p = p_new - p_0;
```

```
    if p_new == p_0
        K_r = V' * K_0 * V;  % No correction term needed
    else
        dK_dp = (K_new - K_0) / delta_p;
        K_r = V' * K_0 * V + (delta_p * (V' * dK_dp * V));
    end
    f_r = V' * f;
    x_r = K_r \ f_r;
    x = V * x_r;
    time_mmmm = toc;

    % Exact Calculation
    tic
    x_exact = K_new \ f;
    time_exact = toc;

    % Compare solutions
    idx = abs(x_exact) > 1e-9; % Analyze only non-zero values
    rel_error = zeros(size(x_exact));
    rel_error(idx) = (x(idx) - x_exact(idx)) ./ x_exact(idx) * 100;
    rel_error_median = median(abs(rel_error(idx)));
end

%% 3. Generate K Matrix
function K_final = generate_K_matrix(n_beams, L, E, I, A, known_dofs)
    % Generate the global stiffness matrix for a Euler Bernoulli cantilever beam
    % Input:
    %   n_beams: Number of beams in the structure
    %   L: Length of each beam
    %   E: Young's Modulus of the material
    %   I: Moment of inertia of the beam cross-section
    %   A: Cross-sectional area of the beam
    %   known_dofs: A list of DOFs that are fixed
    % Output:
    %   K_final: The stiffness matrix (excluding the known DOFs)

    n_dofs = n_beams * 3 + 3;
    K = zeros(n_dofs, n_dofs);

    for i = 1:3:n_dofs-3
        k_local = [
            (E*A) / L, 0, 0, -(E*A) / L,  0, 0;
            0, (12*E*I) / L^3, (6*E*I) / L^2, 0, -(12*E*I) / L^3, (6*E*I) / L^2;
            0, (6*E*I) / L^2, (4*E*I) / L, 0, -(6*E*I) / L^2, (2*E*I) / L;
            -(E*A) / L, 0, 0, (E*A) / L, 0, 0;
            0, -(12*E*I) / L^3, -(6*E*I) / L^2, 0, (12*E*I) / L^3, -(6*E*I) / L^2;
            0, (6*E*I) / L^2, (2*E*I) / L, 0, -(6*E*I) / L^2, (4*E*I) / L;
        ];
        K(i:i+5, i:i+5) = K(i:i+5, i:i+5) + k_local;
    end

    K_final = K;
    K_final(known_dofs, :) = [];
    K_final(:, known_dofs) = [];
end

%% 4. Generate F Matrix
function F_final = generate_F_matrix(n_beams, F_MMMM, F_distance, known_dofs)
    % This function generates the force matrix F used in setting up V.
    % Input:
    %   n_beams: Number of beams in the structure
```

```
    %   F_MMMM: Force used in setting up V
    %   F_distance: Number of elements between forces
    %   known_dofs: A list of DOFs that are fixed

    % Output:
    %   F_final: The final force matrix after removing the known DOFs

    n_dofs = n_beams * 3 + 3;
    num_forces = n_beams / F_distance;
    F_final = zeros(n_dofs - numel(known_dofs), num_forces);
    for j = 1:num_forces
        F = zeros(n_dofs, 2);  % Two columns: one for x-direction, one for y-
            direction
        node_dof_x = F_distance * 3 * j + 1; % DOF in x-direction
        node_dof_y = F_distance * 3 * j + 2; % DOF in y-direction
        F(node_dof_x, 1) = F_MMMM;  % Force in x-direction
        F(node_dof_y, 2) = F_MMMM;  % Force in y-direction
        F(known_dofs, :) = [];
        F_final(:, 2*j-1:2*j) = F;
    end
end
```

C.2. POD-RBF approximation

```
%%%%%%%%%%%%%%%%%%%
%% GENERAL CODE %%
%%%%%%%%%%%%%%%%%%%

%% 1. Define Parameters
E_start = 2E11;           % Start value of E-modulus (N/m^2)
E_end = 4E11;             % End value of E-modulus (N/m^2)
E_n = 3;                  % Number of training points
E_values = linspace(E_start, E_end, E_n);  % Training parameter vector
E_new = 3.5E11;           % New E-modulus that is tested

I = 3E-3;                 % Moment of inertia of the beam cross-section (m^4)
A = 0.3;                  % Cross-sectional area of the beam (m^2)
L = 0.15;                 % Length of each beam element (m)
F_size = -15000;          % Applied force magnitude (N)
F_location = 400;         % Location of applied force (beam element number)

n_beams = 400;            % Number of beam elements
known_dofs = [1, 2, 3];   % A list of DOFs that are fixed

K_threshold = 0.99;       % Minimum percentage of energy to retain in POD
K_min_cutoff_value = 3;   % Minimum number of POD modes

%% 2. Train POD-RBF model: Generate P, U, A_hat and B
cnt = 0;
f = generate_f_vector(n_beams, F_size, F_location, known_dofs);
for E = E_values
    cnt = cnt + 1;
    K = generate_K_matrix(n_beams, E, I, A, L, known_dofs);
    u = K \ f;
    U(:, cnt) = u;
    P(:, cnt) = E;
end

N = size(U, 1);  % Number of DOFs
M = size(U, 2);  % Number of samples

```

```
% POD: Compute basis Phi using eigenvalue decomposition
if M < N
    D = U' * U;
    [V, LambdaMat] = eig(D);
    lambdaVec = diag(LambdaMat);
    [lambdaVec, idx] = sort(lambdaVec, 'descend');
    cumsumlambdaVec = cumsum(lambdaVec);
    sumlambdaVec = sum(lambdaVec);
    K_cutoff = find(cumsumlambdaVec/sumlambdaVec >= K_threshold, 1, 'first');
    if isempty(K_cutoff)
        K_cutoff = length(lambdaVec);
    end
    V = V(:, idx);
    Phi = zeros(N, M);
    for i = 1:M
        lambda_i = lambdaVec(i);
        v_i = V(:, i);
        Phi(:, i) = (U * v_i) / sqrt(lambda_i);
    end
    Phi = Phi(:, 1:max(K_cutoff, K_min_cutoff_value));
end

if M >= N
    C = U * U';
    [V, LambdaMat] = eig(C);
    lambdaVec = diag(LambdaMat);
    [lambdaVec, idx] = sort(lambdaVec, 'descend');
    cumsumlambdaVec = cumsum(lambdaVec);
    sumlambdaVec = sum(lambdaVec);
    K_cutoff = find(cumsumlambdaVec/sumlambdaVec >= K_threshold, 1, 'first');
    if isempty(K_cutoff)
        K_cutoff = length(lambdaVec);
    end
    V = V(:, idx);
    Phi = V(:, 1:max(K_cutoff, K_min_cutoff_value));
end

A_hat = Phi' * U;        % Coefficients for the training data in reduced space
B = podBmtx(A_hat, P);   % RBF coefficient matrix

%% 3. Predict u with POD-RBF approximation
pX = (E_new - E_start) / (E_end - E_start);  % Normalize test parameter
g = podGvec(P, pX);                          % Compute RBF interpolation weights
u_PODRBF = Phi * B * g;                      % Approximate displacement

%% 4. Compute u with direct full model
K_direct = generate_K_matrix(n_beams, E_new, I, A, L, known_dofs);
f_direct = generate_f_vector(n_beams, F_size, F_location, known_dofs);
u_direct = K_direct \ f_direct;

%% 5. Compare results
idx = abs(u_direct) > 1e-12;        % Only compare non-zero entries
rel_error = zeros(size(u_direct));
rel_error(idx) = (u_PODRBF(idx) - u_direct(idx)) ./ u_direct(idx) * 100;
rel_error_median = median(abs(rel_error(idx)))  % Median relative error in %

%%%%%%%%%%%%%%%
%% FUNCTIONS %%
%%%%%%%%%%%%%%%

%% 3. Generate K Matrix
```

```
function K_final = generate_K_matrix(n_beams, L, E, I, A, known_dofs)
    % Generate the global stiffness matrix for a Euler Bernoulli cantilever beam
    % Input:
    %   n_beams: Number of beams in the structure
    %   L: Length of each beam
    %   E: Young's Modulus of the material
    %   I: Moment of inertia of the beam cross-section
    %   A: Cross-sectional area of the beam
    %   known_dofs: A list of DOFs that are fixed
    % Output:
    %   K_final: The stiffness matrix (excluding the known DOFs)

    n_dofs = n_beams * 3 + 3;
    K = zeros(n_dofs, n_dofs);

    for i = 1:3:n_dofs-3
        k_local = [
            (E*A) / L, 0, 0, -(E*A) / L,  0, 0;
            0, (12*E*I) / L^3, (6*E*I) / L^2, 0, -(12*E*I) / L^3, (6*E*I) / L^2;
            0, (6*E*I) / L^2, (4*E*I) / L, 0, -(6*E*I) / L^2, (2*E*I) / L;
            -(E*A) / L, 0, 0, (E*A) / L, 0, 0;
            0, -(12*E*I) / L^3, -(6*E*I) / L^2, 0, (12*E*I) / L^3, -(6*E*I) / L^2;
            0, (6*E*I) / L^2, (2*E*I) / L, 0, -(6*E*I) / L^2, (4*E*I) / L;
        ];
        K(i:i+5, i:i+5) = K(i:i+5, i:i+5) + k_local;
    end

    K_final = K;
    K_final(known_dofs, :) = [];
    K_final(:, known_dofs) = [];
end

%% 2. Generate Force Vector f
function f_final = generate_f_vector(n_beams, F_size, F_location, known_dofs)
    % This function generates a force vector with a point load at a specified
        location.
    % Input:
    %   n_beams: Number of beam elements
    %   F_size: Magnitude of the applied force (N)
    %   F_location: The location of the force on the beam (element number)
    %   known_dofs: A list of DOFs that are fixed
    % Output:
    %   f_final: The resulting force vector after removing the fixed DOFs

    n_dofs = n_beams * 3 + 3;
    f_final = zeros(n_dofs - numel(known_dofs), 1);
    node_dof = [F_location * 3 - 1, F_location * 3 - 2];
    f_final(node_dof) = F_size;
end

%% 3. Compute RBF Matrix B
function [B] = podBmtx(A, p)
    % This function computes the RBF matrix B for training data.
    % Input:
    %   A: The matrix of reduced-order coefficients
    %   p: The parameter vector for RBF interpolation
    % Output:
    %   B: The RBF coefficient matrix

    for j = 1:size(p, 1)
        minP(j, 1) = min(p(j, :));
```

```
        maxP(j, 1) = max(p(j, :));
        for i = 1:size(p, 2)
            x(j, i) = (p(j, i) - minP(j)) / (maxP(j) - minP(j));  % Normalize to [0,1]
        end
    end
    N = size(x, 2);
    for i = 1:N
        for j = 1:N
            G(i, j) = norm(x(:, i) - x(:, j));
        end
    end
    Bt = inv(G') * A';
    B = Bt';
end

%% 4. Compute RBF Weight Vector g
function g = podGvec(p, pX)
    % This function computes the RBF interpolation weights for a new parameter pX.
    % Input:
    %   p: The parameter matrix for training
    %   pX: The new parameter value for interpolation
    % Output:
    %   g: The RBF interpolation weight vector for pX

    N = size(p, 2);
    for j = 1:size(p, 1)
        minP(j, 1) = min(p(j, :));
        maxP(j, 1) = max(p(j, :));
        for i = 1:N
            x(j, i) = (p(j, i) - minP(j)) / (maxP(j) - minP(j));  % Normalize to [0,1]
        end
    end

    gi = @(x, y) norm(x - y);  % Euclidean distance
    for k = 1:N
        g(k, 1) = gi(pX, x(:, k));
    end
end
```

C.3. Substructuring

```
%%%%%%%%%%%%%%%%%%%%
%%  GENERAL CODE %%
%%%%%%%%%%%%%%%%%%%%

%% 1. Define Input
dataPath      = '...';   % Location where files retrieved from ANSYS are stored
model_original = ;       % Number assigned in APDL code to original model
model_modified = ;       % Number assigned in APDL code to modified model
elem_nodes_m   = [];     % interface/master nodes (retrieved using the node_info file)
elem_nodes_s_1 = [];     % slave nodes of the corner substructure

%% 2. Retrieve stiffness matrices, load vector and coupling vectors
tmatrix = tic;
K_a   = load_stiffness_matrix(model_original, dataPath);
K_b   = load_stiffness_matrix(model_modified, dataPath);
f_0   = load_force_vector(model_original, dataPath);
SO2IO = load_SO2IO(model_original, dataPath);
```

```
IO2UO = load_IO2UO(model_original, dataPath);
SO2UO = reorder_blocks(SO2IO,IO2UO);
fprintf('Retrieving matrix : %.4f s\n', tmatrix);

