

**SMART FINITE ELEMENTS: AN APPLICATION OF MACHINE LEARNING
TO REDUCED-ORDER MODELING OF MULTI-SCALE PROBLEMS**

A Ph.D. Thesis
Presented to
The Academic Faculty

By

German Capuano

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of School of Aerospace Engineering

Georgia Institute of Technology

August 2019

Copyright © German Capuano 2019

**SMART FINITE ELEMENTS: AN APPLICATION OF MACHINE LEARNING
TO REDUCED-ORDER MODELING OF MULTI-SCALE PROBLEMS**

Approved by:

Dr. Julian J. Rimoli, Advisor
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Claudio V. Di Leo
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Evangelos Theodorou
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Massimo Ruzzene
School of Aerospace Engineering
Georgia Institute of Technology

Dr. Arash Yavari
School of Civil and Environmental
Engineering
Georgia Institute of Technology

Date Approved: April 25, 2019

ACKNOWLEDGEMENTS

Part of this material is based upon work supported by the National Science Foundation Division of Civil, Mechanical, and Manufacturing Innovation under Grant No. CMMI-1130368.

TABLE OF CONTENTS

| | |
|--|-----|
| Acknowledgments | iii |
| List of Figures | vi |
| Chapter 1: Introduction and Background | 1 |
| Chapter 2: Multi-scale finite element models | 5 |
| 2.1 Formulation | 6 |
| 2.1.1 Boundary shape functions | 8 |
| 2.1.2 Interior shape functions | 10 |
| 2.2 Case studies | 12 |
| 2.2.1 Wave propagation in one-dimensional bars | 12 |
| 2.2.2 Stress wave propagation in a two-dimensional periodic elastic domain | 23 |
| Chapter 3: Smart elements | 30 |
| 3.1 Overview of machine learning | 30 |
| 3.1.1 Linear Regression | 32 |
| 3.1.2 Support Vector Regression | 32 |
| 3.1.3 Gaussian process regression | 33 |
| 3.1.4 Neural Networks | 35 |

| | | |
|-------|--|-----------|
| 3.2 | Formulation | 39 |
| 3.2.1 | Example: GMsFEM vs Linear Regression | 43 |
| 3.2.2 | Introducing physical considerations | 46 |
| 3.3 | Case studies | 50 |
| 3.3.1 | 3D truss structure | 50 |
| 3.3.2 | Nonlinear Multiscale problem | 57 |
| 3.4 | Application to material modeling | 65 |
| 3.5 | History dependent problems | 66 |
| 3.5.1 | Method | 68 |
| 3.5.2 | Input sequence generation | 70 |
| 3.5.3 | Example: Multi-scale Material | 72 |
| 3.5.4 | Example: Multiscale Beam | 79 |
| | Chapter 4: Conclusions | 82 |
| | References | 90 |

LIST OF FIGURES

| | | |
|-----|---|----|
| 2.1 | Schematics of boundary (top row) and interior (bottom row) shape functions for a square element without internal features. The former represent the deformed configuration for unit displacement of the nodes while the latter represents the natural modes of its fine scale model. The vertical axis reflects the corresponding value of the shape functions over the domain. | 8 |
| 2.2 | Schematic of a uniform bar of length L with a displacement $u_0(t)$ applied at the left boundary. | 14 |
| 2.3 | Shape functions for a uniform bar element. The boundary shape functions (dashed lines) correspond to the static deformed configuration when a unit displacement is applied at one of the tips. The interior shape functions (solid lines) are the natural modes of of the bar. | 15 |
| 2.4 | Maximum relative error in the elastic energy for the different models of the uniform bar as function of the ratio η between the the length of the element and the length of the pulse. Solid lines are for curves with the same amount of added modes per element, and dashed lines connect models with the same total number of degrees of freedom for comparison purposes. | 16 |
| 2.5 | Schematic of a notched bar with a displacement $u_0(t)$ applied at the left boundary. | 18 |
| 2.6 | Shape functions for a notched bar element. The boundary shape functions (dashed lines) correspond to the static deformed configuration when a unit displacement is applied at one of the tips. The interior shape functions (solid lines) are the natural modes of the traditional finite element model that corresponds to the fine scale mesh. | 18 |
| 2.7 | Maximum relative error of the displacement field for models with different amount of fine-scale elements per mode (epm), as a function of the total number of degrees of freedom. | 19 |
| 2.8 | Stable time increments for a bar with a notch of length h and 50% reduction of the cross-sectional area. | 21 |

| | | |
|------|--|----|
| 2.9 | Geometry of an elastic plate structure containing a periodic domain and a V-shaped notch. The periodic domain consists of a uniform array of circular holes. The notch cuts through 4 rows of holes and ends in one of them. | 25 |
| 2.10 | Mesh corresponding to the central portion of the structure. On the sides of the Multi-scale element we show a portion of the traditional finite element mesh in the uniform section of the plate. The part of the structure not shown in this figure follows the same uniform mesh pattern. | 26 |
| 2.11 | Example of selected interior shape functions. Each figure displays a natural mode of the fine scale model within the multi-scale element. | 27 |
| 2.12 | Horizontal displacement of the structure produced by the different waves. The results obtained using FEM are compared with those using GMsFEM and MFEM. (a) Complete structure describing the plotted sections in figures b, c, d and e. (b) Incident wave traveling from left to right at $50\mu s$. (c) Displacement field produced during the interaction of the waves with the periodic domain and the notch at $66\mu s$. (d) and (e) Transmitted and Reflected waves respectively at $90\mu s$. Note that there is an antisymmetric component of the wave on the left of figure (d) and on the right of figure (e) Since these antisymmetric components travel at slower speeds than their symmetric counterparts they trail the symmetric waves. | 28 |
| 2.13 | Cascade plots of the displacement as function of time for FEM, GMsFEM, and MFEM. Each line corresponds to the axial displacement observed at a node on the top surface of the plate, at the labeled position from the left tip. The figures show (a) a comparison between MFEM and a reference FEM model, and (b) a comparison between GMsFEM and the same FEM reference model. | 29 |
| 3.1 | Example of a feedforward neural network with three layers. Each node represents an operation $\sigma^l(\sum_k K_{jk}^l x_k^l + b_j^l)$, where K_{jk}^l and b_j^l are the trainable parameters in node j of layer l , and x^l are the inputs for the layer. | 36 |
| 3.2 | Example of a recurrent neural network with three layers. The operations on the nodes are similar to those in a feedforward network except that each the outputs of the recurrent layer are also used as input in the next time increment. | 38 |
| 3.3 | Example of a simplified recurrent neural network and an unrolled version of the same network. h_i represents the hidden variables in the model, and x_i and y_i its inputs and outputs, respectively. | 39 |

| | | |
|------|---|----|
| 3.4 | Example of a geometric multiscale finite element with the fine-scale model represented as a mesh in its interior. | 44 |
| 3.5 | Example of shape functions used in the geometric multiscale finite element method. | 44 |
| 3.6 | Example truss structure with its applied loads and imposed displacements. . | 51 |
| 3.7 | Stress-strain relationship of the material in the truss structure. Adapted from Kirchdoefer et. al. [30]. | 52 |
| 3.8 | Heatmaps for hyperparameter fine tuning. The colormap indicates the R^2 score for a test dataset of size 10000 and the value inside each box indicates the sparsity of the solution. The R^2 values have been limited to a minimum of 0 in the color visualization. | 54 |
| 3.9 | RMS Error of the stress in the structure for different values of ϵ and γ depending on the size of the training sample N . Each point represents the average over the converged models obtained from 10 different training samples. | 55 |
| 3.10 | Comparison of the RMS Error of the stress in the structure ϵ_{RMS}^σ for smart elements (SE) and data-driven computational mechanics (DDCM) as function of the training sample size N . The output in the training samples have a normally distributed error with a standard deviation σ_f . Each point in the plot represents the average error from 10 models with different training samples. | 56 |
| 3.11 | A short bar used in example 3.3.2. It contains a uniform array of 5 by 10 holes of diameter 0.4 cm. The mesh corresponding to this geometry does not contain the holes since they are considered part of the fine scale. | 58 |
| | (a) Geometry | 58 |
| | (b) Mesh | 58 |
| 3.12 | Comparison of the deformed configuration of a multiscale finite element model and a smart model for the three load cases in three different time instants. The first is displayed as a colored surface while the latter is displayed as a black wireframe. The shown smart model corresponds to ordinary polynomial regression of order 6 (Ord 6). Note that in the figure the exact MsFE solutions and the smart element solutions are superimposed. . . | 59 |
| 3.13 | Maximum Relative Error ϵ_{max}^u in the displacements for different smart element types and load scenarios. | 62 |

| | | |
|------|---|----|
| 3.14 | Comparison of the Maximum Relative Error ϵ_{max}^u in the displacements of the smart elements when including or excluding consideration of the material properties in the input vector. | 64 |
| 3.15 | Illustration of a 2D plane strain finite element mesh. It contains two materials: a soft elastoplastic matrix (red) and a stiffer elastic particle (blue). . . | 73 |
| 3.16 | Example of input and output sequences for a single sample in the training set. (left) Input sequences. The curves represent the average strain components (e11, e22, e12) as a function of the time. (right) Output sequences. The curves represent the average stress components (s11, s22, s12), the maximum equivalent plastic strain (pe) and the maximum von Mises stress (mises) as a function of the time. | 74 |
| 3.17 | Error of recurrent neural networks trained on material data for different layer sizes and number of training epochs. In all cases, the error indicated for epoch i represents the best score achieved during training until that epoch. | 75 |
| 3.18 | Selected output components test samples 1 and 2 compared with the machine learning prediction. | 77 |
| 3.19 | Selected output components for test samples 3 and 4 compared with the machine learning prediction. | 78 |
| 3.20 | Comparison of the maximum von Mises stress in the fine scale of two multiscale finite element models. (right) Nonlinear multiscale finite element. (left) Surrogate model using recurrent neural networks. The displacement has been scaled 10 times for visualization purposes in both cases. | 80 |
| 3.21 | Comparison of the values observed on the elements of two multi-scale finite element models. One possesses a mutli-scale material that contains a high-fidelity (HF) fine-scale model, and the other possesses a surrogate material generated using machine learning (ML). For reference, each plot also contains a line with unit slope. | 81 |

SUMMARY

To design structures using state-of-the-art materials like composites and metamaterials, we need predictive tools that are capable of taking into account the phenomena occurring at different length scales. However, the upscaling of nonlinear mesoscale behavior to perform system-level predictions is intractable when using conventional modeling techniques. Other methods like multiscale finite elements are capable of solving arbitrary problems, but they tend to be computationally expensive because they rely on detailed models of the element's internal displacement field. We propose a method that utilizes machine learning to generate a direct relationship between the element's state and its forces, skipping altogether the complex and unnecessary task of finding its internal displacements. To generate our model, we choose an existing finite element formulation, extract data from an instance of that element, and feed that data to the machine learning algorithm. The result is an approximated model of the element that can be used in the same context. Unlike most data-driven techniques applied to individual elements, our method is not tied to any particular machine learning algorithm, and it does not impose any restriction on the solver of choice. In addition, we guarantee that our elements are physically accurate by enforcing frame indifference and conservation of linear and angular momentum. Our results indicate that this can considerably reduce the error of the method and the computational cost of producing and solving the model.

CHAPTER 1

INTRODUCTION AND BACKGROUND

For centuries, scientists and engineers have relied on experimental techniques to characterize the effective mechanical properties of materials. These properties, when paired with appropriate constitutive models, traditionally serve as the basis for any engineering-level mechanics of materials analysis. Such effective behavior is usually dominated by a single fundamental aspect, which is the microstructure of the material. That is, the expressions for the driving forces that dictate the evolution of the state variables depends on the configuration of the material constituents at smaller scales (e.g. the micro scale). The process of generating macroscopic properties and constitutive laws from the microstructure of the material is called homogenization. As a simple example, consider a tensile test on a metal. We know that the effective properties of most metals depends on the microstructure, and that their effective behavior is considerably different than the behavior of the crystals that constitute it. But unless we are interested in micro-scale features, we can stretch the metal and perform other physical experiments to measure its effective properties and directly use them in our mathematical models of the structure. We can think of this process as an experimental form of homogenization.

Modern methodologies, however, tend to replace this phenomenological approach with a combination of homogenization schemes and micromechanical models. The hope is that these multi-scale approaches could minimize empiricism and lead to a more fundamental understanding of the relationship between process, microstructure, and performance in the material development cycle. Most common approaches in this area fall within the following categories: mathematical homogenization, micromechanically-inspired constitutive modeling, and computational homogenization. The first approach usually deals with either linear problems, or with nonlinear problems under very specific material and microstruc-

tural assumptions, e.g. incompressive hyperelastic materials with rigid inclusions [1, 2]. The second approach is usually constrained to very rigid sets of assumptions regarding the microstructure, e.g., the presence of slip planes and directions in crystal plasticity (Taylor model [3]), periodic cracks (Deshpande-Evans damage model for ceramics [4]), dilute concentration of voids (Gurson-type plasticity models [5]), etc. The final approach comprises a large variety of computational homogenization schemes, which includes concurrent multiscale modeling [6]. These schemes have the ability to handle arbitrary microstructures in the nonlinear and history-dependent regimes, but usually at a prohibitive computational cost [7, 8, 9]. A representative volume element (RVE) of the microstructure has to be built for every quadrature point of the finite element model, and solved for every time step of the simulation. Some decoupled variants have been proposed in order to reduce the computational cost (e.g: micro-macro decoupling schemes [10], Nonuniform Transformation Field Analysis[11]) but they are only applicable to a narrow range of problems.

Following a homogenized approach is not technically a necessity, and one could use high performance computing to create massive numerical models of the structure that consider each feature of the microstructure. Unfortunately, such approach does not scale well and the computational resources required for realistic structures is so prohibitive, it lacks any practical use. A more reasonable alternative to perform homogenization at the element level using a specialized finite element formulation. There is a large variety of methods in this category, including the global-local methods [12], residual free bubbles [13, 14], the variational multiscale method [15], the discontinuous enrichment method [16, 17], methods based on partition of unity and the generalized finite element method [18, 19, 20], and multi-scale methods [21, 22]. All these methods share a common trait: they expand or modify the set of shape functions to better fit the particular problem. However, those methods differ in their assumptions about the underlying solution. In the most basic cases, a known handbook solution is used to enrich the basis. This is the typical scenario when using the extended finite element method for crack propagation [23]. In some other cases

the methods are used in conjunction with assumptions over the distribution of features, such as periodicity or volume representation. While additional assumptions tend to considerably restrict the number of applicable problems, the resulting methods tend to have a better computational performance. That is, the knowledge of the particular problem is used as an advantage to optimize the calculations. Element formulations that make few or no assumptions are applicable to more problems but it is difficult to generate the same level of computational efficiency.