%% 3. Build DOF sets
[~, master_index] = index_retriever(elem_nodes_m, SO2UO);
[~, slave_index_1] = index_retriever(elem_nodes_s_1, SO2UO);
all_indices   = 1:size(K_a,1);     % complete DOF list
taken_idx     = [master_index(:); slave_index_1(:)];  % already-used DOF
slave_index_2 = setdiff(all_indices, taken_idx);      % remaining slave DOF

%% A: SETTING UP SUBSTRUCTURING
%% 4a. Full solution
tExact_a = tic;
x_exact_a = K_a \ f_0;
tExact_a = toc(tExact_a);

%% 5a. Generating substructure of the corner
tSub_a = tic;
[K_mm_hat_1_a, f_m_hat_1_a, termA1_a, termb1_a] = ...
        schur_block(K_a,f_0,master_index,slave_index_1);

%% 6a. Generating substructure of the rest
[K_mm_hat_2_a, f_m_hat_2_a, termA2_a, termb2_a] = ...
        schur_block(K_a,f_0,master_index,slave_index_2);

%% 7a. Assemble substructures
K_mm_a       = K_a(master_index,master_index);
K_mm_hat_a   = K_mm_a - (termA1_a + termA2_a);   % single Schur complement
f_m_hat_a    = f_0(master_index) - (termb1_a + termb2_a);
u_m_a        = K_mm_hat_a \ f_m_hat_a;           % interface displacements
tSub_a = toc(tSub_a);

%% 8a. Time analysis
fprintf('\n--- TIME RESULTS A ----------------------------------\n');
fprintf('Full solve  : %.4f s\n', tExact_a);
fprintf('Substructur : %.4f s\n', tSub_a);
fprintf('-------------------------------------------------\n');

%% OPTIONAL: 9a. Back-substitute to find displacements of slave-DOF
tBack_a = tic;
x_s_1_a = backSub(K_a,f_0,u_m_a,master_index,slave_index_1);
x_s_2_a = backSub(K_a,f_0,u_m_a,master_index,slave_index_2);

%% OPTIONAL: 10a. Build complete solution vector
x_sub_a = zeros(size(K_a,1),1);
x_sub_a(master_index)    = u_m_a;
x_sub_a(slave_index_1)   = x_s_1_a;
x_sub_a(slave_index_2)   = x_s_2_a;
tBack_a = toc(tBack_a);

%% OPTIONAL: 11a. Error analysis
nz = abs(x_exact_a) > 1e-20;
rel_err = abs((x_sub_a(nz)-x_exact_a(nz))./x_exact_a(nz))*100;
fprintf('\n--- ERROR RESULTS A ---------------------------------\n');
fprintf('Median rel %%   : %.3e\n', median(rel_err));
fprintf('Max  rel %%   : %.3e\n', max(rel_err));
fprintf('Backsubstition : %.4f s\n', tBack_a);
fprintf('-------------------------------------------------\n');

%% B: RE-EVALUATION WITH MODIFIED STIFFNESS IN CORNER SUBSTRUCTURE
```

```
%% 4b. Full solution
tExact_b = tic;
x_exact_b = K_b \ f_0;
tExact_b  = toc(tExact_b);

%% 5b. Generating substructure of the corner
tSub_b = tic;
[K_mm_hat_1_b, f_m_hat_1_b, termA1_b, termb1_b] = ...
        schur_block(K_b, f_0, master_index, slave_index_1);

%% 6b. Generating substructure of the rest
% → identical to model A, so reuse contributions
termA2_b = termA2_a;
termb2_b = termb2_a;

%% 7b. Assemble substructures
K_mm_b     = K_b(master_index, master_index);
K_mm_hat_b = K_mm_b - (termA1_b + termA2_b);       % single Schur complement
f_m_hat_b  = f_0(master_index) - (termb1_b + termb2_b);
u_m_b      = K_mm_hat_b \ f_m_hat_b;               % interface displacements
tSub_b     = toc(tSub_b);

%% 8b. Time analysis
fprintf('\n--- TIME RESULTS B ----------------------------------\n');
fprintf('Full solve  : %.4f s\n', tExact_b);
fprintf('Substructur : %.4f s\n', tSub_b );
fprintf('---------------------------------------------\n');

%% OPTIONAL: 9b. Back-substitute to find displacements of slave-DOF
tBack_b = tic;
x_s_1_b = backSub(K_b, f_0, u_m_b, master_index, slave_index_1);
x_s_2_b = backSub(K_b, f_0, u_m_b, master_index, slave_index_2);

%% OPTIONAL: 10b. Build complete solution vector
x_sub_b = zeros(size(K_b,1),1);
x_sub_b(master_index)  = u_m_b;
x_sub_b(slave_index_1) = x_s_1_b;
x_sub_b(slave_index_2) = x_s_2_b;
tBack_b = toc(tBack_b);

%% OPTIONAL: 11b. Error analysis
nz = abs(x_exact_b) > 1e-20;
rel_err = abs((x_sub_b(nz) - x_exact_b(nz)) ./ x_exact_b(nz)) * 100;

fprintf('\n--- ERROR RESULTS B ---------------------------------\n');
fprintf('Median rel %%   : %.3e\n', median(rel_err));
fprintf('Max  rel %%   : %.3e\n', max(rel_err));
fprintf('Backsubstition : %.4f s\n', tBack_b);
fprintf('---------------------------------------------\n');

%%%%%%%%%%%%%%%%%%
%%  FUNCTIONS  %%
%%%%%%%%%%%%%%%%%%

%% 1. FUNCTIONS RETRIEVING MATRICES/VECTOR FROM ANSYS
%% 1.1. Retrieving Stiffness Matrix
function K_final = load_stiffness_matrix(i, dataPath)
    % This function retrieves the stiffness matrix in Solver Ordering
    filename = sprintf('K_SO%.0f.mtx', i);
    fullpath = fullfile(dataPath, filename);
    if ~isfile(fullpath)
```

```
140         error('File does not exist: %s', fullpath);
141     end
142     K_final = mmread(fullpath);
143 end
144 
145 %% 1.2. Retrieving Force Vector
146 function f_final = load_force_vector(i, dataPath)
147     % This function retrieves the force vector in Solver Ordering
148     filename = sprintf('f_SO%.0f.mtx', i);
149     fullpath = fullfile(dataPath, filename);
150     if ~isfile(fullpath)
151         error('File does not exist: %s', fullpath);
152     end
153     f_final = mmread(fullpath);
154 end
155 
156 %%  1.3. Retrieving SO2IO Vector
157 function values = load_SO2IO(i, dataPath)
158     % This function retrieves the Solver Ordering to Internal Ordering
159     % vector
160     filename = sprintf('SO2IO%.0f.txt', i);
161     fullpath = fullfile(dataPath, filename);
162     if ~isfile(fullpath)
163         error('File does not exist: %s', fullpath);
164     end
165     fid = fopen(fullpath, 'r');
166     if fid == -1
167         error('Could not open file.');
168     end
169     values = [];
170     try
171         while ~feof(fid)
172             line = fgetl(fid);
173             if contains(line, '[')
174                 parts = strsplit(line, ':');
175                 if numel(parts) == 2
176                     value = sscanf(parts{2}, '%f');
177                     values(end+1, 1) = value;
178                 end
179             end
180         end
181     catch ME
182         fclose(fid);
183         rethrow(ME);
184     end
185     fclose(fid);
186 end
187 
188 %%  1.4. Retrieving IO2UO Vector
189 function values = load_IO2UO(i, dataPath)
190     % This function retrieves the Internal Ordering to User Ordering
191     % vector
192     filename = sprintf('IO2UO%.0f.txt', i);
193     fullpath = fullfile(dataPath, filename);
194     if ~isfile(fullpath)
195         error('File does not exist: %s', fullpath);
196     end
197 
198     fid = fopen(fullpath, 'r');
199     values = [];
200     while ~feof(fid)
```

```
201         line = fgetl(fid);
202         nums = sscanf(line, '%d');
203         if ~isempty(nums)
204             values = [values; nums];
205         end
206     end
207     fclose(fid);
208 end
209
210 %% 2. FUNCTIONAL FUNCTIONS
211 %% 2.1. Setting Up SO2UO Vector
212 function SO2UO = reorder_blocks(SO2IO, IO2UO)
213     % This function builds the Solver Ordering to User Ordering vector
214     n_per_block = 6;
215     n_blocks = length(IO2UO);
216     if length(SO2IO) < n_blocks * n_per_block
217         error('SO2IO is to short. You need %d blocks of 6.', n_blocks);
218     end
219     [~, sort_order] = sort(IO2UO); % Sort IO2UO and track original positions
220     SO2UO = zeros(n_blocks * n_per_block, 1);  % Preallocate
221     for i = 1:n_blocks
222         block_index = sort_order(i);  % which block position should go here
223         start_idx = (block_index - 1) * n_per_block + 1;
224         end_idx = block_index * n_per_block;
225         SO2UO((i - 1) * n_per_block + 1 : i * n_per_block) = SO2IO(start_idx:
                end_idx);
226     end
227 end
228
229
230 %% 2.2. Index Retriever
231 function [active_mask_K_elem, global_index] = index_retriever(elem_nodes, SO2UO)
232     %   Maps a list of node numbers to global, unconstrained DOF indices.
233     %   Input:
234     %       elem_nodes : Row vector with node numbers
235     %       SO2UO      : Mapping from super-order to user-order DOF
236     %   Output:
237     %       active_mask: Logical mask of active DOF in the local slice
238     %       global_idx : Column vector with active global DOF indices
239     index = [];
240     for i = 1:length(elem_nodes)  % Loop over rijen
241         idx = elem_nodes(i);
242         range = (idx-1)*6 + 1 : idx*6;
243         index = [index SO2UO(range)];
244     end
245     active_mask_K_elem = index ~= 0;                          % binairy mask
246     global_index = index(active_mask_K_elem);                 % global DOF indices
          without boundary conditions
247 end
248
249 %% 2.3. Schur-complement block
250 function [K_hat,f_hat,Apart,bpart] = schur_block(K,f,master,slave)
251     %   Computes the condensed stiffness and load contribution of one slave
252     %   partition with respect to a common master/interface DOF set.
253     %   Input:
254     %       K        : Global stiffness matrix  [nDOF × nDOF]
255     %       f        : Global load vector       [nDOF × 1]
256     %       master   : Row vector of master / interface DOF indices
257     %       slave    : Row vector of slave  DOF indices (disjoint from master)
258     %   Output:
259     %       K_hat    : Schur complement of the slave block
```

```
260     %       f_hat    : Condensed load vector for the interface DOF
261     %       Apart    : K_ms * (K_ss)^-1 * K_sm   (stored for later reuse)
262     %       bpart    : K_ms * (K_ss)^-1 * f_s    (stored for later reuse)
263     K_mm = K(master,master);
264     K_ss = K(slave ,slave);
265     K_ms = K(master,slave);
266     K_sm = K(slave ,master);
267
268     f_m  = f(master);
269     f_s  = f(slave);
270
271     invKss_Ksm = K_ss \ K_sm;        % (K_ss)^-1 K_sm
272     invKss_fs  = K_ss \ f_s ;        % (K_ss)^-1 f_s
273
274     Apart = K_ms * invKss_Ksm;       % contribution to K_mm
275     bpart = K_ms * invKss_fs ;       % contribution to f_m
276
277     K_hat = K_mm - Apart;            % condensed K
278     f_hat = f_m  - bpart;            % condensed f
279 end
280
281 %% 2.4. Back-substitution of slave DOF
282 function us = backSub(K,f,uMaster,masterIdx,slaveIdx)
283     %   Reconstructs slave displacements after the master solution is known.
284     %   Input
285     %       K           : Global stiffness matrix
286     %       f           : Global load vector
287     %       uMaster     : Displacements of master / interface DOF
288     %       masterIdx   : Row vector with master indices
289     %       slaveIdx    : Row vector with slave  indices
290     %   Output
291     %       us          : Displacement vector of the slave DOF
292     Kss = K(slaveIdx ,slaveIdx );
293     Ksm = K(slaveIdx ,masterIdx);
294     fs  = f(slaveIdx );
295     us  = -(Kss \ (Ksm * uMaster)) + (Kss \ fs);
296 end
297
298
299 %% 2.5. Reading MTX files
300 function  [A,rows,cols,entries,rep,field,symm] = mmread(filename)
301     % Standard function
```

D

ANSYS Python API code for different techniques

D.1. MMMM

```
################################################################################
######################## Setting up finite element model ########################
################################################################################

########################### 0. Importing packages ##################################

import ansys.mapdl.core as pymapdl
import numpy as np
import scipy.io
import matplotlib.pyplot as plt