To increase the chances of success, it is best to learn from the particular problem under consideration before attempting to solve it. Unfortunately, some problems are beyond human comprehension or their range of applications is too narrow to be worth studying in detail. Cases involving large amounts of variables, high levels of uncertainty, and rapid change in behavior are among the typical scenarios. A possible solution in those cases is to use machine learning to automatically generate a model using data from past experiences. The number of applications of machine learning is extensive and includes self-driving cars, high-frequency trading, house price estimation, and search engines, to name a few [24].

Computational mechanics is not the exception. Machine learning has been used to formulate multiscale elements [25, 26], to enhance the performance of traditional elements [27], to extract constitutive manifolds [28, 29] and to produce a data-driven solver [30]. In fact, some linear methods based on the application of unit displacements can be reinterpreted as models based on linear regression (see example in Section 3.2.1). A problem in all these data-driven methods is that their formulation is intimately related with a particular learning method, and in many cases they involve custom solvers or intrusive techniques. This imposes limits on the learning algorithm, the numerical solver for the system, or both. For example, the formulation of the data-driven solver mentioned above implicitly uses a k-nearest neighbors algorithm with a single neighbor, and changing that algorithm could potentially alter the method.

In this work, we propose to use surrogate modeling to reduce the computational cost

of multi-scale element formulations . That is, we use a machine learning algorithm on data extracted from finite elements or RVEs to build a computationally cheaper approximation to those elements. By separating the element behavior, the learning process, and the solution method, we are able to infer the element model using any learning algorithm, assemble the system of equations using traditional techniques, and solve the generated system using any black-box solver. We also propose several techniques to make effective use of surrogate modeling on element data, including the use of corotational coordinates and the enforcement of physical constraints on the internal forces.

The rest of the thesis is organized in the following way. In Chapter 2 we discuss the challenges of multi-scale modeling and develop a family of finite elements particularly well suited for that kind of problem. In Chapter 3, we develop our approach for using machine learning to generate efficient approximations of existing multi-scale formulations. We consider approaches that deal with multi-scale variants of the finite element method, as well as computational homogenization schemes. In addition, we show how to apply our method to effectively solve problems that involve geometric nonlinearity and history-dependent material nonlinearity. We conclude this thesis in Chapter 4 by summarizing our main findings and defining future directions for our research.

CHAPTER 2

MULTI-SCALE FINITE ELEMENT MODELS

Multi-scale problems are challenging to solve because the difference between the smallest and largest features in a model have a strong impact on the conditioning number of its stiffness. Attempting to represent all scales of the model using traditional finite elements could lead to computationally expensive matrix inversions, and, in the case of dynamic problems, small time increments.

Our work in this chapter is based on the Multi-scale Finite Element Method (MsFEM) [21] and the Geometric Multi-scale Finite Element Method (GMsFEM) [31, 32]. These two approaches deal with multi-scale phenomena using local solutions within the domain of each element to generate the shape functions. Those solutions represent the displacement field inside the element when a unit displacement is applied at a single node and the remaining nodes are fixed. In the case of GMsFEM, each solution is obtained by means of a fine-scale finite element model that uses an auxiliary mesh within the domain of the element. That is, the approach is similar to the traditional Guyan-Irons reduction [33, 34] applied on each auxiliary mesh, with the additional enforcement of inter-element compatibility.

In this chapter, we present an extension to the GMsFEM formulation to better predict the dynamic response of heterogeneous materials and structures. In our approach, we enrich the set of GMsFEM shape functions with a number of vibration normal modes. By doing this, the method allows us to resolve wavelengths shorter than the macro-element size, as well as the effect of internal element features (e.g. inclusions, cracks, etc.) on those frequencies, which has been shown to be a limitation of GMsFEM [35]. Vibration modes are calculated over the domain of each element under fixed boundary conditions, as described in the next section. This scheme for obtaining shape functions resembles

the Craig-Bampton method [36, 34], however here it is applied to individual elements instead of being utilized for reducing the order of structural components. This approach, as demonstrated in subsequent sections, provides a higher degree of control over the FE mesh resolution, helping to avoid the excessively small stable time increments usually incurred by the presence of small geometric features. Moreover, the previously mentioned ill-conditioning problems are entirely avoided.

This chapter is organized as follows. First we introduce the formulation of our modal-based multi-scale finite elements in section 2.1. Then we apply that method to different wave propagation problems in section 2.2. We use those models to measure the performance of our method with respect to traditional finite elements.

2.1 Formulation

Consider a wave propagation problem in a continuous media defined through the differential equation

$$\mathcal{K}[\mathbf{u}(\mathbf{x}, t)] + \mathcal{M}[\ddot{\mathbf{u}}(\mathbf{x}, t)] = \mathbf{q}(\mathbf{x}, t) \quad (2.1)$$

where \mathcal{K} and \mathcal{M} are spatial differential operators specific to each kind of problem, $\mathbf{u}(\mathbf{x}, t)$ is the displacement field, \mathbf{x} is the position in the domain Ω of the problem, $t \in [0, T]$ is the time, $\mathbf{q}(\mathbf{x}, t)$ is a forcing term, and a dot over a variable indicates a derivative over time. We want to find a solution $\mathbf{u}(\mathbf{x}, t)$ of this differential equation that satisfies the boundary condition $\mathcal{B}[\mathbf{u}(\mathbf{x}, t)] = 0$ over the boundary of the domain Γ .

Applying the conventional procedures for finite elements [37, 38], the domain can be discretized into a non-overlapping union of elements that constitute a mesh. The domain of an element e is denoted Ω_e and its boundary is denoted Γ_e . By then employing the weak formulation, an approximate solution $\tilde{\mathbf{u}}(\mathbf{x}, t)$ can be obtained for each domain Ω_e

$$\tilde{\mathbf{u}}(\mathbf{x}, t) = \mathbf{h}(\mathbf{x})\mathbf{u}(t) \quad (2.2)$$

where $\mathbf{h}(\mathbf{x})$ is a matrix of size d by m , with d the dimension of the problem, $m = n \cdot d$, and n the number of nodes, and $\mathbf{u}(t)$ is a vector of length m containing all the degrees of freedom corresponding to the displacement of the nodes. In our approach, we add to this definition another matrix $\bar{\mathbf{h}}(\mathbf{x})$ and the degrees of freedom $\bar{\mathbf{u}}$, which correspond to enrichment terms. The resulting approximate solution is

$$\tilde{\mathbf{u}}(\mathbf{x}, t) = \mathbf{h}(\mathbf{x})\mathbf{u}(t) + \bar{\mathbf{h}}(\mathbf{x})\bar{\mathbf{u}}(t) \quad (2.3)$$

where $\bar{\mathbf{h}}(\mathbf{x})$ is a matrix of size d by p , with p is the number of enrichment functions, and $\bar{\mathbf{u}}(t)$ is a vector containing the additional p degrees of freedom. The approximation (2.3) may be written in the compact form

$$\tilde{\mathbf{u}}(\mathbf{x}, t) = \mathbf{H}(\mathbf{x})\mathbf{u}^e(t) \quad (2.4)$$

where all the degrees of freedom are grouped into a single vector $\mathbf{u}^e(t) = [\bar{\mathbf{u}}^T(t), \mathbf{u}^T(t)]^T$, and the functions into a single matrix $\mathbf{H}(\mathbf{x}) = [\bar{\mathbf{h}}(\mathbf{x}), \mathbf{h}(\mathbf{x})]$.

We restrict the functions $\bar{h}_{ij}(\mathbf{x})$ to be zero valued over Γ_e to preserve continuity across element boundaries independently of the value of $\bar{\mathbf{u}}$, that is, $\bar{h}_{ij}(\mathbf{x}) = 0 \forall x \in \Gamma_e$. Henceforth, we refer to the functions $\bar{h}_{ij}(\mathbf{x})$ as *interior shape functions* and to the functions $h_{ij}(\mathbf{x})$ as *boundary shape functions*.

The boundary shape functions are meant to provide some representation of the field for any specified displacements on Γ_e . They can be traditional FE shape functions, or more elaborated shape functions as in the case of GMsFEM. The interior shape functions can be added to the standard set of boundary shape functions to enrich this basis. Fig. 2.1 shows an example of a particular choice of boundary and interior shape functions for a two-dimensional element.

For the class of stress wave propagation problems considered in this paper, we propose to adopt the GMsFEM shape functions proposed by Casadei et al. [32] as boundary shape

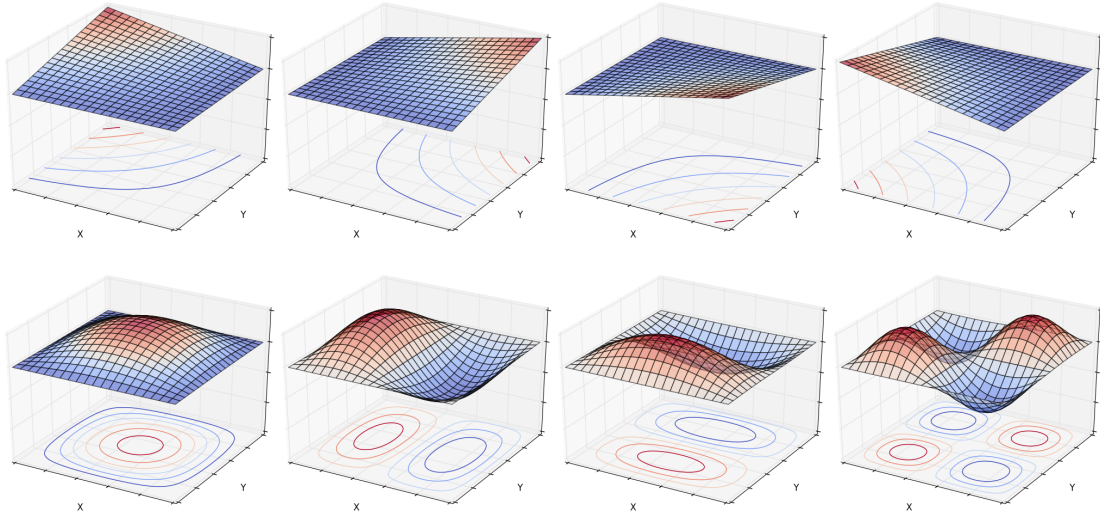


Figure 2.1: Schematics of boundary (top row) and interior (bottom row) shape functions for a square element without internal features. The former represent the deformed configuration for unit displacement of the nodes while the latter represents the natural modes of its fine scale model. The vertical axis reflects the corresponding value of the shape functions over the domain.

functions and analytical and/or numerically computed elemental eigenmodes as interior shape functions. Henceforth, we will refer to our proposed method as *Modal-Based Finite Element Method (MFEM)*. Our choice of splitting the approximating base between interior and boundary shape functions resembles the residual free bubble finite element method, in the sense that the interior shape functions could also be regarded as bubbles. However, there is a critical difference: in the RBF method, the number of bubbles equals the number of basis functions used to span the piecewise polynomial defined in the global element [39], whereas in our approach we can adopt as many natural modes as needed to resolve the required vibration frequencies for the problem at hand.

2.1.1 Boundary shape functions

We propose to use GMsFEM shape functions as boundary shape functions. Consider an element e , the shape functions for this element are built using an auxiliary mesh over the domain Ω_e of the element, which is called *fine scale mesh*. The element e may contain

as many nodes as desired on its boundary in the same way as a traditional element. Each node on the element e should coincide with a node on the corresponding fine scale mesh. The nodes in the fine scale mesh that coincide with the nodes in the element e are called *coarse scale nodes*. The GMsFEM shape functions associated with a particular coarse scale node correspond to the static deformed configuration of the fine scale mesh when a unit displacement is imposed to that node and the other coarse scale nodes remain fixed in space.

Let us consider a problem where the element e contains n_c coarse grain nodes on its boundary and n_f fine scale nodes within the element domain. The equilibrium equations when imposing a unit displacement to one of the coarse scale nodes may be written as

$$[K]\{U\}_k = \{F\}_k \quad (2.5)$$

where $[K]$ is the stiffness matrix, $\{U\}_k$ is the vector of displacements, $\{F\}_k$ is the vector of forces, and the subscript $k \in [1, m_c]$ denotes the particular loading condition, where $m_c = n_c \cdot d$ is the number of DOF for the coarse grain nodes. All the m_c systems can be written in compact form using

$$[K][U] = [F] \quad (2.6)$$

where $[U]$ and $[F]$ are matrices in which the k -th column contains the vectors $\{U\}_k$ and $\{F\}_k$, respectively. Hence, the matrices $[U]$ and $[F]$ have dimensions of m_f by m_c where $m_f = n_f \cdot d$ is the total number of DOFs for the fine-scale mesh. Then, the equations (2.6) may be rearranged, placing in the first m_c rows the equations corresponding to the degrees of freedom of the coarse scale nodes. To make the effect of this rearrangement more visible, the matrices may be partitioned and written in the following form

$$\begin{bmatrix} K_{bb} & K_{ib} \\ K_{bi} & K_{ii} \end{bmatrix} \begin{bmatrix} U_b \\ U_i \end{bmatrix} = \begin{bmatrix} F_b \\ F_i \end{bmatrix} \quad (2.7)$$

where the coarse scale degrees of freedom are denoted with the letter b and the remaining degrees of freedom with the letter i . Since for each load case one coarse scale degree of freedom is equal to 1 and the remaining ones are equal to 0, the upper part of the displacement matrix, $[U_b]$, containing these values, is the identity matrix. The value of the remaining unknown degrees of freedom are contained in $[U_i]$. The forces applied to the coarse scale nodes are also unknowns and contained in $[F_b]$, while the forces applied to the other fine scale nodes, contained in $[F_i]$, are zero. Thus, equation 2.7 can be written as

$$\begin{bmatrix} K_{bb} & K_{ib} \\ K_{bi} & K_{ii} \end{bmatrix} \begin{bmatrix} I \\ U_i \end{bmatrix} = \begin{bmatrix} F_b \\ 0 \end{bmatrix} \quad (2.8)$$

In order to construct the shape functions we are interested only in the resulting deformed configuration of the system, hence we need only solve for $[U_i]$ using

$$U_i = -K_{ii}^{-1} K_{bi} \quad (2.9)$$

where each column in $[U_i]$ represents the values of the fine scale degrees of freedom associated with one of the shape functions.

In general, the number of shape functions associated with each node is equal to the dimension of the model. A more detailed analysis of the procedure, including conditions to ensure conformity of the solution, can be found in the work by Casadei et al. [32]. In the next Section we show how to enrich the basis to include extra information that might have been lost during the condensation of the interior degrees of freedom.

2.1.2 Interior shape functions

Consider an element e with domain Ω_e and boundary Γ_e . We propose to adopt the natural modes of the portion of the structure with domain Ω_e as interior shape functions of the element e . For problems under consideration, the natural modes $\phi(\mathbf{x})$ are defined as the

vectors that satisfy the eigenproblem

$$\mathcal{K}[\phi(\mathbf{x})] = \lambda\mathcal{M}[\phi(\mathbf{x})] \quad (2.10)$$

for $\mathbf{x} \in \Omega_e$ and subject to the boundary condition $\phi_i(\mathbf{x}, t) = 0$ for all $\mathbf{x} \in \Gamma_e$. Because of the particular choice of boundary conditions, the resulting natural modes are local to each element.

Using these natural modes as interior shape functions provides many advantages. First, they can span the complete space of solutions. Therefore, there is always a sufficiently large number of modes that can give a good approximation to the solution of the problem under consideration, granted that the boundary shape functions are properly chosen. In addition, when comparing with other high order schemes, our approach has the advantage that the natural modes are always related to the problem under consideration, and therefore, they may contain information about physical features such as heterogeneities and cracks.

It is worth mentioning that one of the inconveniences of working with natural modes is that, in most cases, analytical solutions are not known and numerical results are computationally expensive. However, in our approach, the geometry assigned to each element is usually very simple, and solutions over their domains are known for many kinds of problems. In particular, we are interested in solutions for fixed boundary conditions, which are usually some of the simplest to obtain.

In cases in which an analytical solution is not known, e.g., for elements with irregular shape or internal features, a numerical solution can be employed. In this case, the numerical solution of the natural modes is calculated only over a very small part of the domain, which corresponds to the domain of each MFEM element (Ω_e). Since these calculations are performed on an element by element basis, the computational cost of this procedure is linear over the total number of elements. This numerical solution can be obtained using the same fine scale mesh previously used for obtaining the boundary shape functions. In

this case, care must be taken regarding the quality of the underlying mesh, which should be appropriate for its use within a traditional finite element framework. For example, it should be appropriate for finding the eigenmodes of the element under consideration. Thus, as a general rule, highly distorted elements must be avoided in the fine scale mesh.

While these concepts are similar to those proposed by Efendiev et al. [40], there are some key differences. In their work, each eigenvalue problem is defined over the neighborhood of the nodes and Newmann boundary conditions are applied. As a consequence, continuity must be enforced across the boundaries of the elements by employing XFEM procedures. However, the shape functions that are associated with any two nodes of an element contain similar information over the domain of that element. Thus, the resulting matrices might be ill conditioned. These undesired effects do not appear in our approach.

2.2 Case studies

In this Section, we apply our method to a set of selected problems. First, we focus on one-dimensional problems for which metrics to evaluate the performance of the method are easy to define. Then, we shift our attention to the simulation of wave propagation on a two-dimensional periodic elastic domain with a notch, and show that our method is particularly well suited to modeling this kind of problem.

2.2.1 Wave propagation in one-dimensional bars

We conduct different numerical tests on one-dimensional bars to highlight the performance of the proposed method. In the first test, we consider the problem of stress wave propagation over a uniform bar, and provide some insights into the accuracy of the method. In the second test, we consider the problem of stress wave propagation over a notched bar, and provide some understanding on the effects of small features and the fine-scale mesh. Finally, we investigate the performance of the method with respect to integration in time.

In all one-dimensional cases considered in this Section, the simulation domain Ω is

$[0, L]$, where L is the length of a bar. For time marching, we use a central difference time integration scheme. The \mathcal{K} and \mathcal{M} operators of the one-dimensional continuum problem are:

$$\mathcal{K} = \frac{\partial}{\partial x} EA(x) \frac{\partial}{\partial x} \quad (2.11)$$

$$\mathcal{M} = \rho A(x) \quad (2.12)$$

where $A(x)$ is the cross-sectional area of the bar, and E and ρ are the elastic modulus and the density of the material respectively. Thus, the elemental stiffness matrix \mathbf{K}^e and mass matrix \mathbf{M}^e are simply

$$\mathbf{K}^e = \int_{\Omega_e} EA(x) \mathbf{B}(x) \mathbf{B}(x)^T dx \quad (2.13)$$

and

$$\mathbf{M}^e = \int_{\Omega_e} \rho A(x) \mathbf{H}(x) \mathbf{H}(x)^T dx \quad (2.14)$$

where

$$\mathbf{B}(x) = \left[\frac{\partial \bar{h}_1(x)}{\partial x}, \dots, \frac{\partial \bar{h}_p(x)}{\partial x}, \frac{\partial h_1(x)}{\partial x}, \frac{\partial h_2(x)}{\partial x} \right]^T, \quad \text{and} \quad (2.15)$$

$$\mathbf{H}(x) = [\bar{h}(x)_1, \dots, \bar{h}(x)_p, h(x)_1, h(x)_2]$$

are defined following the same ordering as for \mathbf{u}^e in (2.4). Also, the maximum index of the boundary shape functions m_c is already chosen to be 2 to avoid the addition of unnecessary (and maybe redundant) functions.

Uniform bar

Let us consider the long bar shown in Fig. 2.2. A longitudinal pulse is applied to one of its

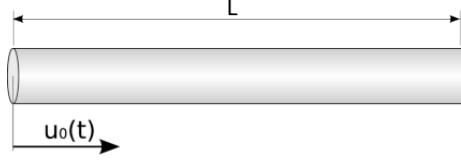


Figure 2.2: Schematic of a uniform bar of length L with a displacement $u_0(t)$ applied at the left boundary.

boundaries by imposing a displacement equal to

$$u_0(t) = \begin{cases} \frac{u_M}{2} \left(1 - \cos(2\pi \frac{t}{t_0}) \right) & : 0 \leq t \leq t_0 \\ 0 & : t_0 < t \end{cases}, \quad (2.16)$$

where t is the time and u_M a constant. The natural modes for fixed boundary conditions are known [41] and equal to $\bar{h}_i(x) = \sin(i\pi x)$, with $i = 1, 2, \dots, p$, for the natural domain of the element $\Omega_e = [0, 1]$. Therefore, a fine-scale mesh is not necessary to obtain the interior shape functions. Further, the static deformed configuration of the bar for unit displacement at one boundary and fixed displacement at the other, is linear. Hence, two linear functions can be used as boundary shape functions. The selected set of shape functions is shown in Fig. 2.3.

The model is composed of N identical elements of length L/N , each of them containing p natural modes and the two boundary shape functions. For the sake of simplicity, the time increments are chosen to be at most $1/20$ of the stable time increment of the equivalent traditional finite element. The effects of the proposed method on the stable time increment are studied later in this Section.

The maximum relative error in the elastic energy of the bar is used to compare the different simulations. Its expression is given by

$$\epsilon_r = \max_{t_0 \leq t < t_f} \frac{|P - P_e(t)|}{P} \quad (2.17)$$

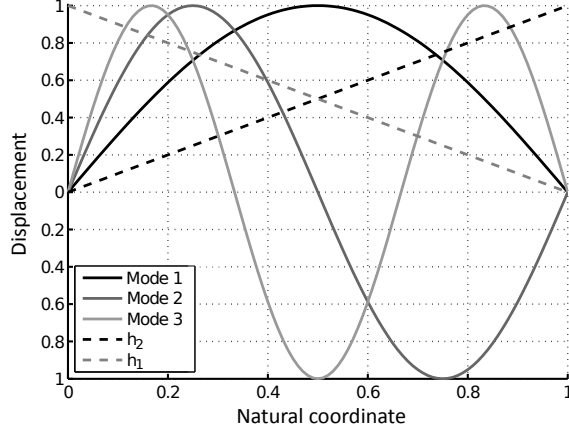


Figure 2.3: Shape functions for a uniform bar element. The boundary shape functions (dashed lines) correspond to the static deformed configuration when a unit displacement is applied at one of the tips. The interior shape functions (solid lines) are the natural modes of the bar.

where P is the analytic value of the elastic energy of the bar after time t_0 , given by

$$P = \int_0^{ct_0} \frac{1}{2} AE \left(\frac{du}{dx} \right)^2 dx = \frac{1}{4} \frac{(\pi u_M)^2 A}{t_0} \sqrt{\rho E}, \quad (2.18)$$

where $c = \sqrt{E/\rho}$ is the speed of sound in the bar. Finally, $P_e(t)$ is the estimation of the energy given by the models through the expression

$$P_e(t) = \frac{1}{2} \mathbf{u}^T(t) \mathbf{K} \mathbf{u}(t), \quad (2.19)$$

In the previous equation, \mathbf{K} is the stiffness matrix of the assembled model, $\mathbf{u}(t)$ the vector containing all the degrees of freedom of the model, t refers to a time instant during the simulation, and t_f is the total simulation time, in this case chosen to be equal to $6t_0$. Note that after the instant in which $t = t_0$, the elastic energy of the bar should remain constant. For this reason, the value of P does not depend on the time t . In this way, the measured error is easy to interpret, and we avoid division by a value of the energy close to zero at the beginning of the analysis.

Results are summarized in Fig. 2.4, where the maximum relative error in the elastic

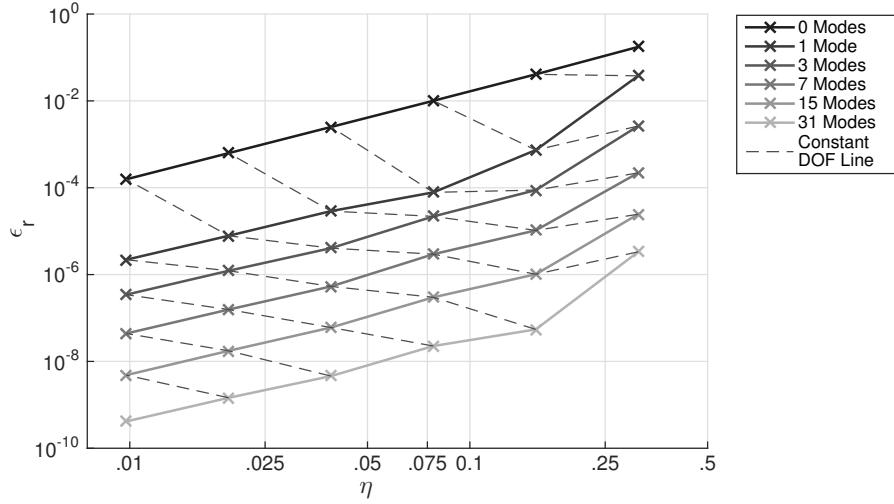


Figure 2.4: Maximum relative error in the elastic energy for the different models of the uniform bar as function of the ratio η between the the length of the element and the length of the pulse. Solid lines are for curves with the same amount of added modes per element, and dashed lines connect models with the same total number of degrees of freedom for comparison purposes.

energy is plotted as a function of the non-dimensional parameter

$$\eta = \frac{L_e}{L_p} \quad (2.20)$$

where L_e is the length of the elements and $L_p = t_0\sqrt{E/\rho}$ is the length of the pulse propagated through the bar. In traditional FEM, we require $\eta \ll 1$ for accurate results. The solid lines in Fig. 2.4 are simulations with the same number of enrichment functions (i.e. number of modes). To provide a sense of computational cost, we added dotted lines to the same plot connecting those models containing the same total number of degrees of freedom (interior plus boundary ones for all elements in the models).

Results of this test indicate that, for most cases, the error is considerably reduced by choosing a higher number of modes *while keeping a constant number of degrees of freedom*. This trend is broken in the vicinity of $\eta = 0.25$, where the error starts to increase with η , as the model loses the ability to represent the corresponding waves. That is, for η larger

than 1 the length of the wave would be shorter than the element itself. Considering that the structure of the resulting elemental matrices is similar for all cases, the lines with constant degrees of freedom can provide a comparison of the computational cost for elements with different values of p during the time marching scheme. Note that the curves for a constant number of modes converge at the same rate. This behavior is because the only difference between points on the same solid line is the number of elements. Consequently, the behavior of the solid curves is similar than that of traditional finite elements (the line in the plot corresponding to 0 modes.)

Bar with a notch

To illustrate the behavior of the proposed scheme for non-homogeneous cases, let us consider a bar of length L where a portion of the bar of length $h = L/10$, has a reduction of 50% in the cross-sectional area, see Fig. 2.5. The imposed displacements are the same as in the previous case, but this time, a model with only one MFEM element containing p modes is compared with a model containing N traditional finite elements.

The analytical solution to the eigenvalue problem is not trivial this time. Thus, numerical solutions are used instead to compute the interior shape functions. A refined standard finite element mesh is created over the domain of the desired element Ω_e . Denoting by \mathbf{K}^f and \mathbf{M}^f the stiffness and mass matrices of the refined model, the interior shape functions correspond then to the solutions ϕ_i of the eigenproblem $\mathbf{K}^f \phi_i - \omega_i^2 \mathbf{M}^f \phi_i = 0$, where the components of the ϕ_i vector at the tips are prescribed to be zero. A limited set of the resulting shape functions is shown in Fig. 2.6 for illustration purposes. The figure also displays the corresponding boundary shape functions, obtained in this case through the GMsFEM formulation.

The maximum relative error ϵ_u in the displacement field of the bar is adopted to compare

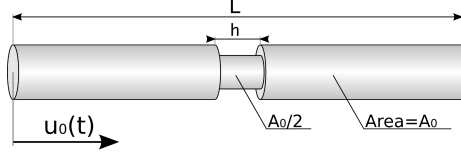


Figure 2.5: Schematic of a notched bar with a displacement $u_0(t)$ applied at the left boundary.

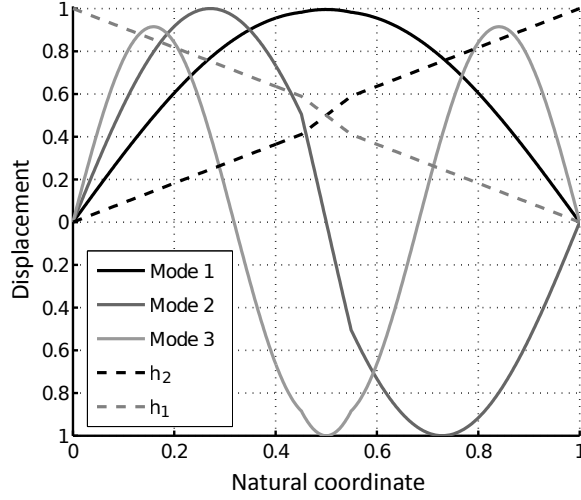


Figure 2.6: Shape functions for a notched bar element. The boundary shape functions (dashed lines) correspond to the static deformed configuration when a unit displacement is applied at one of the tips. The interior shape functions (solid lines) are the natural modes of the traditional finite element model that corresponds to the fine scale mesh.

different simulations. It is defined as

$$\epsilon_u = \max_{t_0 \leq t < t_f} \left(\frac{\|\tilde{\mathbf{u}}(t) - \mathbf{u}(t)\|}{\|\mathbf{u}(t)\|} \right) \quad (2.21)$$

where t_0 , and t_f are as defined in the previous Section, $\tilde{\mathbf{u}}(t)$ is the approximate solution for the case under consideration, and $\mathbf{u}(t)$ is a reference solution computed through a highly refined standard finite element model.