########################### 1. Defining parameters ##################################

E_shell_values = [2e11, 1e11, 4e11]  # E-moduli (N/m^2)
t_shell = 0.15                       # Shell thickness (m)
l_x_shell = 0.75                     # Length in x-direction per shell element (m)
l_y_shell = 0.75                     # Length in y-direction per shell element (m)

nu_shell = 0.3                       # Poisson's ratio
F_shell = 30000                      # Load magnitude (N)
force_nodes = [17, 20, 23, 26, 29, 62, 65, 68, 71, 74, 107, 110, 113, 116, 119,
    152, 155, 158, 161, 164, 197, 200, 203, 206, 209]  # Node numbers where force
    should be applied (needed to construct force matrix)

n_shell = 15                         # Number of nodes in x- and y-direction

mapdl = pymapdl.launch_mapdl()

for E_shell in E_shell_values:

    file_name_K_SolverOrdering = "filename"
    file_name_F_SolverOrdering = "filename"

    mapdl.clear()
    mapdl.prep7()

    ########################### 2. Material definitions ##########################

```

```
    mapdl.mp("EX", 1, E_shell)
    mapdl.mp("PRXY", 1, nu_shell)

    mapdl.et(1, "SHELL181")
    mapdl.r(1, t_shell)

    ############################## 3. Generate nodes ##############################

    node_ids_shell = []
    for j in range(n_shell):
        for i in range(n_shell):
            nid = j * n_shell + i + 1
            x = i * l_x_shell
            y = j * l_y_shell
            mapdl.n(nid, x, y, 0)
            node_ids_shell.append(nid)

    corner_nodes_shell = [1, n_shell, (n_shell * (n_shell - 1) + 1), (n_shell**2)]

    ############################ 4. Generate elements #############################

    mapdl.mat(1)
    mapdl.type(1)
    mapdl.real(1)
    for row in range(n_shell - 1):
        for col in range(n_shell - 1):
            n1 = row * n_shell + col + 1
            n2 = n1 + 1
            n3 = n1 + n_shell + 1
            n4 = n1 + n_shell
            mapdl.e(n1, n2, n3, n4)

    ######################## 5. Apply boundary conditions #########################

    for b_node in corner_nodes_shell:
        mapdl.d(b_node, "UX", 0)
        mapdl.d(b_node, "UY", 0)
        mapdl.d(b_node, "UZ", 0)

    ################################ 6. Apply loads ###############################

    for force_node in force_nodes:
        mapdl.f(force_node, "FZ", -F_shell)

    ############################### 7. Solve system ###############################

    mapdl.allsel()
    mapdl.run("/SOLU")
    mapdl.antype("STATIC")
    mapdl.run("OUTRES,ALL,ALL")
    mapdl.run("WRFULL")
    mapdl.solve()
    mapdl.finish()

    ###############################################################################
    ######### Retrieving stiffness matrix and force vector as .mtx file #########
    ###############################################################################

    # Retrieves sparse stiffness matrix (solver ordering) for reduced memory use
    mapdl.run("*SMAT, Kmatrix_SolverOrdering, D, import, full, file.full, STIFF")

```

```
    # Retrieves dense force vector (solver ordering)
    mapdl.run("*DMAT, fvector_SolverOrdering, D, import, full, file.full, RHS")

    mapdl.run(f"*EXPORT,Kmatrix_SolverOrdering,mmf,{file_name_K_SolverOrdering}.
        mtx")
    mapdl.run(f"*EXPORT,fvector_SolverOrdering,mmf,{file_name_F_SolverOrdering}.
        mtx")

    ###############################################################################
    ############################ Downloading .MTX Files ############################
    ###############################################################################

    mapdl.download(f"{file_name_K_SolverOrdering}.mtx")
    mapdl.download(f"{file_name_F_SolverOrdering}.mtx")

    ###############################################################################
    #### Reconstruct force vector as force matrixx and download as .txt file ####
    ###############################################################################

    # Load force vector
    f_SO = scipy.io.mmread(f'YOURPATH/{file_name_F_SolverOrdering}.mtx')

    # Convert force vector to force matrix
    nonzero_indices = np.nonzero(f_SO)[0]
    F_SO = np.zeros((len(f_SO), len(nonzero_indices)))

    for col, row in enumerate(nonzero_indices):
        F_SO[row, col] = f_SO[row]

    # Save force matrix as .txt file
    np.savetxt(f'{file_name_F_SolverOrdering}.txt', F_SO)
```

D.2. POD-RBF approximation

```
###############################################################################
###################### Setting up finite element model ########################
###############################################################################

######################### 0. Importing packages ################################

import ansys.mapdl.core as pymapdl
import numpy as np
import scipy.io
import matplotlib.pyplot as plt

######################### 1. Defining parameters ###############################

E_shell_start = 2e11                # Starting value for Young's modulus (N/m^2)
E_shell_end = 4e11                  # Ending value for Young's modulus (N/m^2)
E_shell_n = 3                       # Number of samples for Young's modulus
E_shell_linspace = np.linspace(E_shell_start, E_shell_end, E_shell_n)

nu_shell = 0.3                      # Poisson's ratio
t_shell = 0.15                      # Shell thickness (m)
l_x_shell = 0.75                    # Shell element length in x-direction (m)
l_y_shell = 0.75                    # Shell element length in y-direction (m)
F_shell = 30000                     # Load magnitude (N)
F_loc_shell = 113                   # Node number where point force is applied
n_shell = 15                        # Number of nodes in x- and y-direction

file_name_U_training = "filename"
```

```
file_name_P_training = "filename"
U_training = []
P_training = []

mapdl = pymapdl.launch_mapdl()

for E_shell in E_shell_linspace:

    mapdl.clear()
    mapdl.prep7()

    ########################## 2. Material definitions ##########################
    # Identical to Appendix C.1

    ############################ 3. Generate nodes ############################
    # Identical to Appendix C.1

    ########################### 4. Generate elements ###########################
    # Identical to Appendix C.1

    ###################### 5. Apply boundary conditions ######################
    # Identical to Appendix C.1

    ############################### 6. Apply loads ###############################

    mapdl.f(F_loc_shell, "FZ", -F_shell)

    ############################### 7. Solve system ##############################
    # Identical to Appendix C.1

    ##############################################################################
    ###################### Retrieve displacement vector ##########################
    ##############################################################################

    mapdl.post1()

    displacements_x = mapdl.post_processing.nodal_displacement('X')
    displacements_y = mapdl.post_processing.nodal_displacement('Y')
    displacements_z = mapdl.post_processing.nodal_displacement('Z')
    rotations_x = mapdl.post_processing.nodal_rotation('X')
    rotations_y = mapdl.post_processing.nodal_rotation('Y')
    rotations_z = mapdl.post_processing.nodal_rotation('Z')

    u_array = np.array([val for tup in zip(displacements_x, displacements_y,
        displacements_z, rotations_x, rotations_y, rotations_z) for val in tup])

    ##############################################################################
    ######### Constructing displacement matrix U and parameter matrix P ########
    ##############################################################################

    U_training.append(u_array)
    P_training.append([E_shell, t_shell])

U_training_matrix_T = np.array(U_training)
U_training_matrix = U_training_matrix_T.T
P_training_matrix_T = np.array(P_training)
P_training_matrix = P_training_matrix_T.T

# Save both matrices to .txt files
np.savetxt(f"{file_name_U_training}.txt", U_training_matrix)
np.savetxt(f"{file_name_P_training}.txt", P_training_matrix)
```

D.3. MMMM adjusted for local variations

```

################################################################################
#################### Setting up finite element model ###########################
################################################################################

############################## 0. Importing packages ###############################

import ansys.mapdl.core as pymapdl
import numpy as np
import scipy.io
import matplotlib.pyplot as plt

# ######################### 1. Defining parameters ################################
E_shell1 = 2e11             # 'Youngs modulus - material 1  (N/m^2)
E_shell2 = 4e11             # 'Youngs modulus - material 2  (N/m^2)
t_shell  = 0.15             # Shell thickness (m)

l_x_shell = 0.75            # Length in x- direction per shell element (m)
l_y_shell = 0.75            # Length in y- direction per shell element (m)

nu_shell = 0.3              # Poisson ratio
F_shell  = 30000            # Load magnitude (N)

force_nodes = [3, 31, 33, 113]   # Nodes that receive point loads
n_shell = 15                     # Number of nodes in x- and y- direction

# Elements that should use material 2 in loop 2
elements_mat2 = [1, 2, 15, 16]

mapdl = pymapdl.launch_mapdl()

for loop in range(3):

    file_K = f"K_2DShell_n{n_shell}_K{loop}_E"
    file_F = f"F_2DShell_n{n_shell}_K{loop}_E"

    mapdl.clear()
    mapdl.prep7()

    ######################### 2. Material definitions ############################
    if loop == 0:
        mapdl.et(1, "SHELL181")
        mapdl.r(1, t_shell)
        mapdl.mp("EX", 1, E_shell1)
        mapdl.mp("PRXY", 1, nu_shell)

    if loop == 1:
        mapdl.et(1, "SHELL181")
        mapdl.r(1, t_shell)
        mapdl.mp("EX", 1, E_shell2)
        mapdl.mp("PRXY", 1, nu_shell)

    if loop == 2:
        mapdl.et(1, "SHELL181")
        mapdl.r(1, t_shell)
        mapdl.mp("EX", 1, E_shell1)
        mapdl.mp("PRXY", 1, nu_shell)

        mapdl.et(2, "SHELL181")
        mapdl.r(2, t_shell)
```

```
        mapdl.mp("EX", 2, E_shell2)
        mapdl.mp("PRXY", 2, nu_shell)

    ############################ 3. Generate nodes ############################
    # Identical to Appendix C.1

    ########################## 4. Generate elements ###########################
    if loop == 0:
        mapdl.type(1)
        mapdl.mat(1)
        mapdl.real(1)
        for row in range(n_shell - 1):
            for col in range(n_shell - 1):
                n1 = row * n_shell + col + 1
                n2 = n1 + 1
                n3 = n1 + n_shell + 1
                n4 = n1 + n_shell
                mapdl.e(n1, n2, n3, n4)

    if loop == 1:
        mapdl.type(1)
        mapdl.mat(1)
        mapdl.real(1)
        for row in range(n_shell - 1):
            for col in range(n_shell - 1):
                n1 = row * n_shell + col + 1
                n2 = n1 + 1
                n3 = n1 + n_shell + 1
                n4 = n1 + n_shell
                mapdl.e(n1, n2, n3, n4)

    if loop == 2:
        mapdl.real(1)
        element_counter = 1
        for row in range(n_shell - 1):
            for col in range(n_shell - 1):
                n1 = row * n_shell + col + 1
                n2 = n1 + 1
                n3 = n1 + n_shell + 1
                n4 = n1 + n_shell

                if element_counter in elements_mat2:
                    mapdl.mat(2)  # gebruik materiaal 2
                else:
                    mapdl.mat(1)  # gebruik standaard materiaal 1

                mapdl.e(n1, n2, n3, n4)
                element_counter += 1

    ###################### 5. Apply boundary conditions #######################
    # Identical to Appendix C.1

    ################################ 6. Apply loads ###############################
    for force_node in force_nodes:
        mapdl.f(force_node, "FZ", -F_shell)
        mapdl.f(force_node, "FY", -F_shell)
        mapdl.f(force_node, "FX", -F_shell)
        mapdl.f(force_node, "MX", 1000)  # Moment rond de X-as (in N·m)
        mapdl.f(force_node, "MY", 1000)  # Moment rond de Y-as
        mapdl.f(force_node, "MZ", 1000)  # Moment rond de Z-as

```