In this reference model, we use the same time integration scheme as in the MFEM approach, as well as time increments that are smaller than $1/20$ of the stable time increment. It is worth noting that, since the fine scale mesh can be arbitrarily chosen, results are

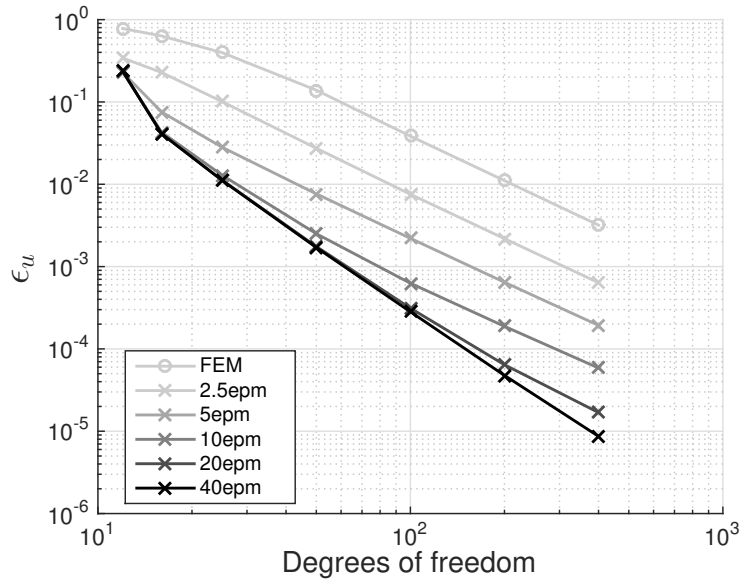


Figure 2.7: Maximum relative error of the displacement field for models with different amount of fine-scale elements per mode (epm), as a function of the total number of degrees of freedom.

compared for different mesh densities.

The maximum relative error ϵ_u in the displacement field is plotted as a function of the number of degrees of freedom in Fig. 2.7. In the plot, the different lines correspond to simulations with the same number of fine-scale elements per number of modes. For example, if 4 modes are used to enrich the element, and we use 10 fine-scale elements to compute them, then we say that we use 2.5 fine-scale elements per mode. This approach provides a good measure of how well the fine-scale mesh can represent the natural modes of a single enriched element. When the amount of fine-scale elements per mode is increased, the estimation of the natural modes used as shape functions gets closer to the real natural modes of the domain under consideration.

We note that, even though in this Section we focus on the case of a single MFEM element for analysis purposes, a more detailed model could contain multiple such elements. In that case, the resultant elemental matrices would have a similar structure to those obtained through the standard finite element method. Furthermore, the corresponding eigenvalue

problems determining the interior shape functions would need to be solved over independent elemental regions. These regions, in general, would be much smaller than the overall size of the problem under consideration. Therefore, the comparison given in Fig. 2.7, using the number of degrees of freedom, can be related to the efficiency of an element when it is included in a large model.

The results show that MFEM provides an increase in precision with respect to standard finite elements for the particular problems considered. From the smooth bar simulations, Fig. 2.4, we showed that for most cases, the error is considerably reduced by choosing a higher number of modes while keeping a constant number of degrees of freedom. From the simulations of a notched bar, Fig. 2.7, we show that for a constant number of degrees of freedom, the results improve when the density of the fine scale mesh is increased. This improvement can be performed up to a converged value of the error, and these converged values show a higher rate of convergence than regular finite elements.

Stable time increment using MFEM

In the previous examples, the integration time steps are small compared to the stable time increment of the time marching scheme, and the error is a converged value over those time increments. In real implementations, however, this is generally not the case. Time increments are usually chosen closer to the stable limit of the integration algorithm in order to reduce the total number of increments in simulations. Different finite element methods generate different stable time increments, which in turn affect the computational cost of a particular approach. This difference in stable time increments must be accounted for when comparing different methods.

In calculations, we adopt a central difference time integration scheme. Consequently, the stable time increment is given by $\Delta t_s = 2/\omega_{max}$, where ω_{max} is the maximum natural frequency of the system. In the case of our proposed elements, there are two competing conditions that induce the stable time increment. One of them is induced by the resolution

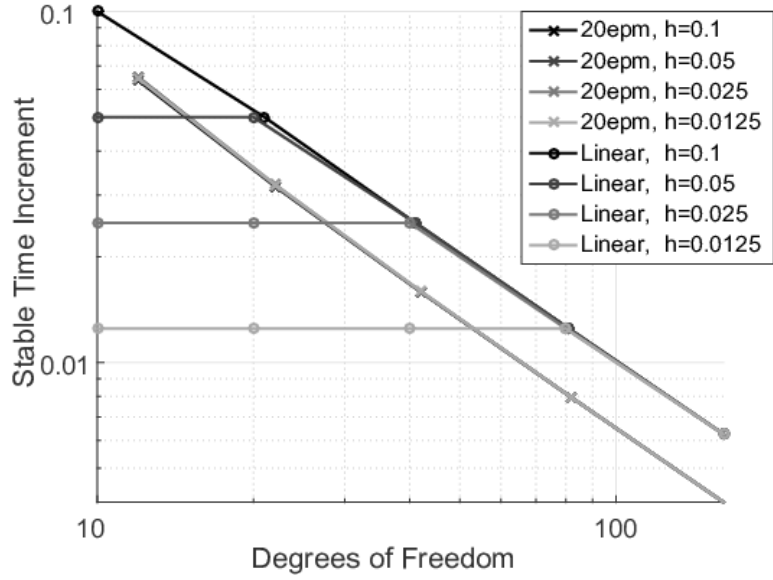


Figure 2.8: Stable time increments for a bar with a notch of length h and 50% reduction of the cross-sectional area.

provided by the coarse scale nodes and can be easily estimated using traditional techniques, i.e., $\Delta t_{s1} = L_e/c$, where L_e is the shortest distance between coarse scale nodes of the element, and c is the speed of sound on the material under consideration. The other one is given by the resolution inside the element, or element order. In our approach, when the natural modes are obtained, we also obtain the associated natural frequencies of the element. The maximum of those natural frequencies ω_{max} induces a stable time increment $\Delta t_{s2} = 2/\omega_{max}$. In calculations, we then adopt the smallest time increment of those competing conditions, i.e., $\Delta t_s = \min(\Delta t_{s1}, \Delta t_{s2})$.

Fig. 2.8 shows the stable time increments for the notched bar simulations, as considered in Sect. 2.2.1, as a function of the system degrees of freedom. The plot shows curves for simulations using traditional linear FE (\bullet symbols) and simulations using MFEM (\times symbols) with the modal shape functions used in 2.2.1. Further, here we consider simulations with a varying notch length h . For MFEM simulations, we chose the fine-scale mesh such that there are 20 fine-scale elements per mode.

When discretizing the notched bar with traditional finite elements, the notch length h

limits the size of the shortest element in the mesh to be at most of length h , that is $L_e \leq h$. Hence, this short element leads to a stable time increment that is equal to or smaller than h/c , where c is the speed of sound in the material of the bar. Thus, for cases in which the notch size h is considerably smaller than the overall length of the structure L , the stable time increment is significantly reduced. This is observed for the linear models (\bullet symbols) in Fig. 2.8, where as the number of degrees of freedom is reduced (by using larger elements), the stable time increment reaches a constant value, set by the size of the notch h , where at least one element must be of size h .

On MFEM elements, the number of DOFs is directly related to the amount of modes included. Also, modes are added to elements starting from their lowest natural frequency and moving up in the frequency domain. Consequently, as shown in Fig. 2.8 (\times symbols), the stable time step for MFEM elements is reduced as we increase the number of DOFs. This indicates that there is a price associated to the highest resolution of the method obtained as more modes are introduced, which is reflected in a reduction of the stable time step. It is worth mentioning, however, that this might not be necessarily a problem, as explained below.

For problems with small features as compared to the overall component size, the finite element mesh far away from those features tends to be much larger than the feature's size. This mesh size, in turn, determines the highest vibration frequency the model can handle. Thus, even if the shortest wavelengths are resolved by a locally refined FEM mesh, they would be filtered out by the rest of the model. From an engineering perspective, when considering this kind of problems, we are usually interested on the effect of those small features in the frequency range resolved by the average mesh size. It is precisely for this kind of situations that our approach is well suited: natural modes can be chosen up to the maximum frequency resolved by the FEM mesh, thus not affecting the stable time step. At the same time, these modes are computed accounting for the presence of small features within the element domain, thus reflecting their effect on the frequency range of interest.

In this range, the stable time step of MFEM elements is insensitive to the size of small features, and only dependent of the maximum enrichment frequency included.

2.2.2 Stress wave propagation in a two-dimensional periodic elastic domain

Stress wave propagation in two- and three-dimensional metamaterials is an active research topic [42, 43, 44, 45]. From the numerical point of view, most studies rely on the traditional Finite Element Method, which has proven to be adequate for problems involving relatively homogeneous materials. That might not be the case for composites and metamaterials, where the distribution of inclusions or voids could lead to small geometric scales that require a large number of small elements in the model. The example presented in this Section shows how our approach can be utilized to efficiently simulate this kind of multi-scale problems.

Let us consider a two-dimensional problem consisting of a plate $0.6m$ long and $3mm$ thick, made of an isotropic material and in plane strain conditions. For two-dimensional problems, the continuum differential operators \mathcal{K} and \mathcal{M} in (2.1) can be written in matrix form as

$$\mathcal{K} = \frac{E}{2(1+\nu)} \begin{bmatrix} \frac{2(1-\nu)}{1-2\nu} \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} & \frac{1}{1-2\nu} \frac{\partial^2}{\partial x \partial y} \\ \frac{1}{1-2\nu} \frac{\partial^2}{\partial x \partial y} & \frac{2(1-\nu)}{1-2\nu} \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial x^2} \end{bmatrix} \quad (2.22)$$

$$\mathcal{M} = \begin{bmatrix} \rho \frac{\partial^2}{\partial t^2} & 0 \\ 0 & \rho \frac{\partial^2}{\partial t^2} \end{bmatrix} \quad (2.23)$$

where E is the elastic modulus, ν is the Poisson's ratio, and ρ is the density of the material. For this case we also have that the elemental stiffness matrix \mathbf{K}^e and the mass matrix \mathbf{M}^e are

$$\mathbf{K}^e = \int_{\Omega_e} h \mathbf{B}(\mathbf{x}) \mathbf{C} \mathbf{B}(\mathbf{x})^T dx dy \quad (2.24)$$

and

$$\mathbf{M}^e = \int_{\Omega_e} \rho h \mathbf{H}(\mathbf{x}) \mathbf{H}(\mathbf{x})^T dx dy \quad (2.25)$$

where $\mathbf{H}(\mathbf{x})$ is a matrix containing the shape functions, $\mathbf{B}(\mathbf{x})$ is a matrix containing the derivatives of $\mathbf{H}(\mathbf{x})$, \mathbf{C} is the stiffness of the material in plane strain conditions and h is the thickness of the plate in the out of plane direction. However, when all of the shape functions are described as deformed configurations of a fine scale mesh, we can use the procedure developed in [32] obtaining \mathbf{K}^e and \mathbf{M}^e from the following equations

$$\mathbf{K}^e = \mathbf{N} \mathbf{K}^f \mathbf{N}^T \quad (2.26)$$

$$\mathbf{M}^e = \mathbf{N} \mathbf{M}^f \mathbf{N}^T \quad (2.27)$$

where \mathbf{K}^f and \mathbf{M}^f are the stiffness and mass matrices of the fine scale mesh, and \mathbf{N} is a matrix that relates the degrees of freedom of the multi-scale element \mathbf{d} and the vector containing the displacements of the nodes in the fine scale mesh \mathbf{d}^f . Since \mathbf{N} is a linear mapping we can write

$$\mathbf{d}^f = \mathbf{N} \mathbf{d} \quad (2.28)$$

That is, in our problem, each column of \mathbf{N} contains the displacements for all the nodes of the fine scale mesh associated with a particular shape function of the coarse-scale element. Some of those shape functions are the GMsFEM shape functions described in Section 2.1.1 and the remaining ones are local natural modes. Those natural modes are obtained in the same way than in Section 2.2.1, where we use $\mathbf{K}^f \phi_i - \omega_i^2 \mathbf{M}^f \phi_i = 0$.

For the particular problem under consideration, a portion of the structure is composed of a periodic elastic domain characterized by a 15mm by 3mm rectangular array of circular

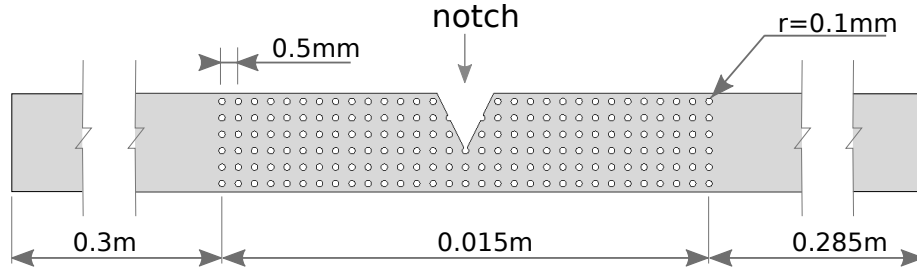


Figure 2.9: Geometry of an elastic plate structure containing a periodic domain and a V-shaped notch. The periodic domain consists of a uniform array of circular holes. The notch cuts through 4 rows of holes and ends in one of them.

voids of radius 0.1mm with a uniform spacing of 0.5mm. This array of voids is embedded within the homogeneous plate material and placed at the center of the plate. At the center of this region, there is a notch starting at the free surface and propagating through plate to a depth of 1.5mm. The geometry of the structure is shown in Fig. 2.9. We confine the location of the periodic domain to a small region at the center of the plate to keep the finite element model (used for comparison) within a reasonable size.

We want to study the combined effects of a periodic domain and a notch on the 2-dimensional propagation of stress waves within the plate. The structure is composed of steel, with an elastic modulus of 200GPa, Poisson ratio of 0.3 and density of 7800kg/m^3 . A uniform pressure is applied on the left side of the plate as a narrow band signal consisting of a five cycle tone burst of 400kHz in a Hanning window. Under these conditions only the s_0 mode is excited, as it is the only symmetric mode that can exist at that frequency [46]. An antisymmetric mode a_0 could also exist at that frequency, but the symmetry of the applied load leads to a pure symmetric excitation.

The problem is studied using a very refined FEM mesh which serves as a reference solution, and comparisons are made with solutions obtained through GMsFEM and MFEM respectively. The mesh upon which all models are based on is shown in Fig. 2.10. In the periodic domain, the mesh is unstructured with at least 8 nodes on the perimeter of each void. The remainder of the structure possesses a structured mesh with square elements

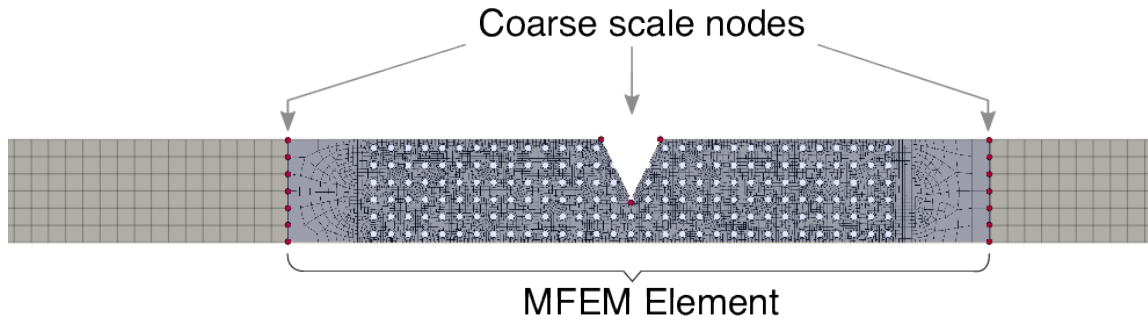


Figure 2.10: Mesh corresponding to the central portion of the structure. On the sides of the Multi-scale element we show a portion of the traditional finite element mesh in the uniform section of the plate. The part of the structure not shown in this figure follows the same uniform mesh pattern.

of 0.5mm size. The only actual difference in the mesh of the three models is that the elements in the shaded area are replaced by a single multi-scale element in the GMsFEM and MFEM cases. The multi-scale element in the GMsFEM and MFEM models use the traditional elements they have replaced as the fine-scale mesh, and the highlighted nodes in Fig. 2.10 as their coarse-scale nodes. For both the GMsFEM and MFEM models, integration over the multi-scale element domain is performed through quadrature points placed over the elements of the fine-scale mesh. Consequently, the associated cost of spatial numerical integration is the same across the 3 models under consideration. The number of modes added to the multi-scale element basis is 64, some of them shown in Fig. 2.11 for illustration purposes. As a consequence, the degrees of freedom in the periodic domain are reduced from 19028 to only 98 in the MFEM element and 34 in the GMsFEM one. More important than a reduction in the number of degrees of freedom, the use of the multi-scale element increases the stable time step by a factor of 25, significantly reducing the temporal integration computational cost associated to this model. Note that our approach reduces this on-line computational cost at the expense of an off-line computation of the eigenmodes of the element. This trade-off is highly dependent on the problem at hand and has to be considered by the analyst.

Fig. 2.12 illustrates the shape of the waves before, during and after the interaction with

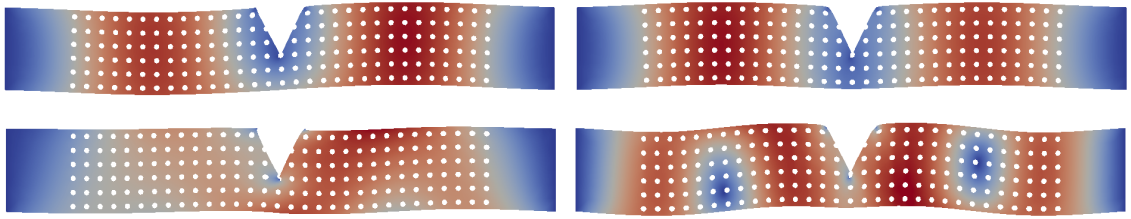


Figure 2.11: Example of selected interior shape functions. Each figure displays a natural mode of the fine scale model within the multi-scale element.

the notch. In addition, Fig. 2.13 shows the horizontal displacements on nodes uniformly distributed over the top surface as a function of time, computed using FEM (solid line) and MFEM (dashed line) on the left figure, and FEM (solid line) and GMsFEM (dashed line) on the right figure. It can be observed that when the wave hits the notch at approximately $t = 60\mu s$, it splits into the symmetric and antisymmetric components of the reflected and transmitted waves. The antisymmetric waves can be identified in this plot because they travel at lower speed than the symmetric components. The speed of each wave can be obtained by measuring the time required to travel the distance between two sensors in the plot. This is also intuitively observed in the plot as the slope of a line connecting contiguous pulses corresponding to the same wave.

Since the multi-scale finite element model uses a basis that is a subspace of the FEM, the best possible scenario is when they produce the same output. The results shown in Fig. 2.13 indicate that MFEM correctly captures the behavior of the FEM with the advantage of larger time increments. As previously mentioned, the time increments of MFEM are approximately 25 times larger than in the FEM for this particular problem. As was anticipated in the previous Section, using MFEM helped to filter the high frequencies of the model to allow the use of larger time increments. Since the larger traditional elements in most parts of the model cannot capture the high frequencies that might be introduced through the input or when the wave hits the notch, any additional filtering of those frequencies in the multi-scale element does not have a significant effect in the model overall.

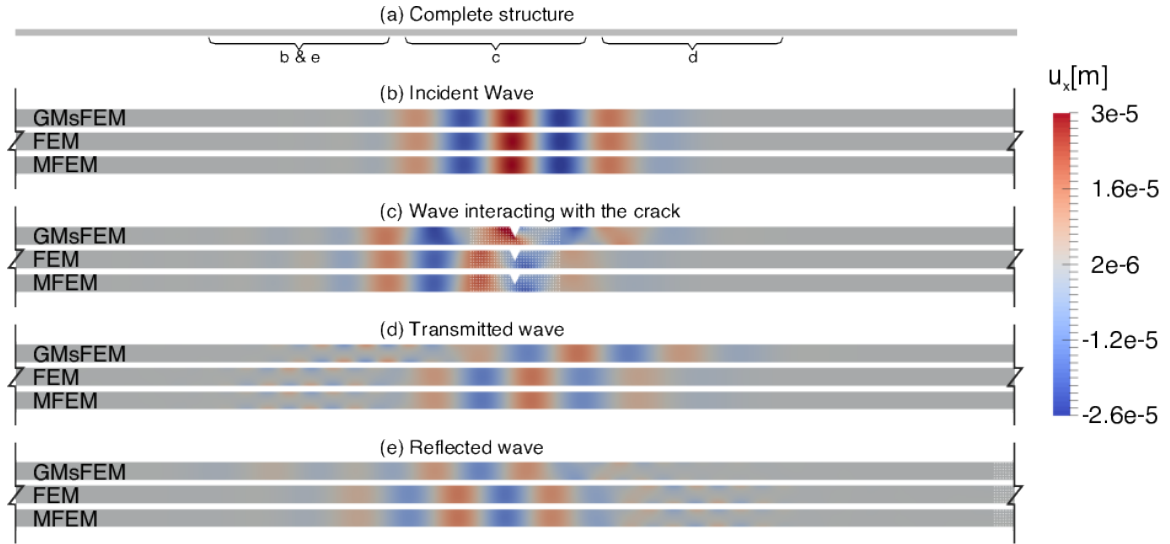


Figure 2.12: Horizontal displacement of the structure produced by the different waves. The results obtained using FEM are compared with those using GMsFEM and MFEM. (a) Complete structure describing the plotted sections in figures b, c, d and e. (b) Incident wave traveling from left to right at $50\mu s$. (c) Displacement field produced during the interaction of the waves with the periodic domain and the notch at $66\mu s$. (d) and (e) Transmitted and Reflected waves respectively at $90\mu s$. Note that there is an antisymmetric component of the wave on the left of figure (d) and on the right of figure (e) Since these antisymmetric components travel at slower speeds than their symmetric counterparts they trail the symmetric waves.

Finally, it can be seen both in Figs. 2.12 and 2.13 that the GMsFEM model fails to capture the response of the system. This is mainly due to the fact that the frequency of interest is characterized by a wavelength that is shorter than the GMsFEM element itself. This example makes clear how the proposed approach improves the enriching technique introduced through GMsFEM for problems involving wavelengths shorter than the multi-scale element. As discussed in Section 2.2.1, a larger time step is attained by incorporating modes up to the frequency of interest. These modes do not attempt to resolve the wavelengths of the fine scale elements used to capture the internal features, in which case there would be no gain in the time step, but to capture the impact of those features on frequencies that are on the range of those resolved by the coarse scale section of the FEM mesh.

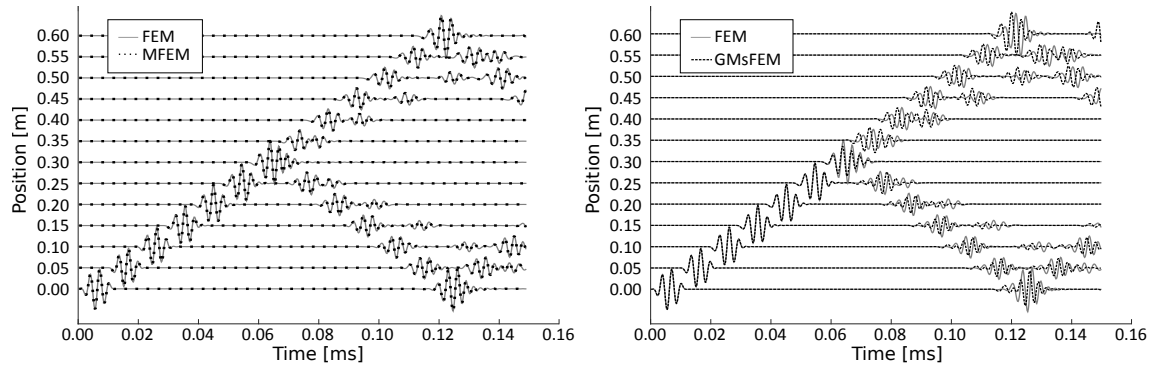


Figure 2.13: Cascade plots of the displacement as function of time for FEM, GMsFEM, and MFEM. Each line corresponds to the axial displacement observed at a node on the top surface of the plate, at the labeled position from the left tip. The figures show (a) a comparison between MFEM and a reference FEM model, and (b) a comparison between GMsFEM and the same FEM reference model.

CHAPTER 3

SMART ELEMENTS

In the previous chapter, we developed a simple technique for multi-scale modeling based on projections of the finite element space in a fine-scale mesh onto a subspace formed with chosen solutions. The main limitation of the method is that it uses solutions to the linearized problem, which may be ineffective in a general and non-linear setting.

In this chapter we follow a different approach and we propose to reduce the computational cost of the fine-scale model by using surrogate modeling. The idea is to extract data from the multi-scale elements and use a machine learning algorithm (see section 3.1) to obtain an approximated and computationally cheaper model of that element. Since individual elements usually have a small dimension, our approach makes effective use of current machine learning algorithms. We also address some inefficiencies of learning directly from finite element data by using corotational coordinates and enforcing physical constraints on the internal forces.

This chapter is organized as follows. We first give an overview of machine learning in section 3.1. Then we define the formulation of our method in section 3.2, and provide a simple example of a linear regression model. In section 3.3, we apply our method to two non-linear problems: a 3D truss structure, and a 2D non-linear multi-scale structure. Here we study the behavior of the method and test its performance. Finally, in section 3.5 we show how to apply our method in a history dependent scenario.

3.1 Overview of machine learning

In our method, we use regression to generate models using finite element data. *Regression* is a category of machine learning that estimates the relationship between some input variables and one or more numeric outputs. Other categories include classification, where the

output variable is a label, and unsupervised learning, where there is no output variable in the data and the objective is to find patterns or regularities in the input.

In broad terms, a *machine learning method* uses an optimization algorithm to find the parameters θ of a function $g(\cdot)$ that minimizes the approximation error $E(\cdot)$ over a set of values \mathcal{D} . In the case of regression, the dataset is in the form $\mathcal{D} = \{x^i, r^i\}_{i=1}^{N_s}$, where N_s is the number of samples, x^i is an input of arbitrary dimension in a system, and $r^i \in \mathbb{R}^m$ is its output. The family of functions $g(x|\theta)$ produces particular functions (or hypothesis) for different values of θ . For example, if $g(x|\theta)$ are the polynomials of a particular order, then θ are its coefficients. The error function defines the distance between an output value in the data set r^i and the output produced by the approximated model $g(x^i|\theta)$. Each particular machine learning method contains its own error function and optimization algorithm. The optimization process over the data \mathcal{D} is known as *model training*, and it produces the optimal parameters θ_m . Those values define a model $g(x|\theta_m)$ that we can use to *predict* the output corresponding to any input value x .

There is a large variety of machine learning methods including ridge regression, lasso regression, k-nearest neighbors, support vector machines, Gaussian processes, neural networks, decision trees, and ensemble methods (which combine different models). Most methods also allow the user to choose a number of values called *hyperparameters*, which give control over some aspects of the method's behavior. The books in references [47, 24] provide a good introduction to the topic, and the article in reference [48] provides an overview of the methods used in optimization from an aerospace engineering perspective. For completeness, we provide below a basic description of the methods used in this work, which are ordinary least squares regression, support vector regression, Gaussian process regression and neural networks. Some of these methods produce only single outputs ($r^i \in \mathbb{R}$) in most implementations. In those cases, we can still produce multiple outputs by simply producing multiple machine learning models, one per output component.

3.1.1 Linear Regression

Linear regression is one of the simplest and most popular machine learning methods, and it is very natural to most engineers because it can be interpreted as an extension to linear interpolation. In the case of regression, we have more points than degrees of freedom, so we fit them in approximated form to obtain the function $g(x|\theta) = \theta^T x$, where $\theta \in \mathbb{R}^n$ is a weight vector, and $x \in \mathbb{R}^n$ is the input vector. In order to simplify the notation we consider the constant term in this function by assuming that the first component of x is always equal to 1. The most common error function is given by

$$E(\mathcal{D}, \theta) = \sum_{i=1}^{N_s} [r^i - g(x^i|\theta)]^2$$

and we can solve for its minimum using

$$\theta_m = (X^T X)^{-1} X^T R$$

where θ_m is the value of θ that minimizes the error, X is a matrix whose i -th row is the vector x^i , and R is a vector whose i -th component is r^i .

Linear regression can also fit nonlinear functions (e.g. higher order polynomials or trigonometric functions) if a mapping is applied to the vector x . In that case, the approximation function is given by $g(x|\theta) = \theta^T f(x)$, where $\theta \in \mathbb{R}^p$ and $f(x)$ is any desired *feature mapping* from \mathbb{R}^n to \mathbb{R}^p . Note that the unit value in the first component of x^i is no longer necessary because one of the components of $f(x)$ can be independent of the input. Predictions made with this method have a computational cost of $\mathcal{O}(p)$.