```
122
123     ############################### 7. Solve system ##############################
124     # Identical to Appendix C.1
125
126     ##############################################################################
127     ######## Retrieving stiffness matrix and force vector as .mtx file #########
128     ##############################################################################
129
130      # Identical to Appendix C.1
131
132     ##############################################################################
133     ########################### Downloading .MTX Files ###########################
134     ##############################################################################
135
136      # Identical to Appendix C.1
137
138     ##############################################################################
139     ### Reconstruct force vector as force matrixx and download as .txt file ####
140     ##############################################################################
141
142      # Identical to Appendix C.1
```

E

Results of technique applications to the case 2 model

E.1. MMMM

A total of 25 forces are applied to the shell, which are included in the setup of $\mathbf{V}$ in the MMMM. How these forces are applied is shown in Figure E.1. The figure also shows the standard parameters from which variations are made.

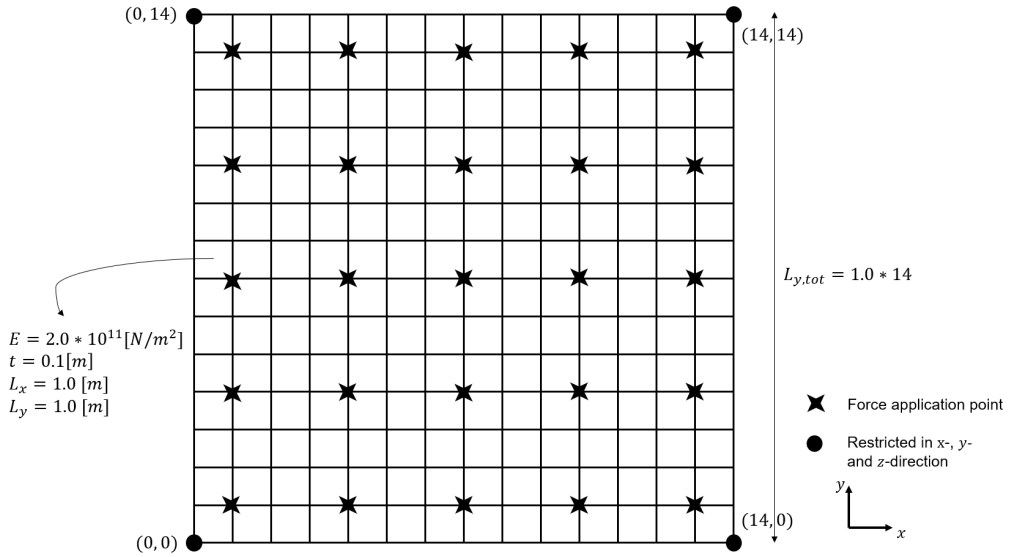

Figure E.1: Case 2 model for the MMMM.

In Table E.1, below the median and maximum errors for different changing parameters, various placements and magnitudes of F, and different numbers of columns in the projector matrix $\mathbf{V}$ can be seen. In the table, the second column indicates the location where the force is applied when using the MMMM. This follows the grid from Figure E.1, where (7,7) corresponds to the center of the plate. The third column shows whether the force location matches a force vector used in constructing $\mathbf{V}$. In Figure E.1, a force is applied at (7,7) during the construction of $\mathbf{V}$, so the answer in this case is 'yes'. The fourth column shows the force in the z-direction, while the fifth and sixth columns indicate the number of columns in $\mathbf{V}$ and the relative error compared to the direct solution.

Modified Parameters	Force Location(s) on Grid (x, y)	Force Location Corresponds with Force Vector used in V	F_z on Application Nodes [kN]	Number of Columns in V	Median Error [%]	Max Error [%]
E *2	(7,7)	Yes	30	49	*8.52E-10*	*8.93E-10*
E *½	(7,7)	Yes	30	49	*4.15E-10*	*4.31E-10*
t *2	(7,7)	Yes	30	49	*1.10E-01*	*1.56*
t *1.1	(7,7)	Yes	30	49	*1.40E-02*	*1.20E-01*
L_x *1.1	(7,7)	Yes	30	49	*57.98*	*68.72*
L_y *1.1	(7,7)	Yes	30	49	*57.98*	*68.72*
E *2, t *2	(7,7)	Yes	30	49	*0.31*	*4.52*
E *½, t *½	(7,7)	Yes	30	49	*0.29*	*4.44*
E *½	(0,7)	No	30	49	*4.75*	*22.73*
E *½	(1,7)	Yes	30	49	*5.00E-10*	*5.24E-10*
E *½	(2,7)	No	30	49	*10.25*	*34.12*
E *½	(3,7)	No	30	49	*4.52*	*23.39*
E *½	(4,7)	Yes	30	49	*1.11E-09*	*1.23E-09*
E *½	(5,7)	No	30	49	*1.41*	*14.93*
E *½	(6,7)	No	30	49	*1.27*	*13.38*
E *½	(7,7)	Yes	30	49	*4.15E-10*	*4.23E-10*
E *½	(1,7), (4,7), (7,7)	Yes	30	49	*4.47E-09*	*4.56E-09*
E *½	(2,7), (3,7), (6,7)	No	30	49	*1.66*	*17.56*
E *½	(7,7)	Yes	30	49	*4.15E-10*	*4.31E-10*
E *½	(7,7)	Yes	10	49	*4.15E-10*	*4.31E-10*
E *½	(7,7)	Yes	200	49	*4.15E-10*	*4.31E-10*
t *½	(3,7)	No	30	49	*5.41*	*27.81*
t *½	(3,7)	No	30	101	*3.75*	*26.89*
t *½	(3,7)	No	30	220	*2.12*	*26.67*
t *½	(3,7)	No	30	333	*0.39*	*26.54*

Table E.1: Results of the MMMM on the case 2 model for varying different parameters, force locations and magnitudes, and number of columns in the projection matrix.

E.2. POD-RBF approximation

The case is shown in Figure E.2. The difference with structure used for applying the MMMM is the placement of the force. For POD-RBF, no force matrix is needed, so only a single force is applied. In the standard case, this force is placed in the middle of the shell but the location of this force in the x-direction will also be treated as a parameter in the tests. In Table E.2, the errors are shown for different changing parameters, various numbers of interpolation points, different cut-off values, and varying test values.

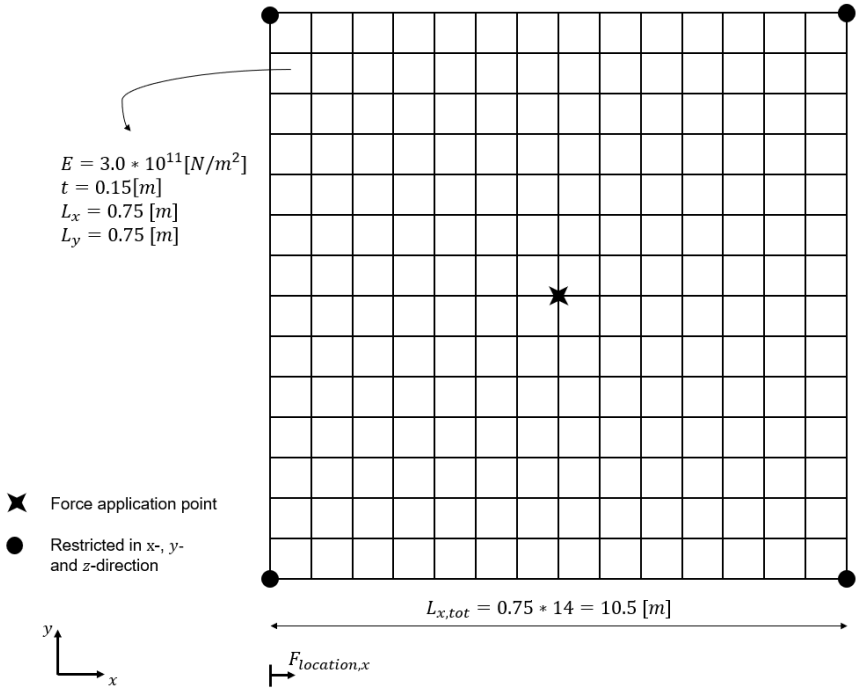

Figure E.2: Case 2 model for POD-RBF approximation.

Parameter	Unit	Range	Interpolation points per parameter	Total number of interpolation points	K (cut-off)	Test value	Median Error [%]	Max Error [%]
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	3	3	3.50E+11	*2.08*	*2.08*
t	[m]	0.10 ~ 0.20	3	3	3	0.175	*7.88*	*19.21*
L_x	[m]	0.50 ~ 1.00	3	3	3	0.875	*2.51*	*6.50*
L_y	[m]	0.50 ~ 1.00	3	3	3	0.875	*2.51*	*6.50*
F_{size}	[kN]	20.00 ~40.00	3	3	3	35.00	*3.76E-11*	*3.89E-11*
$F_{location,\ x}$	[m]	0.75 ~9.75	3	3	3	7.5	*14.98*	*146.32*
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	9	3	3.50E+11	*22.19*	*35.44*
t	[m]	0.10 ~ 0.20	3			0.175		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	81	3	3.50E+11	*41.46*	*71.61*
t	[m]	0.10 ~ 0.20	3			0.175		
L_x	[m]	0.50 ~ 1.00	3			0.875		
L_y	[m]	0.50 ~ 1.00	3			0.875		
$F_{location,\ x}$	[m]	0.75 ~9.75	3	3	3	7.5	*14.98*	*146.32*
$F_{location,\ x}$	[m]	0.75 ~9.75	7	7	3	7.5	*7.93*	*87.65*
E	[N/m^2]	2.00E+11 ~ 4.00E+11	3	9	3	3.50E+11	*22.19*	*35.44*
t	[m]	0.10 ~ 0.20	3			0.175		
E	[N/m^2]	2.00E+11 ~ 4.00E+11	7	49	3	3.50E+11	*1.34*	*2.91*
t	[m]	0.10 ~ 0.20	7			0.175		
t	[m]	0.10 ~ 0.20	7	7	1	0.175	*1.46*	*7.46*
t	[m]	0.10 ~ 0.20	7	7	2	0.175	*1.27*	*3.37*
t	[m]	0.10 ~ 0.20	7	7	3	0.175	*1.24*	*2.77*
t	[m]	0.10 ~ 0.20	7	7	4	0.175	*1.24*	*2.77*
t	[m]	0.10 ~ 0.20	7	7	5	0.175	*1.24*	*2.77*
t	[m]	0.10 ~ 0.20	3	3	3	0.15	*2.60E-08*	*2.68E-08*
t	[m]	0.10 ~ 0.20	3	3	3	0.16	*5.32*	*16.32*
t	[m]	0.10 ~ 0.20	3	3	3	0.175	*7.88*	*19.21*
t	[m]	0.10 ~ 0.20	3	3	3	0.19	*4.22*	*14.22*
t	[m]	0.10 ~ 0.20	3	3	3	0.20	*1.72E-07*	*1.98E-07*
$F_{location,\ x}$	[m]	0.75 ~9.75	3	3	3	0.75	*1.27E-10*	*1.27E-10*
$F_{location,\ x}$	[m]	0.75 ~9.75	3	3	3	3.00	*13.94*	*132.38*
$F_{location,\ x}$	[m]	0.75 ~9.75	3	3	3	5.25	*1.38E-10*	*1.38E-10*

Table E.2: Results of POD-RBF approximation on the case 2 model for varying different parameters, varying number of interpolation points, varying K-value, and varying test values.

F

Analysis of parameter dependencies in stiffness matrices

The performance of the MMMM strongly depends on how certain parameters appear in the stiffness matrix. How these parameters appear in the stiffness matrix of different ANSYS elements is not available in the literature or ANSYS manuals. Therefore, a quick method has been developed to check how certain parameters appear in the stiffness matrix. The following steps should be followed to find out how the parameters appear in the stiffness matrix:

1. Create a simple model with a few elements of the element-type you want to investigate and retrieve the stiffness matrix of this model.
2. Now make a change to the parameter you want to examine. An easy way is to double the value of the parameter. Now retrieve the stiffness matrix again.
3. Divide all nonzero values of the two stiffness matrices by each other and examine the ratios.

If a value in this matrix is two, the relationship in these entries is linear and the parameter appears as p. However, if the value in an entry is 4 or 8, the parameter appears there as p^2 or p^3. For the SHELL181 element, which is also used in example 2 in Chapter 4.1.2, the ratios can be seen in Figure F.1. From this, it can be concluded that E appears linearly in all entries, t appears mostly linearly in the entries but in some places as t^3 and L_x is highly nonlinear.

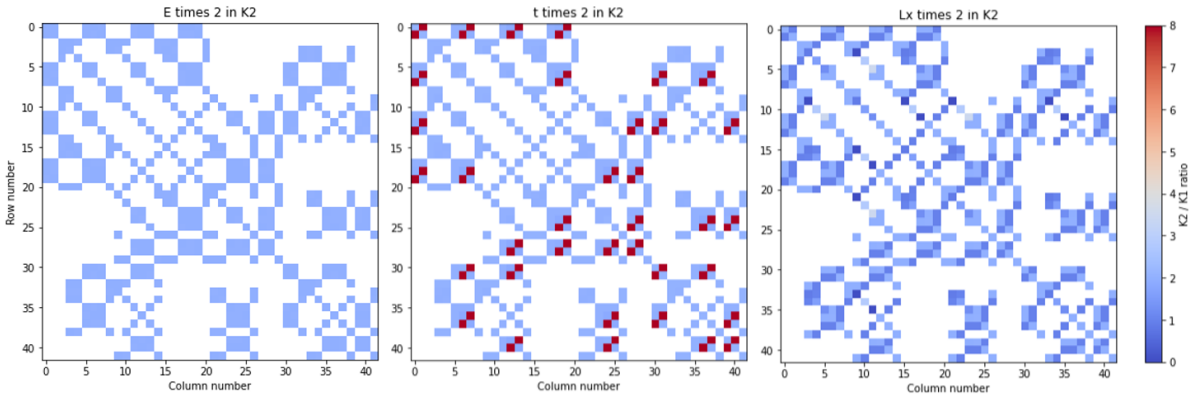

Figure F.1: Comparison of entries in stiffness matrices for a doubling of E, t, and L_x for SHELL181 elements.

G

ANSYS APDL code used

G.1. Substructuring

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  EXPORT OF FEM DATA FOR SUBSTRUCTURING IN MATLAB                              !
!  ------------------------------------------------------------------------     !
!  Purpose                                                                      !
!    - Export global stiffness K, element-wise stiffness Ke, and load          !
!      vector f in Matrix-Market (*.mtx).                                       !
!    - Write solver-ordering to internal-ordering maps (SO2IO, IO2UO).          !
!    - Export node coordinates (node_info) and element connectivity.            !
!                                                                               !
!  File naming convention:                                                      !
!    K_SO<i>.mtx         : Stiffness matrix for solver-ordering i               !
!    K_e_SO<i>.mtx       : Element stiffness matrices                           !
!    f_SO<i>.mtx         : Global load vector (RHS)                             !
!    SO2IO<i>.txt        : Solver-ordering to internal-ordering map             !
!    IO2UO<i>.txt        : Internal-ordering to user-ordering map               !
!    node_info_<i>.txt   : Node list with coordinates                           !
!    conn<i>.txt         : Element connectivity list                            !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! 0. UNIT SYSTEM
! Workbench project units: Metric (m, kg, N, s, V, A)

! 1. EXPORT NODE COORDINATES
*CFOPEN,'node_info_1','txt',' '
ALLSEL, ALL
*GET,ncount,NODE,,COUNT
*DO,I,1,ncount
    *GET,loc_x,NODE,I,LOC,X
    *GET,loc_y,NODE,I,LOC,Y
    *GET,loc_z,NODE,I,LOC,Z
    *VWRITE,I,loc_x,loc_y,loc_z
    (4E16.8)
*ENDDO
*CFCLOS

! 2. EXPORT ELEMENT CONNECTIVITY
ALLSEL, ALL
*GET,ecount,ELEM,,COUNT
/OUTPUT,conn1,txt
ELIST,1,ecount
```

```
/OUTPUT

! 3. ASSEMLE MATRICES
ALLSEL, ALL
/SOLU
ANTYPE,STATIC,NEW        ! Static structural analysis type
EMATWRITE,YES            ! Store element stiffness matrices (*.emat file)
OUTRES,ALL,ALL
EQSLV,SPARSE             ! Use sparse solver
WRFULL,1                 ! Write matrices, but skip solving solution
SOLVE
FINISH

! 4. IMPORT SOLVER MATRICES INTO APDL ARRAYS
*SMAT, K_SO ,D,IMPORT,FULL,file.full,STIFF    ! Import stiffness matrix
*DMAT, f_SO ,D,IMPORT,FULL,file.full,RHS      ! Import load vector
*DMAT, K_e_SO,D,IMPORT,EMAT,file.emat,STIFF,1 ! Import element stiffness matrices

! 5. EXPORT MATRICES TO MATRIX MARKET FORMAT
*EXPORT, K_SO ,MMF,K_SO1.mtx
*EXPORT, f_SO ,MMF,f_SO1.mtx
*EXPORT, K_e_SO,MMF,K_e_SO1.mtx

! 6. EXPORT DOF ORDERING VECTORS
*SMAT, SO2IO,D,IMPORT,FULL,file.full,NOD2SOLV  ! Solver-order to internal-order
*VEC , IO2UO,I,IMPORT,FULL,file.full,BACK      ! Internal-order to user-order
*PRINT, SO2IO,SO2IO1.txt
*PRINT, IO2UO,IO2UO1.txt
```