3.1.2 Support Vector Regression

Another popular and simple method is support vector regression. This method is characterized by its error function $E(\mathcal{D}, \theta) = \sum_{i=1}^{N_s} e_\epsilon(r^i, g(x^i|\theta))$, where $e_\epsilon(\cdot)$ is the ϵ -insensitive

loss function given by

$$e_\epsilon(r^i, g(x^i|\theta)) = \begin{cases} 0, & \text{if } |r^i - g(x^i|\theta)| < \epsilon \\ |r^i - g(x^i|\theta)| - \epsilon, & \text{otherwise} \end{cases}$$

where ϵ is a hyperparameter that determines a region in the input space where the error function is insensitive to the data. Any sample that falls within that area can be dropped and, as a result, the weight vector in the model is sparse. This error function is minimized by solving a convex optimization problem. In addition, since the feature mappings only appear inside inner products, for some particular feature maps we can obtain the inner product directly and without explicitly applying the mapping. Thus, the method allows us to map our features into high-dimensional spaces or even infinite-dimensional spaces. This procedure is called a *kernel trick* in the machine learning literature. We refer the interested reader to references [47, 24] for further details on its implementation. Predictions made with this method have a computational cost of $\mathcal{O}(\nu N_s)$, where ν is the sparsity of the weight vector. Consequently, the method allows us to choose the balance between the error dictated by ϵ and the computational cost dictated by ν .

3.1.3 Gaussian process regression

A stochastic process is an indexed collection of random variables. A Gaussian process is a stochastic process in which every finite subset of the collection has a Gaussian distribution. We can describe a Gaussian process as

$$f(x) \sim GP(m(x), \kappa(x, x')) \tag{3.1}$$

where $m(x)$ is the mean function and $\kappa(x, x')$ is the covariance function or kernel. Given these two functions, the Gaussian process is completely determined.

There are many commonly used kernels, including the squared exponential covari-

ance function, the rational quadratic covariance function, piecewise polynomial covariance functions of compact support, and covariance functions from the Matérn class. In this manuscript we will exclusively use the squared exponential covariance function (RBF kernel), given by

$$\kappa_{SE}(x, x') = \exp\left(-\frac{|x - x'|^2}{2l^2}\right) \quad (3.2)$$

where l is the characteristic length of the process. This parameter gives an idea of how many times the function crosses a particular output level on a given input interval. All the kernels mentioned above also have a similarly defined characteristic length property.

For a finite set of points x , the function value f at those locations will have a Gaussian joint distribution

$$p(f|x) = \mathcal{N}(f|\mu, K) \quad (3.3)$$

where $K_{ij} = \kappa(x_i, x_j)$ and $\mu_i = m(x_i)$.

We can use the last expression and the properties of the gaussian distribution to perform a regression. Given the data $\mathcal{D} = \{x^i, f^i\}_{i=1}^{N_s}$ and some values x^* for which we want to know the function value f^* , we can expand equation 3.3 as

$$\begin{pmatrix} f^* \\ f \end{pmatrix} = \mathcal{N}\left(\begin{pmatrix} \mu^* \\ \mu \end{pmatrix}, \begin{pmatrix} K^{**} & K^* \\ K^{*T} & K \end{pmatrix}\right) \quad (3.4)$$

where $\mu^* = m(x^*)$, $\mu = m(x)$, $K^{**} = K(x^*, x^*)$, $K^* = K(x, x^*)$ and $K = K(x, x)$. Since f , x and x^* are known, this can be rewritten as

$$p(f^*|f, x, x^*) = \mathcal{N}(f^*|\mu^* + K^{*T}K^{-1}(f - \mu), K^{**} - K^{*T}K^{-1}K^*) \quad (3.5)$$

One of the main advantages of this approach is that we not only obtain an estimation for the value of the function, but also its probability distribution. Later in this chapter, this will allow us to draw sample functions or *realizations* from it, something that is not possible

with most of the competing alternatives.

This overview only contained the minimal information we needed later in this article, but we recommend to the interested reader to check references [47, 49]. Those books provide a more detailed analysis of the method and show the effect of the chosen covariance function, how to deal with noisy datasets, and how to optimize parameters in the covariance function to minimize the error, among other topics.

3.1.4 Neural Networks

Neural networks are a powerful machine learning method that allow us to choose the architecture of the model. In this article we use a particular version of the method called recurrent neural networks, which is useful for sequence to sequence problems. To explain its structure we will first explain a more traditional type of model called feedforward neural networks, and then extend it to the recurrent version.

Feedforward neural networks

Traditional feedforward neural networks take the input vector and consecutively apply a sequence of functions to obtain the output vector. These functions are called *layers* and they have the form $g^l(x) = \sigma^l(K^l x + b^l)$ where x^l is input vector of that layer, $\sigma^l(\cdot)$ is a predetermined function called *activation function*, K^l and b^l are a matrix and a vector containing the training parameters, and $l \in \{1, \dots, l_f\}$ is the layer number. The dimension of the layer output y^l can be different from the dimension of the layer input x^l . The output of a particular layer is the input of the next layer, meaning that $y^{l-1} = x^l$. The hyperparameters in this method are the number of layers, the activation function of each layer, the size of the output at each layer, and the parameters associated with the optimization algorithm (explained below), among others.

We show an example of a feedforward neural network in Fig 3.1. In that graph, the connections show what outputs of a node are feed into the inputs of the nodes on the next

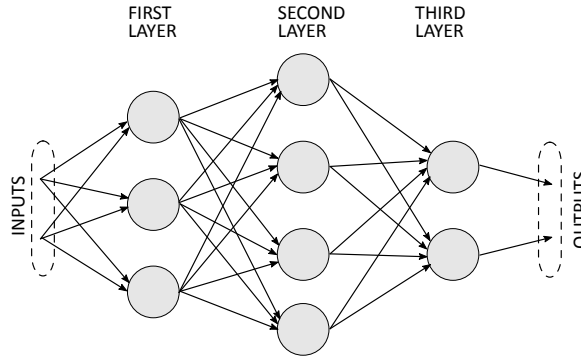


Figure 3.1: Example of a feedforward neural network with three layers. Each node represents an operation $\sigma^l(\sum_k K_{jk}^l x_k^l + b_j^l)$, where K_{jk}^l and b_j^l are the trainable parameters in node j of layer l , and x^l are the inputs for the layer.

layer. The nodes represent operations, which in this case have the form $\sigma^l(\sum_k K_{jk}^l x_k^l + b_j^l)$. This particular example represents a common case where the nodes in a layer are connected to all the nodes in the next layer (the matrix K^l is dense), but this is not always the case.

Once the error metric is chosen, the model is usually trained using gradient descent or one of its varieties. In those methods, a portion of the training samples (a *batch*) is used to estimate the gradient of the error with respect to the parameters. Then the parameters are updated in the direction of the steepest descent. For example, in the most basic scenario we use

$$\Delta\kappa = \eta \nabla E(\kappa)$$

where $\Delta\kappa$ is the change in the weights, η is a hyperparameter called *learning rate* and $E(\kappa)$ is the error given the current weights κ . In this case, κ contains all the parameters in the K^l matrices and the b^l vectors. The most frequently used variations add some momentum to the update rule and use adaptive parameters. To train the model, we perform many of those iterations and go through all the batches in the dataset several times. Each of those passes through the dataset is called *epoch*.

The most-efficient known way to produce the gradient is *backpropagation*, which allows us to compute the required derivatives layer by layer, from the output layer to the first

layer. This algorithm is a special case of *automatic differentiation*[50] and is a cornerstone of modern neural networks. In the most simple case, it is governed by the equations

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) \quad (3.6)$$

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l \quad (3.7)$$

$$\frac{\partial E}{\partial w_{jk}^l} = y_k^{l-1} \delta_j^l \quad (3.8)$$

where $z_j^l = \sum_k K_{jk}^l x_k + b_j^l$ and $\delta_j^l = \partial E / \partial z_j^l$. To use them, we first perform a forward pass through the network to obtain the z_j^l and y_j^l values at all layers. Second, we compute the analytical derivative of the error function with respect to the output parameters $\partial E / \partial y_j^{lf}$, and use those values to compute δ_j^{lf} , which by the chain rule is simply $\partial E / \partial y_j^{lf} \sigma'(z_j^{lf})$. Third, we use Equation 3.6 to obtain δ_j^l for all the layers, starting from the last layer to the first layer. Finally, we use Equations 3.7 and 3.8 to compute the derivatives of the error with respect to the trainable parameters. For more information on backpropagation, we refer the reader to the book in Reference [51], which contains a more detailed and general description of the algorithm.

One of the downsides of feedforward neural networks is that they are not appropriate for sequence to sequence problems. Their output has a fixed length and they do not take into account the input sequence order. For example, if we train a network that predicts the output at a single instant and apply it consecutively to obtain the sequence, the result will likely be poor because the model does not have a memory and it could not use information from the past when predicting. To solve this issue we can use a similar kind of model called recurrent neural networks.

Recurrent neural networks

In recurrent neural networks, the flow of information is not restricted to move from the input layer to the output layer. This is usually represented through connections that feed output values of a layer into the input of earlier layers or into its own input. We can observe a simple example of a recurrent network in Figure 3.2, where the second layer feeds its output to the next layer and onto itself. If we feed the model with a sequence of inputs $\{X^i\}_{i=1}^N$, this recurrence allows the model to store relevant information and use it later on.

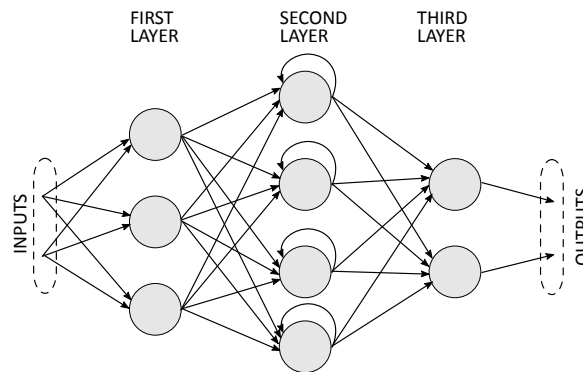


Figure 3.2: Example of a recurrent neural network with three layers. The operations on the nodes are similar to those in a feedforward network except that each the outputs of the recurrent layer are also used as input in the next time increment.

Unfortunately, just feeding the output of a layer into its input vector leads to short memory spans and models with that configuration have very limited applicability. More recently, a number of recurrent layers were developed to handle this kind of problems. In this work we will use a version called *long short-term memory* (LSTM), which has a dedicated memory cell whose information is controlled by a number of gates. Those gates have weights that allow the model to learn how to control the flow of information in and out of the memory.

Training the recurrent neural network is similar to feedforward neural networks, except that each sample consists of a sequence of vectors for the input and another for the output. In addition, to apply the backpropagation algorithm, the libraries need to first expand or

unroll the network, generating a copy of the network for each time increment and connecting the inputs and outputs of consecutive instances. Then, the weights and biases in the network are trained using traditional backpropagation on the unrolled network. This whole process is called *backpropagation through time*. In figure 3.3, we show a simplified recurrent neural network before and after unrolling it. The unrolled network only displays the first three time increments, but subsequent ones would continue with the same pattern. In the figure, the h_i values represent the hidden state in the network and h_0 is the specified initial state. The x_i and y_i values are the inputs and outputs at time increment i . In practice, a symbolic loop can be performed instead of creating the copies so that the training is less memory-intensive.

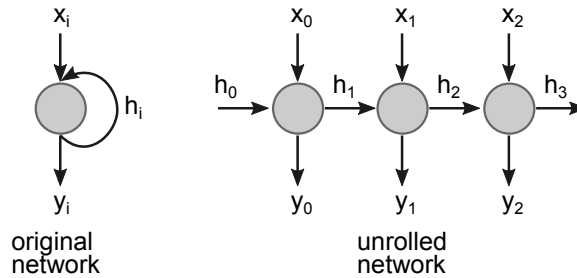


Figure 3.3: Example of a simplified recurrent neural network and an unrolled version of the same network. h_i represents the hidden variables in the model, and x_i and y_i its inputs and outputs, respectively.

3.2 Formulation

Consider a problem in a domain $\Omega \subset \mathbb{R}^d$ that is split into n non-overlapping elements Ω_e such that $\Omega = \cup \Omega_e$. Within each element we approximate a field of interest $u(X, t) \in \mathbb{R}^D$ with a function of the element's degrees of freedom $\phi_e \in \mathbb{R}^{N_e}$, where $X \in \Omega_e$ is the position and $t \in [0, T] \subset \mathbb{R}$ is the time. The degrees of freedom usually consist of the components of $u(X, t)$ at some points $X_e^I \in \Omega_e$ called nodes, but they may contain other values such as rotations or parameters associated with other fields. For example, in a traditional (total Lagrangian) finite element formulation in solid mechanics, the displacement

$u(X, t) \in \mathbb{R}^d$ (here $D = d$) within that element can be approximated using

$$\tilde{u}(X, t) = \sum_{I=1}^{m_e} N_e^I(X) u_e^I(t)$$

where m_e is the number of nodes in the element, $u_e^I(t) \in \mathbb{R}^d$ is the displacement at node I , and $N_e^I(X) \in \mathbb{R}$ is a chosen interpolation function for node I , usually polynomial. In this case, ϕ_e contains the values of all u_e^I .

By using the approximation to the field $u(X, t)$, we discretize the system and reduce the number of variables to a finite number. The elements in this discrete system produce forces $f_e \in \mathbb{R}^{N_e}$ that act on the values ϕ_e and depend on the particular type of problem and formulation. They are forces in a generalized sense and can have components of different types as long as they are complementary to ϕ_e . For example, if a component of ϕ_e is a rotation, its corresponding force component will actually be a moment.

The elements in the model usually have nodes in common with their neighbors (elements sharing a portion of their boundary), and they interact with each other through the nodal parameters in those shared nodes. However, they do not interact in a direct form with non-neighboring elements.

We can assemble the global system of equations, which involves a relation between the element's forces and some derivative of the global degrees of freedom. For example, in mechanical systems, we can write

$$M \frac{d^2 \phi}{dt^2} + \sum_{e=1}^n A_e f_e(\lambda_e) = f(\phi, \dot{\phi}, t)$$

where $\phi \in \mathbb{R}^N$ contains all the degrees of freedom in the model, including the vectors ϕ_e . The input vector λ_e contains all the parameters and values within element e that affect the force f_e and may include ϕ_e , its time derivative $\dot{\phi}_e$, the nodal positions X_e^I , the material properties, internal features of the element, and any other type of information about that element. The matrix $M \in \mathbb{R}^{N \times N}$ is a mass term, which is usually diagonal in dynamic

problems and is zero in static ones. The assembly map A_e (usually represented as a boolean matrix) acts on f_e and produces a vector of size N that corresponds with the global degrees of freedom. Finally, the vector $f(\phi, \dot{\phi}, t) \in \mathbb{R}^N$ represents the external forces.

Although the forces in the element depend on a large number of variables and parameters, it is common to think of λ_e as a vector that only contains terms that change over time or from element to element. In fact, for most traditional elements we think of f_e as a function that only depends on the degrees of freedom ϕ_e . Since the remaining variables are invariant, there is a choice on whether to include them on the definition of λ_e or not. If some values are not included, we can instead write the force function as $f_e^\mu(\lambda_e)$ where μ is a vector that contains the values excluded from λ_e . For example, if the elements in a model share the same material, we can consider the material properties a constant, and exclude them from λ_e . If a number of elements in the model have the same shape, we can exclude the geometric properties associated with those elements from their λ_e vector. The information about the excluded properties would be part of μ and it would implicitly be part of the function $f_e^\mu(\lambda_e)$. To simplify the notation, the μ term will be omitted in the rest of the manuscript.

We say that a number of elements are of the same type if two conditions are met: the parameters included in their input λ_e are the same, and the forces f_e produced by those elements are the same given the same input. Given this definition of element type, the selection of information included in λ_e has an effect on the resulting element types in a model. For example, assume that all the elements in the truss structure from Figure 3 are the same except for their initial length. If we include the initial and the current lengths in λ_e , we can use a single function to obtain the axial force for any element in the structure, and we get only one element type in the model. If λ_e only contains the current length, the information about the initial length can be embedded into the force function f_e . Since the force function used for each element length is different, we would get one element type per element length.

Given a group of elements of the same type, we propose to replace their individual functions $f_e(\lambda_e)$ for a common surrogate model $\bar{f}_e(\lambda_e)$. That is, we obtain a function $\bar{f}_e(\lambda_e)$ by training a machine learning algorithm with a dataset extracted from one or more elements of that type, and use it to predict the internal forces of all the elements in that group.

The elements used to produce that dataset do not need to belong to the particular finite element model, but they do need to share the element type. In this way we can train a model beforehand and apply it on multiple different problems.

A simple way to apply our approach is to train the model with the data given by

$$\mathcal{D}_e = \{\lambda_e^i, f_e^i\}_{i=1}^{N_s}$$

where N_s is the sample size, λ_e^i are particular values of the input λ_e , and $f_e^i = f_e(\lambda_e^i)$ for any element of the chosen type. The particular values of λ_e^i are chosen depending on the kind of problem, the element type, the learning algorithm, and the tolerable error. This set of input values is known as a *sampling plan*, and it is chosen to ensure that the domain where the model is expected to work is sampled with sufficient density. An entire chapter dedicated to this topic can be found in [52], where they discuss several strategies including *stratified random sampling* and *maximin latin hypercubes*.

Unfortunately, training the machine learning model directly on the data \mathcal{D}_e is likely to yield poor results in most cases. A common reason is that the components of λ_e may originate from parameters of a different nature, and their orders of magnitude may depend, among other things, on the choice of system of units. Since most error metrics do not take this into account, the components with a wider range of values have a stronger effect on the resulting error, and as a consequence, they govern the behavior of the trained model. To solve this issue we can simply scale each component in the dataset so that all features have a similar order of magnitude or variance. This process is known as *feature scaling*. We

also show in Section 3.2.2 that we can reduce the dimensionality and size of the sample by using corotational coordinates. This in turn reduces the computational cost of training and prediction, but doing so requires us to apply some mappings on the dataset.

We represent operations on the data \mathcal{D}_e through the mappings $T_{\lambda_e}(\cdot)$ and $T_{f_e}(\cdot)$, which act on λ_e and f_e , respectively. We also define the training data $\hat{\mathcal{D}}_e = \{\hat{\lambda}_e^i, \hat{f}_e^i\}_{i=1}^{N_s}$, where $\hat{\lambda}_e^i = T_{\lambda_e}(\lambda_e^i)$ and $\hat{f}_e^i = T_{f_e}(f_e^i)$. A model \hat{f}_e trained with the set $\hat{\mathcal{D}}_e$ takes $\hat{\lambda}_e = T_{\lambda_e}(\lambda_e)$ as input and produces an output that must be converted to forces f_e using $\bar{f}_e = T_{f_e}^{-1}(\hat{f}_e)$. Thus, we can write

$$\bar{f}_e(\lambda_e) = T_{f_e}^{-1} \circ \hat{f}_e \circ T_{\lambda_e}(\lambda_e)$$

We define a *smart element* as an element that approximates the internal forces f_e using a surrogate model $\bar{f}_e(\lambda_e)$, and a *smart model* as an assembly of finite elements containing one or more smart elements. We also define *base element* as the element or model that generated the dataset \mathcal{D}_e .

Throughout the remainder of this work, we use a total Lagrangian formulation [53] and the displacement field as the field of interest. However, the method is equally applicable to any other formulation and type of problem.

3.2.1 Example: GMsFEM vs Linear Regression

In this section, we illustrate our method through a simple linear regression example. In addition, this analysis will support our earlier claim stating that methods based on the application of unit displacements can be reinterpreted as a linear regression model. To do this, we first describe the geometric multiscale finite element method [54], and then we obtain an identical result using linear regression.

The geometric multiscale finite element method is an intuitive and simple linear model for multiscale structures that makes no assumptions over the distribution of geometric features in the small scale. The method obtains the shape functions using a fine-scale finite

element model within each element. Those shape functions are described using a linear transformation over the coarse scale nodes. That is,

$$u_f = T\phi_e$$

where u_f is the vector containing the displacements in the fine-scale model, ϕ_e contains the displacements at the coarse scale nodes, and T is a transformation matrix.

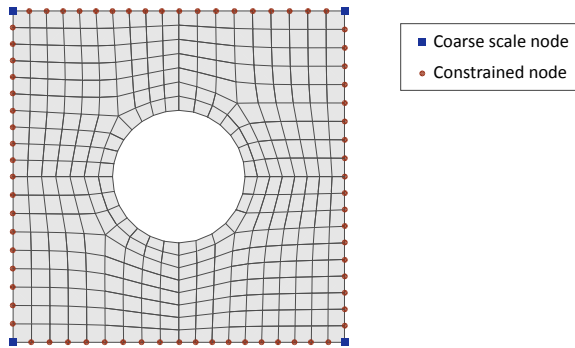


Figure 3.4: Example of a geometric multiscale finite element with the fine-scale model represented as a mesh in its interior.

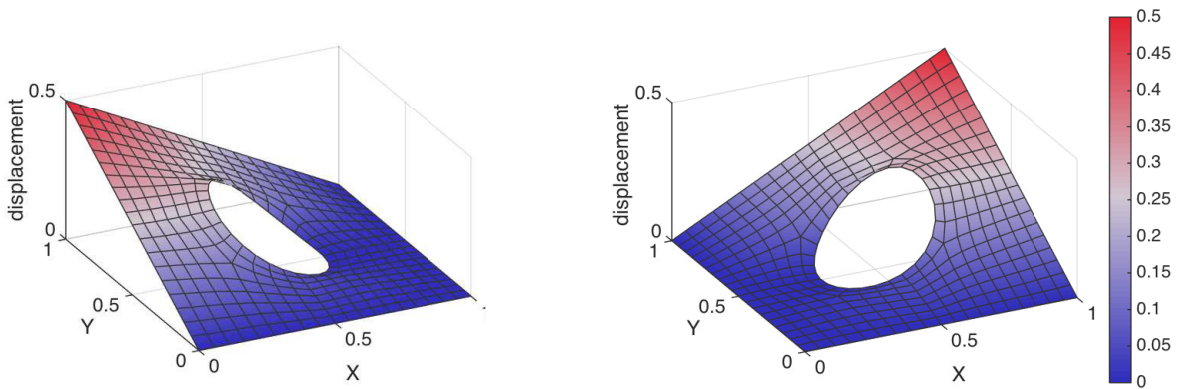


Figure 3.5: Example of shape functions used in the geometric multiscale finite element method.

From the nodes at the boundary of the fine-scale mesh, some are selected as the coarse scale nodes of the element (see Figure 3.4). The remaining nodes at the boundary are

constrained according to some pre-established relationship so they automatically satisfy compatibility with neighboring elements. For example, they can be interpolated with polynomials from the coarse scale nodes. The remaining fine-scale nodal positions or displacements are obtained under the assumption that the dynamic forces in the fine scale are small. Since the boundary is already established we can simply solve the static system to obtain the displacements u_f .

Given that the system is linear, the transformation matrix T is obtained column by column by applying a unit value on a single degree of freedom of the coarse scale. That is, for ϕ_e equal to a vector with value 1 in the i -th row and 0 everywhere else, we obtain a fine-scale displacement vector u_f that is the solution of the fine-scale model and use it as the i -th column of T . Two of such solutions are shown in Figure 3.5. Once this procedure is performed for all degrees of freedom the stiffness matrix of the element K_e can be obtained using

$$K_e = T^T K_f T \quad (3.9)$$

where K_f is the stiffness matrix of the fine-scale model.

Obtaining the matrix T is useful since it describes the displacements of the fine-scale mesh, but in many cases, only the value at the coarse scale nodes is necessary and storing a large matrix per element can generate storage problems. In this case, a smart element can directly obtain the element stiffness matrix. For that purpose, we generate a linearly independent sample $\mathcal{D}_e = \{\phi_e^i, f_e^i\}_{i=1}^{N_s}$ where N_s is equal to the number of degrees of freedom of the element N_e . Each case in the sample is the solution of a static problem in the fine-scale model with forces applied only on the coarse scale nodes. Those applied forces f_e^i can be arbitrary as long as they form a linearly independent set. The solution to the fine-scale model yields the displacement at all nodes, including the displacement at the coarse scale nodes ϕ_e^i . Since we have N_e samples for N_e degrees of freedom, linear regression produces a matrix K_e that perfectly predicts the sample. Since the problem

is linear, it also predicts correctly the forces for any other displacement and the stiffness matrix matches exactly the one obtained through Expression 3.9.

3.2.2 Introducing physical considerations

In this section, we show two ways to increase the performance of smart elements in mechanical systems. In the first case, we use corotational displacements to reduce the input dimension and the sample size, and in the second we enforce constraints in the output of the model to reduce the error and the dimension of the output. We make substantial use of these techniques in the case studies from Section 3.3.

Corotational displacements

In mechanical systems, the rigid body motion of a deformed structure does not affect its stresses, only their orientation. Many nonlinear finite elements reflect this behavior, and their internal forces are independent of the rigid body motion when measured in a reference frame that rotates with the element (its corotational reference frame). Thus, we can calculate the same internal forces using only the information from the deformation component of the displacement. Later in this section, we discuss the potential benefits of this idea.

We can decompose the displacement variables u_e^I of the element into a rigid body component u_{er}^I and a deformation component called corotational displacement u_{ed}^I . That is

$$u_e^I = u_{er}^I + u_{ed}^I$$

The rigid body motion of an element is determined from its translation u_{et} and its rotation, which can be described as a rotation matrix R_e . The rotation matrix R at a particular point in the element can be obtained using the polar decomposition of the deformation gradient F [55]. However, the deformation gradient in most elements changes from point

to point, which means that we can define the rotation of the element in many different ways. For example, we can use the rotation at the center of the element or an average value taken from a number of points [56], or we can derive it from geometric considerations [57]. The translation u_{et} can also be defined in different ways. We can use the displacement at the center of the element, the average displacement of the nodes, or a weighted average displacement over some other points in the element (for example, the integration points).

Once we know the rotation matrix R_e and the translation u_{et} of the element, we can decompose the displacements. The expression of this decomposition depends on the order in which we perform the deformation, the rotation, and the translation. Assuming that we use that order we can obtain the corotational displacement using (adapted from [57])

$$u_{ed}^I = R_e(u_e^I - u_{et} + X_e^I - X_{ec}) - (X_e^I - X_{ec}) \quad (3.10)$$

where X_{ec} is the center of the element, and we also assume it to be the center of the rotation. We can define X_{ec} as the average of the nodal positions X_e^I or the geometric center of the element.

If necessary, we can use this last result to obtain the rigid body component from $u_{er}^I = u_e^I - u_{ed}^I$. But as we mentioned earlier, the internal forces produced by the element in the corotational reference frame are usually independent of these values, and we can ignore them.

One advantage of using corotational displacements in the input is that, by construction, the resulting smart elements exactly satisfy frame indifference. Otherwise, frame indifference could only be achieved by approximation because the components of the output generated by a single output machine learning library are not directly related, and the components of the error, which are random, are not necessarily frame indifferent. Using corotational displacements also considerably reduces the computational cost of training and prediction for the machine learning model because it helps us reduce the dimension of

the input and the number of training samples.

We can reduce the dimension of the input because the corotational displacements u_{er}^I are linearly dependent. To prove this, consider a vector ϕ_{ed} of size $N_e = dm_e$ that contains all the components of the corotational displacements u_{ed}^I . To completely determine the rigid body motion of the element we need N_{er} independent parameters (which is 3 for 2D problems and 6 for 3D problems). Given $N_{ed} = N_e - N_{er}$ values of ϕ_{ed} , the remaining ones can be obtained by assuming that the rigid body component of ϕ_{ed} is zero. Thus, we can discard any N_{er} components from ϕ_{ed} without losing information.

In addition to reducing the dimension of the input, using the corotational displacements drastically reduces the number of samples required for training. Since the rigid body motion of the element is unbounded, a sample of u_e^I needs to cover a large range of possible displacements. At the same time, a comparatively small change in its values can lead to large deformations, meaning that we need a high density of training samples. Satisfying both requirements is computationally prohibitive because it would lead to a large sample size. On the other hand, using the corotational displacements as input is computationally sound because the material behavior restricts the range of deformation. That range is much smaller than the range for the rigid body motion, and the resulting number of samples is reduced proportionally.

If the internal forces do depend on the rigid body motion, it is still best to decompose the displacement into a rigid body motion part and a deformation part so that we can pass both sets of values as inputs in $\hat{\lambda}_e$. Then we can use feature scaling so that both components have similar ranges of values, and thus the same opportunity to affect the internal forces in the machine learning model.

A consequence of using the corotational displacements is that the resulting forces will be in the corotational reference frame. To obtain the forces in the fixed reference frame we only need to apply the inverse rotation R_e^T to the forces predicted by the machine learning model \hat{f}_e . This step corresponds to the mapping T_{fe}^{-1} .

Equilibrium of internal forces

Although the internal forces in an element f_e must satisfy equilibrium, the errors from the surrogate models may induce small spurious total forces and moments. The sum of those small values over a large number of similar elements could lead to a total spurious force or moment of significant magnitude and considerably affect the results. Thus, it is usually best to enforce internal equilibrium of the smart elements.

In mechanical systems the equilibrium is dictated by a zero sum of internal forces and moments. We write this kind of relationship in a generic form using

$$R_e^i(f_e) = 0 \quad (3.11)$$

where R_e^i is some known function of the internal forces, $i = 1, \dots, q_e$, and q_e is the number of relationships. We say that an element is *balanced* if it satisfies these relationships.

A simple way to eliminate the out-of-balance forces and moments is to discard q_e values from the force \hat{f}_e and obtain them solving the q_e equations from Expression 3.11. The positive consequences are that we no longer need to train models corresponding to the discarded components, and we need to predict fewer force components. We used this procedure in the case study from Section 3.3.1.