G.2. Case study

```

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!  CASE STUDY FACTORY FLOOR SIMULATION                                 !
!  ------------------------------------------------------------------  !
!  Purpose                                                             !
!    - Apply machine loads on factory floor model on multiple locations !
!      using load steps.                                               !
!    - Evaluate displacements at leg supports for various for 1 machine !
!      and an additional machine placed on d_small distance.           !
!    - Export displacements of the nodes under the leg supports.       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! 0. UNIT SYSTEM
! Workbench project units:  Metric (m, kg, N, s, V, A)

! 1. DEFINE PARAMETERS
! 1.1 Variable input parameters
Loc_X                         = ARG1      ! x bottom-left corner of pedestal 1 (=L+1)
H1_plus_H2                    = ARG2      ! total height of second floor
t_floor                       = ARG3      ! floor thickness
floor_offset                  = ARG4      ! offset to align beams and floor
n_machines_x                  = ARG5      ! number of machine locs in x direction
n_machines_y                  = ARG6      ! number of machine locs in y direction

! 1.2 Constant geometry parameters (X inserted; not for public release)
Loc_Y                         = X         ! y bottom-left corner of pedestal 1
delta                         = X         ! spacing between machine locations (=L/2)
ly_p1                         = X         ! pedestal 1 length in y-direction
lx_p1                         = X         ! pedestal 1 length in x-direction
ly_p2                         = X         ! pedestal 2 length in y-direction
```

```
lx_p2                       = X        ! pedestal 2 length  in x-direction
distx_p1_p2                 = X        ! x distance between pedestal 1 and 2

! 1.3 Leg positions relative to bottom-lef corner of pedestal 1
support1_x                  = X
support1_y                  = X
support2_x                  = X
support2_y                  = X
support3_x                  = X
support3_y                  = X
support4_x                  = X
support4_y                  = X
support5_x                  = X
support5_y                  = X

! 2. DEFINE MACRO TO APPLY PEDESTAL LOADS TO THE FLOOR
*CREATE, APPLY_LOAD
ALLSEL, ALL
! Select shell elements. Section IDs are derived from input file.
ESEL, S, TYPE,,43
*DO, SecID, 44, 50
    ESEL, A, TYPE,,SecID
*ENDDO
*DO, SecID, 139, 162
    ESEL, A, TYPE,,SecID
*ENDDO
*DO, SecID, 225, 242
    ESEL, A, TYPE,,SecID
*ENDDO
*DO, SecID, 277, 286
    ESEL, A, TYPE,,SecID
*ENDDO
CM, ShellElem, ELEM       ! Create named selection
! Select elements of pedestal 1 and apply load
ESEL, R, CENT, Y, ARG2-0.01, (ARG2+ly_p1)+0.01
ESEL, U, CENT, X, -0.01, ARG1-0.01
ESEL, U, CENT, X, (ARG1+lx_p1)+0.01, 1000
ESEL, U, CENT, Z, 0, H1_plus_H2-0.01
ESEL, STAT
SFE, ALL, 0, PRES, 1, -X     ! Apply surface pressure load to pedestal 1 (X
    inserted; not for public release)
! Select elements of pedestal 2 and apply load
CMSEL, S, ShellElem       ! Select previously defined shell element selection
ESEL, R, CENT, Y, (ARG2+((ly_p1-ly_p2)/2))-0.01, (ARG2+((ly_p1-ly_p2)/2)+ly_p2)
    +0.01
ESEL, U, CENT, X, -0.01, (ARG1+lx_p1+distx_p1_p2)-0.01
ESEL, U, CENT, X, (ARG1+lx_p1+distx_p1_p2+lx_p2)+0.01, 1000
ESEL, U, CENT, Z, 0, H1_plus_H2-0.01
ESEL, STAT
SFE, ALL, 0, PRES, 1, -X    ! Apply surface pressure load to pedestal 2 (X
    inserted; not for public release)
ALLSEL, ALL
*END

! 3. DEFINE SHELL OFFSETS AND THICKNESSES
/PREP7
*DO, SecID, 43, 50       ! Apply shell properties for floor elements
    SECTYPE,SecID,shell
    SECDATA,t_floor
    SECOFF,user,floor_offset
*ENDDO
```

```
*DO, SecID, 139, 162
    SECTYPE,SecID,shell
    SECDATA,t_floor
    SECOFF,user,floor_offset
*ENDDO
*DO, SecID, 225, 242
    SECTYPE,SecID,shell
    SECDATA,t_floor
    SECOFF,user,floor_offset
*ENDDO
*DO, SecID, 277, 286
    SECTYPE,SecID,shell
    SECDATA,t_floor
    SECOFF,user,floor_offset
*ENDDO

! 4. SOLVER SETTINGS
/SOLU
ANTYPE,STATIC,NEW          ! Define static structural analysis
EQSLV,SPARSE               ! Use sparse equation solver
ALLSEL,ALL

! 5. APPLY LOADS WITH LOAD STEPS
time_value = 1
*DO, i_x, 0, (n_machines_x-1)*delta, delta
    *DO, i_y, 0, (n_machines_y-1)*delta, delta
        SFDELE, ALL, ALL   ! Clear any existing surface loads
        TIME, time_value  ! Initiate load step
        Loc_X_value = Loc_X + i_x
        Loc_Y_value = Loc_Y + i_y
        *USE, APPLY_LOAD, Loc_X_value, Loc_Y_value ! Use macro
        SOLVE
        time_value = time_value + 1
    *ENDDO
*ENDDO

! 6. EXTRACT AND EXPORT RESULTS PER LOAD STEP
/POST1
*CFOPEN,'filename','txt',' ' ! Open text file for writing results
*DO, i_time, 1, n_machines_x*n_machines_y,1  ! Loop through load steps
    SET,i_time,1
    ! Retrieve displacents for all supports on different machine locations
    *DO, i_locx_s1, Loc_X + support1_x, Loc_X + support1_x + (n_machines_x-1),
        delta
        *DO, i_locy_s1, Loc_Y + support1_y, Loc_Y + support1_y + (n_machines_y-1),
             delta
            node_select = NODE(i_locx_s1, i_locy_s1, H1_plus_H2)
            *GET, Disp, NODE, node_select, U, Z
            *VWRITE, Disp
            (1E16.8)
        *ENDDO
    *ENDDO
    *DO, i_locx_s2, Loc_X + support2_x, Loc_X + support2_x + (n_machines_x-1),
        delta
        *DO, i_locy_s2, Loc_Y + support2_y, Loc_Y + support2_y + (n_machines_y-1),
             delta
            node_select = NODE(i_locx_s2, i_locy_s2, H1_plus_H2)
            *GET, Disp, NODE, node_select, U, Z
            *VWRITE, Disp
            (1E16.8)
        *ENDDO
```

```
146     *ENDDO
147     *DO, i_locx_s3, Loc_X + support3_x, Loc_X + support3_x + (n_machines_x-1),
            delta
148         *DO, i_locy_s3, Loc_Y + support3_y, Loc_Y + support3_y + (n_machines_y-1),
                 delta
149             node_select = NODE(i_locx_s3, i_locy_s3, H1_plus_H2)
150             *GET, Disp, NODE, node_select, U, Z
151             *VWRITE, Disp
152             (1E16.8)
153         *ENDDO
154     *ENDDO
155     *DO, i_locx_s4, Loc_X + support4_x, Loc_X + support4_x + (n_machines_x-1),
            delta
156         *DO, i_locy_s4, Loc_Y + support4_y, Loc_Y + support4_y + (n_machines_y-1),
                 delta
157             node_select = NODE(i_locx_s4, i_locy_s4, H1_plus_H2)
158             *GET, Disp, NODE, node_select, U, Z
159             *VWRITE, Disp
160             (1E16.8)
161         *ENDDO
162     *ENDDO
163     *DO, i_locx_s5, Loc_X + support5_x, Loc_X + support5_x + (n_machines_x-1),
            delta
164         *DO, i_locy_s5, Loc_Y + support5_y, Loc_Y + support5_y + (n_machines_y-1),
                 delta
165             node_select = NODE(i_locx_s5, i_locy_s5, H1_plus_H2)
166             *GET, Disp, NODE, node_select, U, Z
167             *VWRITE, Disp
168             (1E16.8)
169         *ENDDO
170     *ENDDO
171 *ENDDO
172 *CFCLOS
```

H

MATLAB tool developed in case study

```
%%%%%%%%%%%%%%%%%%%
%% GENERAL CODE %%
%%%%%%%%%%%%%%%%%%%

%% 1. Define parameters
% Flexible parameters (X inserted; not for public release)
w_frame = X;     % Thickness frame (m) (Choose value between X and X)
t_floor = X;     % Thickness floor (m) (Choose value between X and X)

% Parameters with fixed values (X inserted; not for public release)
L = X;             % Distance columns (m) (Choose X, X, X, X, X, X, or X)
H_2 = X;           % Height of column on 2nd floor (m) (Choose X or X)

% Machine locations (X inserted; not for public release)
x_machine_1 = X;   % x 1st machine (m) (Choose between L+1 and 2*L, steps of 1)
y_machine_1 = X;   % y 2nd machine (m) (Choose between 2 and 2*L+7, steps of 1)
x_machine_2 = X;   % x 1st machine (m) (Choose between L+1 and 2*L, steps of 1)
y_machine_2 = X;  % y 2nd machine (m) (Choose between 2 and 2*L+7, steps of 1)

% POD-RBF variables
K_threshold = 0.99;       % Minimum percentage of mode weight selection
K_min_cutoff_value = 25;  % Minimum number of modes (25 gives max accuracy)

% Filepath definition
file_path_def = 'filepath';

%% 2. Perform parameter check (X inserted; not for public release)
valid_L = X:X:X; % Valid values of L
valid_H_2 = [X, X]; % Valid values of H_2
if ~ismember(L, valid_L)
    error('Invalid value for L. Choose X, X, X, X, X, X, or X');
end
if ~ismember(H_2, valid_H_2)
    error('Invalid value for H_2. Choose X or X.');
end
if w_frame < X || w_frame > X
    error('Invalid value for w_frame. It must be between X and X.');
end
if t_floor < X || t_floor > X
    error('Invalid value for t_floor. It must be between X and X.');
end

```

```
%% 3. Retrieve neccesary data files
% Select only the files containing the displ. for the chosen values of L and H_2
L_index = find(valid_L == L);
H_2_index = find(valid_H_2 == H_2);
set_index = (L_index - 1) * 2 + H_2_index;
start_idx = (set_index - 1) * 25 + 1;
end_idx = set_index * 25;
set = start_idx:end_idx;
fprintf('Retrieved data files %d: Files %d to %d\n', set_index, start_idx, end_idx);
all_data = load_displacements_files(set, file_path_def);

%% 4. Edit data files to make extracting data more convinient
% Split each displacement vector into a matrix where the columns represent different load steps and displacements of the 5 legs are stacked vertically.
n_files = numel(all_data);
data_matrix = cell(1, n_files);
for i = 1:n_files
    vec_len = length(all_data{i});
    block_length = sqrt(vec_len / 5) * 5;
    if mod(block_length, 1) ~= 0
        error('Computed block_length is not an integer for file %d. Check input data.', i);
    end
    data_matrix{i} = split_vector_to_matrix(all_data{i}, block_length);
end

% Optional: Display displacements data matrix
% displacements_plot(data_matrix{1})

% Combine matrices for the 25 different w_frame and t_floor combinations.
combined_data_matrix = combine_data_matrices(data_matrix, 25);

% Optional: Display displacements combined data matrix
% displacements_plot(combined_data_matrix{1})

%% 5. Perform POD-RBF approximation
% Retrieve location indices from the coordinate data in order to extract the correct displacements from the combined data matrix.
location_index_machine_1 = get_location_index_from_coordinates(x_machine_1, y_machine_1, L);
location_index_machine_2 = get_location_index_from_coordinates(x_machine_2, y_machine_2, L);

% Determine U for the two machine locations
U_loc1 = generate_U_matrix(combined_data_matrix{1}, location_index_machine_1);
U_loc2 = generate_U_matrix(combined_data_matrix{1}, location_index_machine_2);

% Optional: Display displacements U
% plot_U_displacements(U_loc1)

% Generate A_hat and Phi
[A_hat_loc1, Phi_loc1] = generate_A_hat_and_Phi(U_loc1, K_threshold, K_min_cutoff_value);
[A_hat_loc2, Phi_loc2] = generate_A_hat_and_Phi(U_loc2, K_threshold, K_min_cutoff_value);

% Setting up P (X inserted; not for public release)
w_frame_vals = X:X:X;
t_floor_vals = X:X:X;
[W, T] = meshgrid(w_frame_vals, t_floor_vals);
```

```
P = [W(:)'; T(:)'];

% Generate B
B_loc1 = generate_B(A_hat_loc1, P);
B_loc2 = generate_B(A_hat_loc2, P);

% Generate g (X inserted; not for public release)
P_X = [(w_frame - X) / (X - X); (t_floor - X) / (X - X)];
g = generate_g(P, P_X);

% Generate PODRBF displacement
u_PODRBF_loc1 = Phi_loc1 * B_loc1 * g;
u_PODRBF_loc2 = Phi_loc2 * B_loc2 * g;

% Determine stiffnesses and rotations
extracting_displacements(u_PODRBF_loc1, u_PODRBF_loc2, location_index_machine_1,
    location_index_machine_2)

%%%%%%%%%%%%%%%%
%% FUNCTIONS %%
%%%%%%%%%%%%%%%%

%% 1. Retrieving data
function displacements_data = load_displacements_files(range_i, base_path)
    % This function loads a set of displacement files based on given indices.
    % Each file contains a vector of displacement values for a certain
        configuration.
    num_files = numel(range_i);
    displacements_data = cell(1, num_files);
    for idx = 1:num_files
        i = range_i(idx);
        filename = sprintf('displacements_final_%d.txt', i);
        full_path = fullfile(base_path, filename);
        if ~isfile(full_path)
            error('File not found: %s', full_path);
        end
        displacements_data{idx} = load(full_path);
    end
end

%% 2. Reordering data to a matrix
function result_matrix = split_vector_to_matrix(single_data_vector, block_length)
    % This function restructures a long vector of displacements into a matrix
        format.
    % Each column corresponds to a load step and each row to a reordered
        displacement.
    if mod(length(single_data_vector), block_length) ~= 0
        error('Error in vector length');
    end
    num_blocks = length(single_data_vector) / block_length;
    reshaped = reshape(single_data_vector, block_length, num_blocks);
    n_pootjes = 5;
    n_locs = block_length / n_pootjes;
    result_matrix = zeros(block_length, num_blocks);
    for col = 1:num_blocks
        block = reshaped(:, col);
        reordered = zeros(block_length, 1);
        idx = 1;
        for loc = 1:n_locs
            for poot = 0:n_pootjes-1
                reordered(idx) = block(loc + poot*n_locs);
```

```
153                 idx = idx + 1;
154             end
155         end
156         result_matrix(:, col) = reordered;
157     end
158 end
159 
160 %% 3. Displaying displacements
161 function displacements_plot(data_matrix)
162     % This function visualizes the absolute magnitude of displacements in a
            heatmap.
163     % It is used for inspecting the structure and pattern of displacement data.
164     n_locs = size(data_matrix, 1) / 5;
165     odd_locs = 1:4:n_locs;
166     yticks_odd = (odd_locs - 1) * 5 + 1;
167     ytick_labels = string(odd_locs);
168     figure;
169     imagesc(abs(data_matrix));
170     colormap("turbo");
171     colorbar;
172     xlabel('Timestep');
173     ylabel('Location-index');
174     title('Absolute displacement magnitude');
175     set(gca, 'YTick', yticks_odd, 'YTickLabel', ytick_labels);
176 end
177 
178 %% 4. Combining data matrices
179 function combined_data_matrix = combine_data_matrices(data_matrix_cell, group_size
        )
180     % This function merges individual data matrices into a combined matrix.
181     % It combines them horizontally for further POD analysis.
182     n_total = numel(data_matrix_cell);
183     n_combined = ceil(n_total / group_size);
184     combined_data_matrix = cell(1, n_combined);
185     for i = 1:n_combined
186         idx_start = (i - 1) * group_size + 1;
187         idx_end = min(i * group_size, n_total);
188         current_group = data_matrix_cell(idx_start:idx_end);
189         row_counts = cellfun(@(x) size(x, 1), current_group);
190         if ~all(row_counts == row_counts(1))
191             error('Not all matrices have the same number of rows');
192         end
193         combined_data_matrix{i} = horzcat(current_group{:});
194     end
195 end
196 
197 %% 5. Obtaining location index (X inserted; not for public release)
198 function location_index = get_location_index_from_coordinates(x_coord, y_coord, L)
199     % This function converts physical x,y machine coordinates into a 1D matrix
            index.
200     % This index is used to retrieve displacement values from the combined data
            matrix.
201 
202     % Table with columns [L, x_coord_start, x_coord_end, y_coord_start,
203     % y_coord_end]
204     table_data = [
205         X,  X,  X,  X, X;
206         X,  X,  X,  X, X;
207         X,  X,  X,  X, X;
208         X,  X,  X,  X, X;
209         X,  X,  X,  X, X;
```

```
        X,  X,  X,  X, X;
        X,  X,  X,  X, X;
    ];
    row_idx = find(table_data(:,1) == L, 1);

    start_x = table_data(row_idx, 2);
    end_x   = table_data(row_idx, 3);
    start_y = table_data(row_idx, 4);
    end_y   = table_data(row_idx, 5);
    x_vals = start_x:1:end_x;
    y_vals = start_y:1:end_y;
    if ~ismember(x_coord, x_vals)
        error('Invalid x_coord %.2f for L = %.1f. Valid x range: %.1f to %.1f,
            with steps of 1', ...
              x_coord, L, start_x, end_x);
    end
    if ~ismember(y_coord, y_vals)
        error('Invalid y_coord %.2f for L = %.1f. Valid y range: %.0f to %.0f,
            with steps of 1', ...
              y_coord, L, start_y, end_y);
    end
    col_index = find(x_vals == x_coord);
    row_index = find(y_vals == y_coord);
    n_rows = length(y_vals);
    location_index = (col_index - 1) * n_rows + row_index;
end

%% 6. Generating U
function U = generate_U_matrix(combined_matrix, location_index)
    % This function extracts columns from the combined data matrix to form the U
        matrix.
    % U contains all displacements for a single location for different w_frame and
         t_floor values.
    [n_rows, n_cols] = size(combined_matrix);
    n_data_matrices = 25;
    if mod(n_cols, n_data_matrices) ~= 0
        error('Number of columns is no multiplication of 25');
    end
    U = zeros(n_rows, n_data_matrices);
    for i = 1:n_data_matrices
        col_idx = (i - 1) * (n_cols / n_data_matrices) + location_index;
        U(:, i) = combined_matrix(:, col_idx);
    end
end

%% 7. Displaying displacements U
function plot_U_displacements(U)
    % This function creates a heatmap to visualize the displacements stored in U.
    % Useful for checking input data quality before applying POD.
    n_locs = size(U, 1) / 5;
    if mod(n_locs, 1) ~= 0
        error('Aantal rijen in U is geen veelvoud van 5');
    end
    odd_locs = 1:4:n_locs;
    yticks_odd = (odd_locs - 1) * 5 + 1;
    ytick_labels = string(odd_locs);
    figure;
    imagesc(abs(U));
    colormap("turbo");
    colorbar;
    xlabel('Timestep');
```

```
267     ylabel('Location-index');
268     title('Absolute displacement magnitude');
269     set(gca, 'YTick', yticks_odd, 'YTickLabel', ytick_labels);
270 end
271 
272 %% 8. Generating A_hat and Phi
273 function [A_hat, Phi] = generate_A_hat_and_Phi(U, K_threshold, K_min_cutoff_value)
274     % This function performs POD.
275     % It returns the modal matrix Phi and the projected amplitudes A_hat.
276     N = size(U, 1);
277     M = size(U, 2);
278     if M < N
279         D = U' * U;
280         [V, LambdaMat] = eig(D);
281         lambdaVec = diag(LambdaMat);
282         [lambdaVec, idx] = sort(lambdaVec,'descend');
283         cumsumlambdaVec = cumsum(lambdaVec);
284         sumlambdaVec = sum(lambdaVec);
285         K_cutoff = find(cumsumlambdaVec/sumlambdaVec >= K_threshold, 1, 'first');
286         if isempty(K_cutoff)
287             K_cutoff = length(lambdaVec);
288         end
289 
290         V = V(:, idx);
291         Phi = zeros(N, M);
292         for i = 1:M
293             lambda_i = lambdaVec(i);
294             v_i = V(:, i);
295             Phi(:, i) = (U * v_i) / sqrt(lambda_i);
296         end
297         Phi = Phi(:,1:max(K_cutoff, K_min_cutoff_value));
298     end
299     if M >= N
300         C = U*U';
301         [V, LambdaMat] = eig(C);
302         lambdaVec = diag(LambdaMat);
303         [lambdaVec, idx] = sort(lambdaVec,'descend');
304 
305         cumsumlambdaVec = cumsum(lambdaVec);
306         sumlambdaVec = sum(lambdaVec);
307         K_cutoff = find(cumsumlambdaVec/sumlambdaVec >= K_threshold, 1, 'first');
308         if isempty(K_cutoff)
309             K_cutoff = length(lambdaVec);
310         end
311         V = V(:, idx);
312         Phi = V(:,1:max(K_cutoff, K_min_cutoff_value));
313     end
314     A_hat = Phi' * U;
315 end
316 
317 %% 9. Generating B
318 function B = generate_B(A, P)
319     % This function calculates the matrix B for RBF interpolation.
320     % It uses normalized distances between parameter vectors as input.
321     for j = 1:size(P,1)
322         minP(j,1) = min(P(j,:));
323         maxP(j,1) = max(P(j,:));
324         for i = 1:size(P,2)
325             x(j,i) = (P(j,i) - minP(j)) / (maxP(j) - minP(j)); % Normalize to
                [0,1]
326         end
```

```
    end
    N = size(x, 2);
    for i = 1:N
        for j = 1:N
            G(i,j) = sum((x(:,i) - x(:,j)).^2).^0.5;
        end
    end
    Bt = inv(G') * A';
    B = Bt';
end

%% 10. Generating g
function g = generate_g(P, P_X)
    % This function creates the interpolation vector g for a new parameter point.
    % The vector g is based on the distance between the new and training points.
    N = size(P,2);
    for j = 1:size(P,1)
        minP(j,1) = min(P(j,:));
        maxP(j,1) = max(P(j,:));
        for i = 1:size(P,2)
            x(j,i) = (P(j,i) - minP(j)) / (maxP(j) - minP(j)); % Normalize to
                [0,1]
        end
    end
    gi = @(x, y) sum((x - y).^2).^0.5; % Euclidean distance
    value = P_X;
    for k = 1:N
        g(k,1) = gi(value, x(:,k));
    end
end

%% 11. Displaying displacements, stiffnesses and roations
function extracting_displacements(u_PODRBF_loc1, u_PODRBF_loc2,
    loc_index_machine_1, loc_index_machine_2)
    % This function extracts displacements at two locations and computes:
    % - Local and combined stiffnesses
    % - Resulting rotations around x and y axes
    % Results are shown in tables for comparison.

    % Index under 5 legs
    rows_m1 = (loc_index_machine_1 - 1) * 5 + (1:5);
    rows_m2 = (loc_index_machine_2 - 1) * 5 + (1:5);

    % displacements
    disp_m1_1 = u_PODRBF_loc1(rows_m1);
    disp_m1_2 = u_PODRBF_loc2(rows_m1);
    disp_m1_total = disp_m1_1 + disp_m1_2;

    disp_m2_1 = u_PODRBF_loc1(rows_m2);
    disp_m2_2 = u_PODRBF_loc2(rows_m2);
    disp_m2_total = disp_m2_1 + disp_m2_2;

    % Forces on legs (given) (X inserted; not for public release)
    forces_on_legs = [X; X; X; X; X];

    % Stiffnesses
    k_m1_1 = abs(forces_on_legs ./ disp_m1_1);
    k_m1_total = abs(forces_on_legs ./ disp_m1_total);
    k_m2_2 = abs(forces_on_legs ./ disp_m2_2);
    k_m2_total = abs(forces_on_legs ./ disp_m2_total);

```

```
    % Rotations in µrad (X inserted; not for public release)
    Rx_m1_1 = ((disp_m1_1(4) - disp_m1_1(5)) / X) * 1e6;
    Ry_m1_1 = ((disp_m1_1(2) - ((disp_m1_1(4) + disp_m1_1(5)) / 2)) / X) * 1e6;
    Rx_m1_total = ((disp_m1_total(4) - disp_m1_total(5)) / X) * 1e6;
    Ry_m1_total = ((disp_m1_total(2) - ((disp_m1_total(4) + disp_m1_total(5)) / X)
        ) / 4.75) * 1e6;

    Rx_m2_2 = ((disp_m2_2(4) - disp_m2_2(5)) / X) * 1e6;
    Ry_m2_2 = ((disp_m2_2(2) - ((disp_m2_2(4) + disp_m2_2(5)) / 2)) / X) * 1e6;
    Rx_m2_total = ((disp_m2_total(4) - disp_m2_total(5)) / X) * 1e6;
    Ry_m2_total = ((disp_m2_total(2) - ((disp_m2_total(4) + disp_m2_total(5)) / X)
        ) / 4.75) * 1e6;

    % --- Table: Displacements ---
    disp_table = table((1:5)', disp_m1_1, disp_m1_2, disp_m1_total, ...
                               disp_m2_1, disp_m2_2, disp_m2_total, ...
        'VariableNames', {'Leg', 'M1_t1', 'M1_t2', 'M1_total', ...
                                  'M2_t1', 'M2_t2', 'M2_total'});
    fprintf('\nDisplacements (m):\n');
    disp(disp_table);

    % --- Table: Stiffnesses ---
    stiffness_table = table((1:5)', k_m1_1, k_m1_total, k_m2_2, k_m2_total, ...
        'VariableNames', {'Leg', 'k_M1_t1', 'k_M1_total', 'k_M2_t2', 'k_M2_total'
            });
    fprintf('\nStiffnesses (N/m):\n');
    disp(stiffness_table);

    % --- Table: Rotations ---
    rot_table = table(["M1_t1"; "M1_total"; "M2_t2"; "M2_total"], ...
                      [Rx_m1_1; Rx_m1_total; Rx_m2_2; Rx_m2_total], ...
                      [Ry_m1_1; Ry_m1_total; Ry_m2_2; Ry_m2_total], ...
        'VariableNames', {'Case', 'Rx_urad', 'Ry_urad'});
    fprintf('\nRotations (µrad):\n');
    disp(rot_table);
end
```

I

Graphs of POD-RBF approximation errors in case study

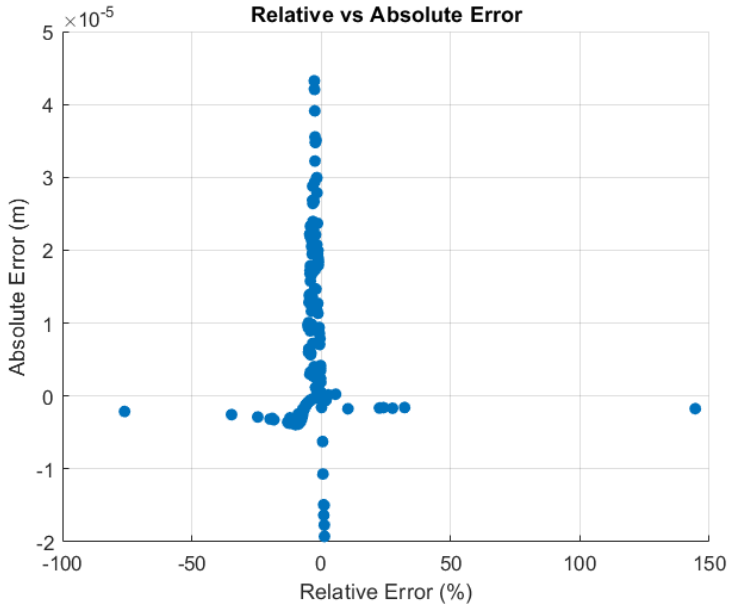

Figure I.1: Relative vs absolute error for $t_{floor} = \mathsf{X}\,[m]$, $w_{frame} = \mathsf{X}\,[m]$, $L = \mathsf{X}\,[m]$, $H_2 = \mathsf{X}\,[m]$, $x_{loc} = \mathsf{X}\,[m]$, $y_{loc} = \mathsf{X}\,[m]$

.

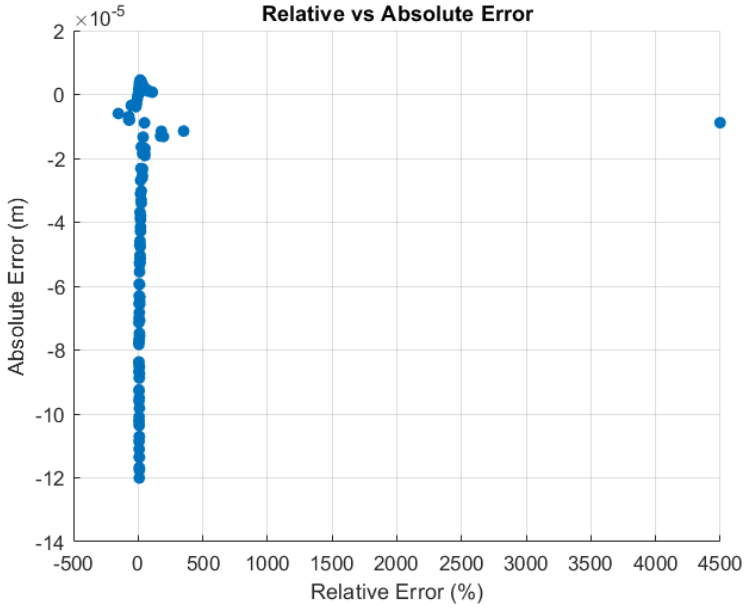

Figure I.2: Relative vs absolute error for $t_{floor} = \mathsf{X}\,[m]$, $w_{frame} = \mathsf{X}\,[m]$, $L = \mathsf{X}\,[m]$, $H_2 = \mathsf{X}\,[m]$, $x_{loc} = \mathsf{X}\,[m]$, $y_{loc} = \mathsf{X}\,[m]$

.

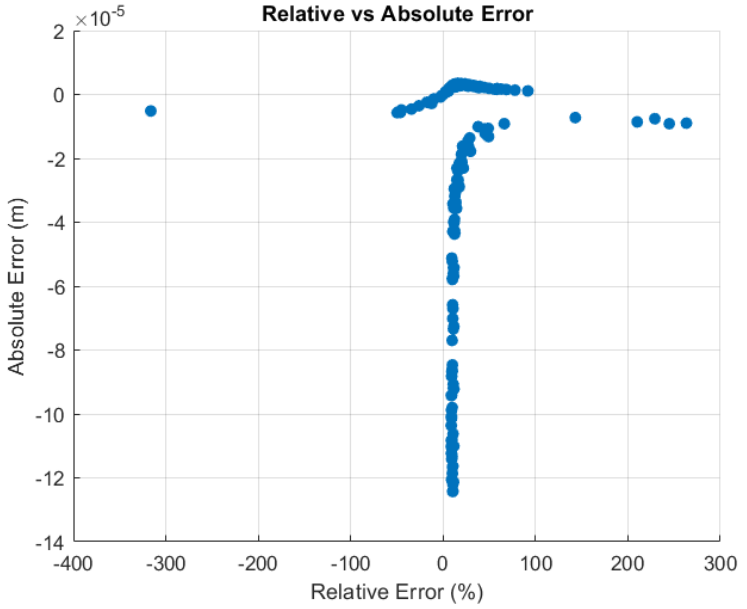

Figure I.3: Relative vs absolute error for $t_{floor} = \mathsf{X}\,[m]$, $w_{frame} = \mathsf{X}\,[m]$, $L = \mathsf{X}\,[m]$, $H_2 = \mathsf{X}\,[m]$, $x_{loc} = \mathsf{X}\,[m]$, $y_{loc} = \mathsf{X}\,[m]$

.

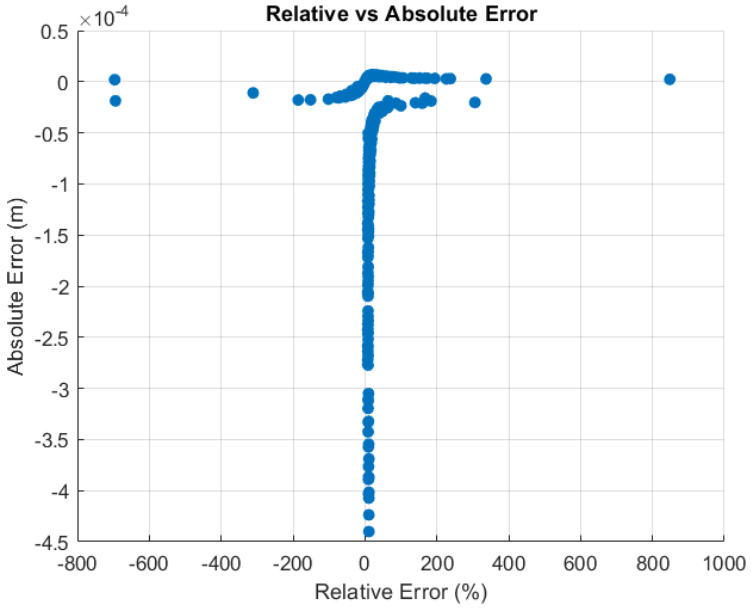

Figure I.4: Relative vs absolute error for $t_{floor} = \mathsf{X}\,[m]$, $w_{frame} = \mathsf{X}\,[m]$, $L = \mathsf{X}\,[m]$, $H_2 = \mathsf{X}\,[m]$, $x_{loc} = \mathsf{X}\,[m]$, $y_{loc} = \mathsf{X}\,[m]$

.

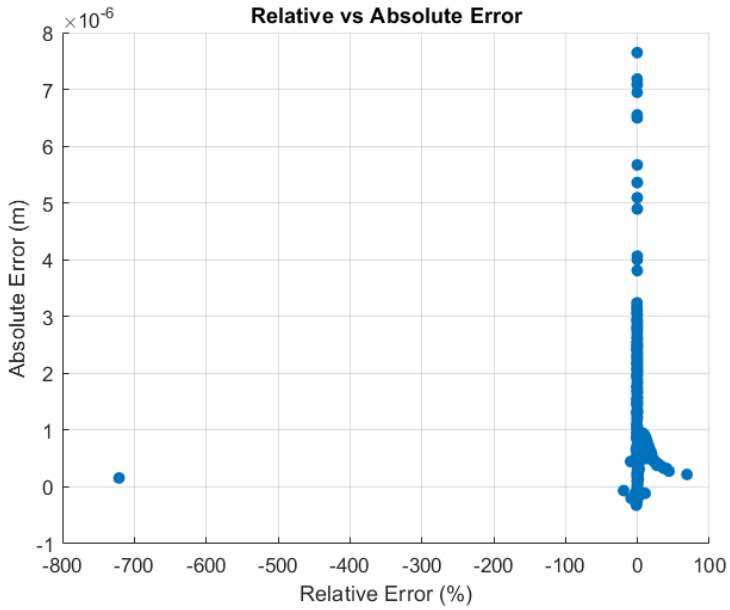

Figure I.5: Relative vs absolute error for $t_{floor} = \mathsf{X}\,[m]$, $w_{frame} = \mathsf{X}\,[m]$, $L = \mathsf{X}\,[m]$, $H_2 = \mathsf{X}\,[m]$, $x_{loc} = \mathsf{X}\,[m]$, $y_{loc} = \mathsf{X}\,[m]$

.

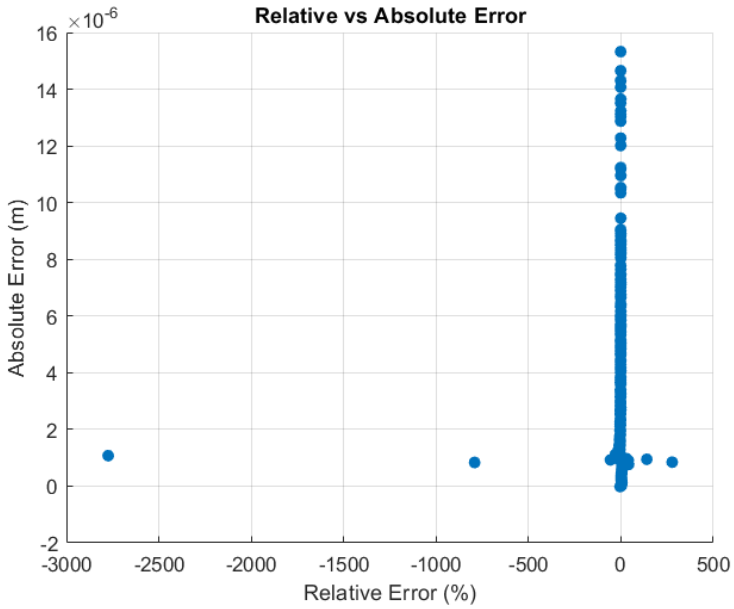

Figure I.6: Relative vs absolute error for $t_{floor} = \mathsf{X}\,[m]$, $w_{frame} = \mathsf{X}\,[m]$, $L = \mathsf{X}\,[m]$, $H_2 = \mathsf{X}\,[m]$, $x_{loc} = \mathsf{X}\,[m]$, $y_{loc} = \mathsf{X}\,[m]$

.

J

Excel tool interface developed in case study

Defining parameters			
Step 1: Define length, height of the 2nd floor, thickness of floor and frame			
Parameters	**Variable name**	**Unit**	**Value**
Length	(L)	[m]	X
Height 2nd floor	(H)	[m]	X
Thickness floor	(t_floor)	[m]	X
Thickness frame	(w_frame)	[m]	X
Step 2: Define machine locations			
Parameters	**Variable name**	**Unit**	**Value**
Machine 1 loc_x	(loc1_x)	[m]	X
Machine 1 loc_y	(loc1_y)	[m]	X
Machine 2 loc_x	(loc2_x)	[m]	X
Machine 2 loc_y	(loc2_y)	[m]	X

Location indeces	
Location index machine 1	X
Location index machine 2	X

Displacements [m]						
Leg	**Machine 1 caused by machine 1**	**Machine 1 caused by machine 2**	**Machine 1 total**	**Machine 2 caused by machine 1**	**Machine 2 caused by machine 2**	**Machine 2 total**
1	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
2	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
3	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
4	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!
5	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!	#VALUE!

Stifnesses [N/m]						
Leg	**Machine 1 caused by machine 1**	**Machine 1 caused by machine 2**	**Machine 1 total**	**Machine 2 caused by machine 1**	**Machine 2 caused by machine 2**	**Machine 2 total**
1	#VALUE!	x	#VALUE!	x	#VALUE!	#VALUE!
2	#VALUE!	x	#VALUE!	x	#VALUE!	#VALUE!
3	#VALUE!	x	#VALUE!	x	#VALUE!	#VALUE!
4	#VALUE!	x	#VALUE!	x	#VALUE!	#VALUE!
5	#VALUE!	x	#VALUE!	x	#VALUE!	#VALUE!

Rotations			
Case	**Rx [urad]**	**Ry [urad]**	**Rt [um]**
Machine 1 caused by machine 1	#VALUE!	#VALUE!	#VALUE!
Machine 1 total	#VALUE!	#VALUE!	#VALUE!
Machine 2 caused by machine 2	#VALUE!	#VALUE!	#VALUE!
Machine 2 total	#VALUE!	#VALUE!	#VALUE!

Figure J.1: Interface of the Excel tool developed for the case study. Yellow cells represent inputs, while blue cells represent outputs (X inserted; not for public release).

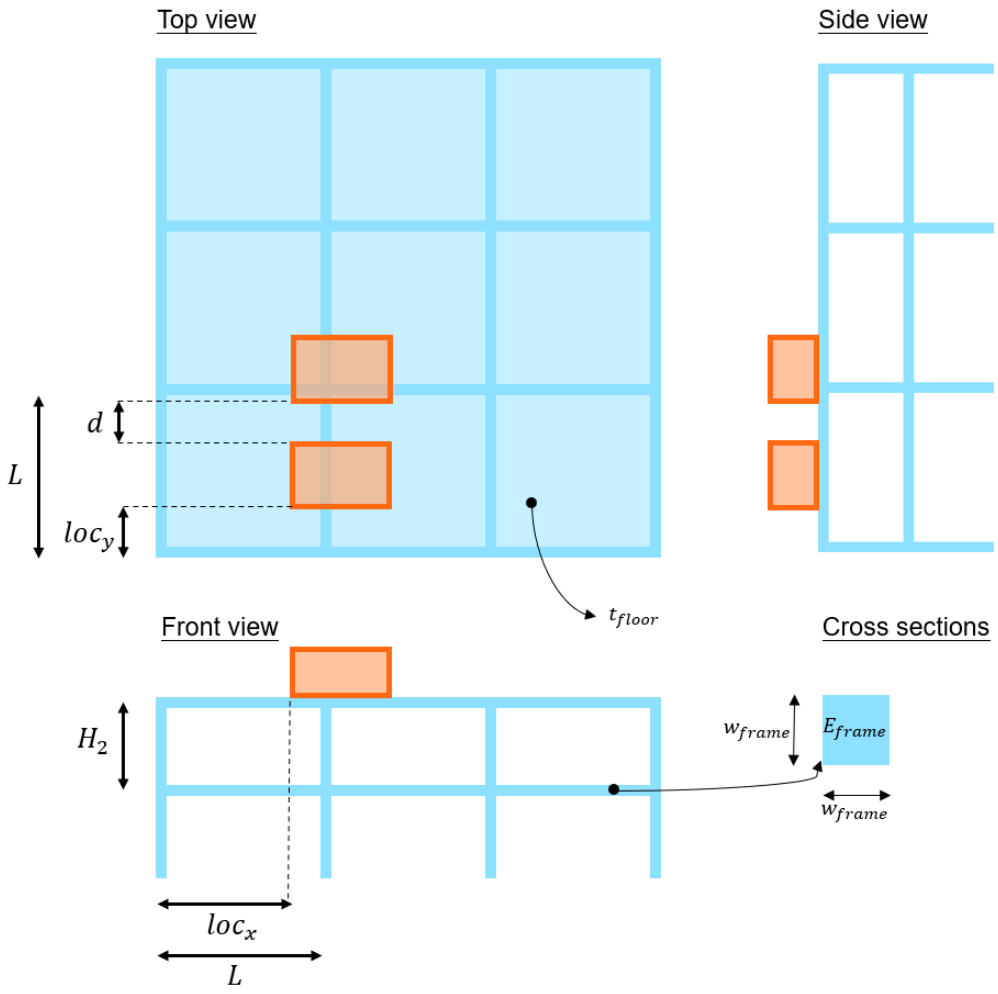

Figure J.2: Supporting figure 1: illustrating the various variable parameters.

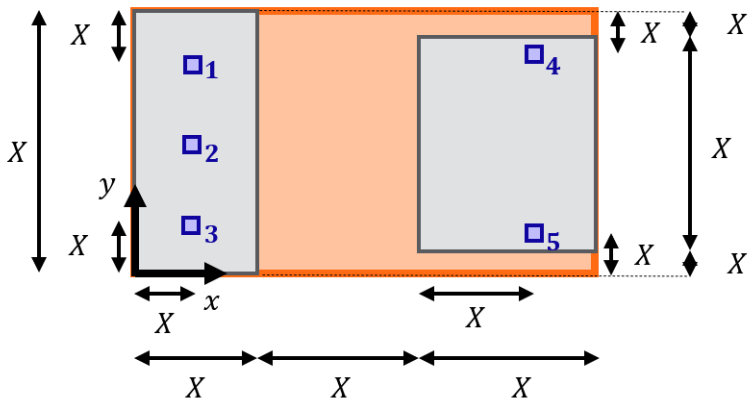

Figure J.3: Supporting figure 2: illustrating the positions of the pedestals and legs (X inserted; not for public release).

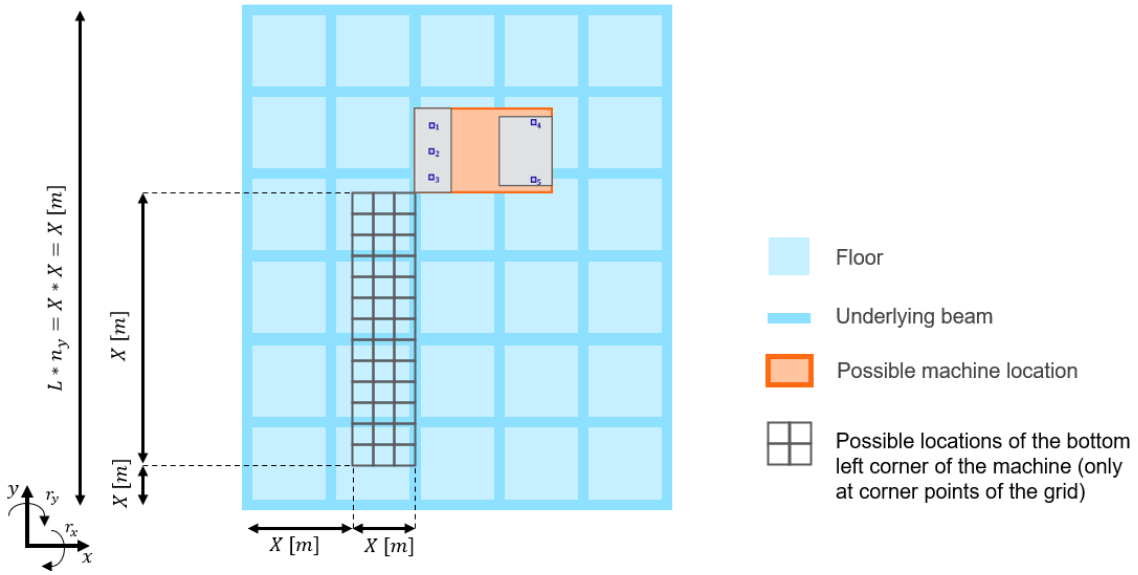

Figure J.4: Supporting figure 3: illustrating the possible machine locations for $L = \mathrm{X}\,[m]$ (X inserted; not for public release).