Alternatively, we can train and predict all components of the output f_e , and use the extra information to reduce the error in the model. Assuming that the forces \bar{f}_e contain an error $\bar{e}_e \in \mathbb{R}_e^N$, we can write

$$R_e^i(\bar{f}_e + \bar{e}_e) = 0$$

for $i = 1, \dots, q_e$. We cannot solve this system of equations because the number of unknowns N_e is larger than the number of equations q_e . However, we can make some assumptions over \bar{e}_e and obtain an approximate solution. For example, we can assume that \bar{e}_e is a linear combination of q_e chosen forces \bar{e}_{ei} , which may represent uniformly distributed

forces or pairs of forces producing a moment. That is,

$$\bar{e}_e = \sum_{i=1}^{q_e} \alpha_i \bar{e}_{ei}$$

where α_i are the unknown coefficients. We used this procedure in the case study from Section 3.3.2.

Other alternatives we can use to balance the forces are finding the force \bar{e}_e with the smallest magnitude that satisfies the constraints, or projecting the force \bar{f}_e onto the subspace of forces that satisfy the constraints. If \hat{f}_e are the corotational forces, we can discard q_e random components and replace them with the solutions to Expression 3.11, repeat the procedure a number of times, and take the average of all the results. However, all the variants presented in this section achieve the same goal: to guarantee that the element satisfies conservation of linear and angular momentum.

3.3 Case studies

In this section, we provide two example applications for smart elements. In the first case we solve a 3D truss structure using smart elements and compare the results with another data-driven method from the literature [30]. In the second one, we solve a 2D continuous beam with voids in the material and compare the behavior induced by different machine learning algorithms. Both cases extensively use the techniques and considerations explained in the previous section.

3.3.1 3D truss structure

In this case study, we follow a static 3D truss problem suggested by Kirchdoerfer et al. [30]. The structure is composed of cubic truss cells of size 1 m and its configuration is given in Figure 3.6 together with its loads and imposed displacements. The bars in the structure have a cross-sectional area of 0.001 m^2 and a hyperelastic material with the stress-strain

relationship from Figure 3.7.

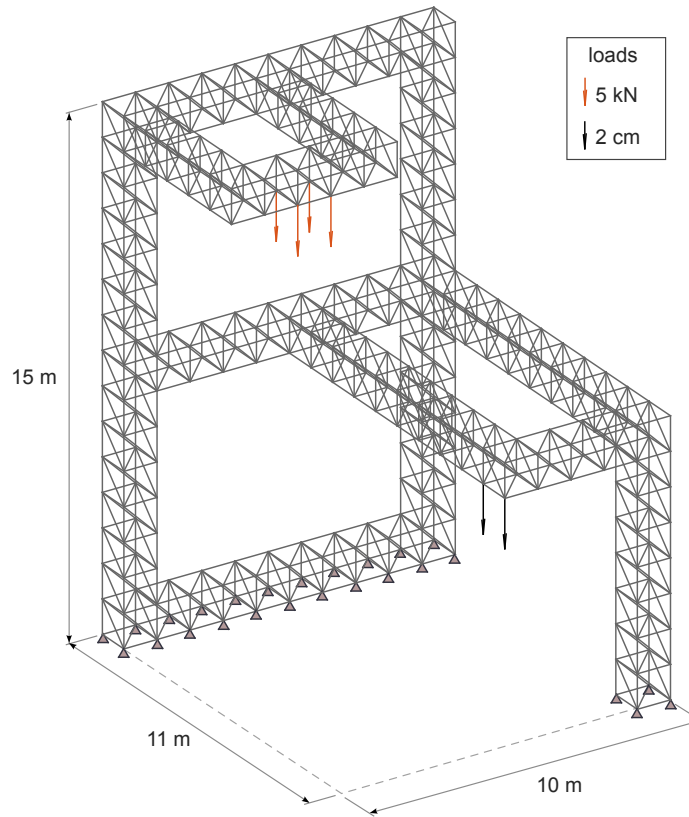


Figure 3.6: Example truss structure with its applied loads and imposed displacements.

To solve the system we use smart elements in a corotational formulation. That is, the stretching of the bar and the axial force are measured in a reference frame that rotates with the bar. We then obtain the nodal forces by extracting the components of the axial force into the global reference frame. We compare the results with a traditional finite element model that is also corotational and with the same definition than the base element. In both cases, we solve the system using a Newton-Raphson solver. We obtain the tangent stiffness matrix for the smart elements using the approximated numerical derivative of the nodal forces [58].

The corotational formulation of the truss elements implicitly follows the ideas from Section 3.2.2 and uses only the information of the displacement variables that affects the deformation. We describe the deformation of the element using its extension ΔL_e , which

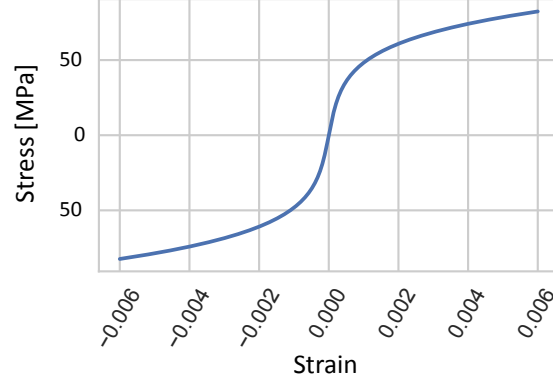


Figure 3.7: Stress-strain relationship of the material in the truss structure. Adapted from Kirchdoefer et. al. [30].

is given by

$$\Delta L_e = L_e - L_{0e} = \|x_e^2 - x_e^1\| - \|X_e^2 - X_e^1\|$$

where L_e and L_{0e} are the current and initial length of bar e , and $x_e^I \in \mathbb{R}^3$ and $X_e^I \in \mathbb{R}^3$ are the current and initial position of the nodes, respectively.

Assuming that the displacements of the bar in the corotational frame $\hat{u}_e^I \in \mathbb{R}$ are measured from the center of the element, we can also write

$$\Delta L_e = \hat{u}_e^2 - \hat{u}_e^1 = 2\hat{u}_e^2 = -2\hat{u}_e^1$$

Thus, we can associate the state in the 3D model with the displacements in the local frame.

For our smart element we use $\hat{\lambda}_e = [1000\Delta L_e, L_{0e}]^T$, where we apply a scaling factor of 1000 to ΔL_e , so that both inputs have similar orders of magnitude. This selection of $\hat{\lambda}_e$ is not the most efficient one, and $\Delta L/L_0$ as a single input would yield a better result in this particular problem. However, our choice is more generic (for example, if the bars could buckle we would need the length) and our intent is to show that the elements can learn the appropriate behavior from the data.

On a similar note, the force obtained for each base element is also in its corotational frame (we obtain only the axial force). Thus, that value already corresponds to the mapped

value \hat{f}_e , and we only need to describe the inverse transformation $T_{f_e}^{-1}$. In this case, that inverse decomposes the axial force predicted by the machine learning model into the nodal forces in the global reference frame. For each node we have

$$f_e^I = \frac{x_e^2 - x_e^1}{\|x_e^2 - x_e^1\|} f_a^I$$

where $f_e^I \in \mathbb{R}^3$ is the nodal force in the global reference frame applied to node I , $f_a^I \in \mathbb{R}$ is the axial forces applied to that node, and $I = 1, 2$.

However, as we explain in Section 3.2.2, if the machine learning model predicts the force f_a^I on both nodes, they may not be equal in magnitude. Thus it is best to predict only one of them (we chose f_a^2), and use the internal force equilibrium relationship in the local reference frame (in this case $f_a^1 = -f_a^2$) to obtain the other. Then we can write

$$T_{f_e}^{-1}(\hat{f}_e) = \left[-\frac{(x_e^2 - x_e^1)^T}{\|x_e^2 - x_e^1\|}, \frac{(x_e^2 - x_e^1)^T}{\|x_e^2 - x_e^1\|} \right]^T \hat{f}_e$$

where $\hat{f}_e = f_a^2$.

In this particular case, we obtain the forces in the smart element using support vector regression (SVR) with a kernel based on radial basis functions (RBF). The main hyperparameters are then C and γ , which define a penalty parameter for the error function and the coefficient in the radial basis function, respectively. The SVR models are trained using a sample generated from independent and identically distributed values of L_0 and ΔL . Each value is extracted from a uniform distribution with $L_0 \sim U(0.5, 2)$ and $\Delta L \sim U(-0.006, 0.006)$.

The exact values of the hyperparameters γ and C are usually not important and there is a range of values that yields models with similar performances. To find a reasonable pair we perform a grid search using the R^2 score of each model against an independent test sample of size $N = 10000$. In Figure 3.8, we show heatmaps with those scores for different values of ϵ and N , together with a number that indicates the sparsity of the system. Smaller sparsity ratios lower the computational cost for force predictions.

This problem is deterministic, so there is no risk of overfitting the model, and increasing the value of C improves the resulting score in all cases. Since increasing C also increases the computational cost of training, we limit its value to a maximum of $1e5$ in the subsequent analysis. The practical implication of this limit is that the optimal value of C is equal to $1e5$ in all of our deterministic scenarios. The results from the grid search also show that the optimal γ is independent of ϵ , but it increases with N . However, for cases with similar scores, it might be best to choose γ based on sparsity to achieve better computational performance.

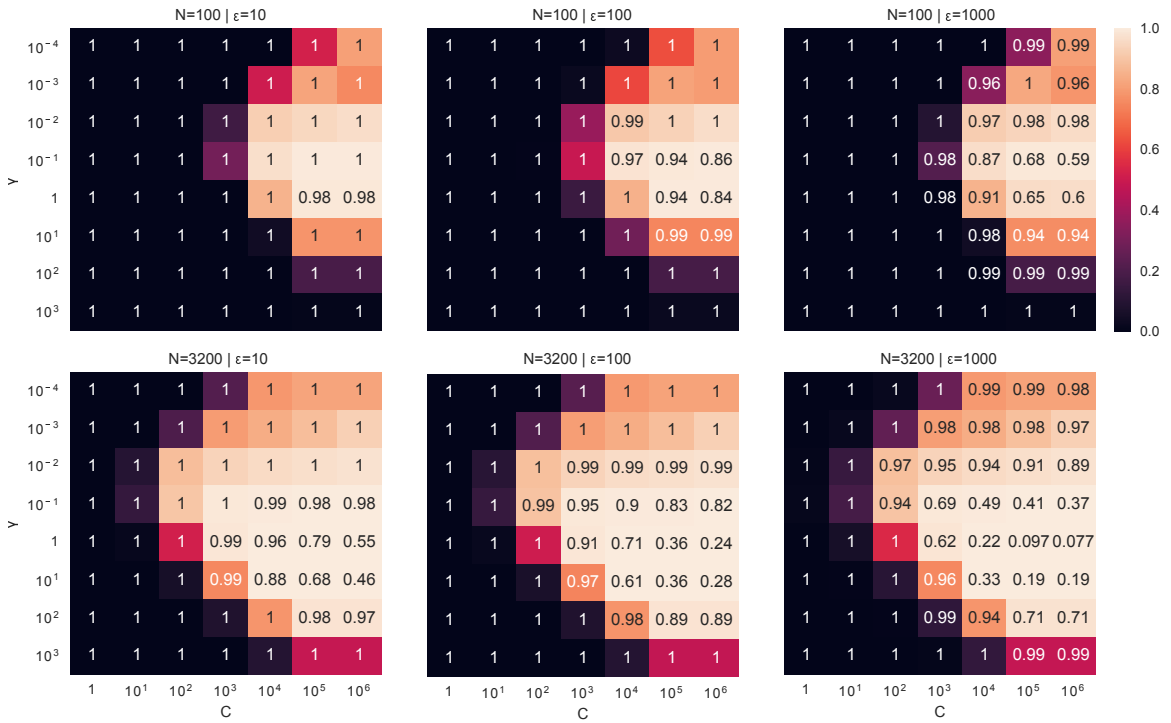


Figure 3.8: Heatmaps for hyperparameter fine tuning. The colormap indicates the R^2 score for a test dataset of size 10000 and the value inside each box indicates the sparsity of the solution. The R^2 values have been limited to a minimum of 0 in the color visualization.

In this particular problem, the loads produce a maximum displacement of approximately 6 cm and a maximum stress equal to 42.2 MPa . While that distance is not large enough to produce large rotations in the structure, the use of corotational elements is still recommended in our formulation because of the reasons described in section 3.2.2. We

show the error in the truss structure in Figure 3.9 for three different values of ϵ and as a function of the number of points in the sample N . Each plot contains different curves for γ while C is fixed and equal to $1e5$ in all cases. The metric is the root mean square (RMS) error in the stress given by

$$\epsilon_{RMS}^{\sigma} = \frac{1}{\max(|\sigma_i^r|)} \left(\sum_{i=1}^m \frac{(\sigma_i^r - \sigma_i)^2}{m} \right)^{\frac{1}{2}}$$

where m is the number of elements and the r superscript indicates that it belongs to the reference traditional finite element model. We also note that in this case the stress produced by the smart elements is obtained by dividing their axial force by the area of the bar.

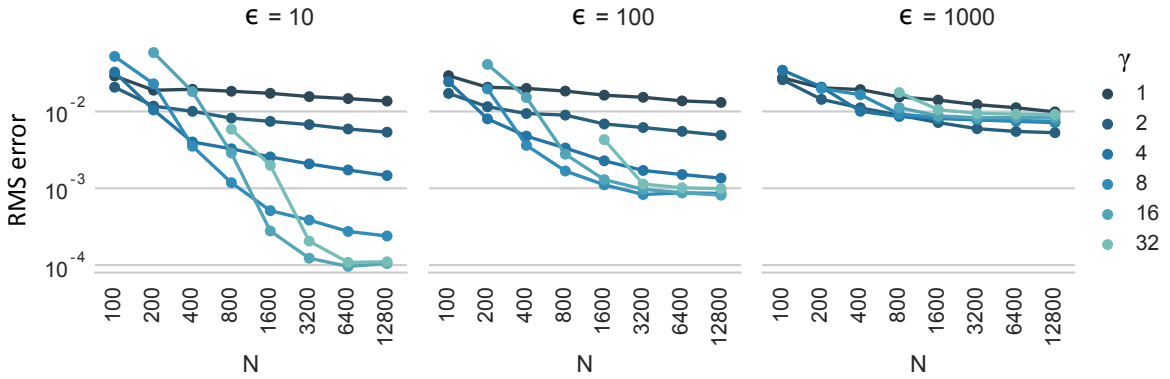


Figure 3.9: RMS Error of the stress in the structure for different values of ϵ and γ depending on the size of the training sample N . Each point represents the average over the converged models obtained from 10 different training samples.

From these figures, we can draw some conclusions even though many are evident from the typical behavior of SVR. As expected, the error is reduced with the number of samples. The convergence rate with fixed γ tends to be small, so increasing the number of samples requires increasing γ for better convergence. For the cases with a small number of samples and large γ , there are areas in the domain of the input that are far from all points in the sample according to the error function. As a consequence, those models tend to underperform.

We also found that the value of ϵ imposes a limit on the minimum error that can be

achieved because the loss function is not sensitive to any error smaller than ϵ during training. For this particular problem small values of ϵ lead to smaller errors, but they also produce dense models with computationally expensive force predictions.

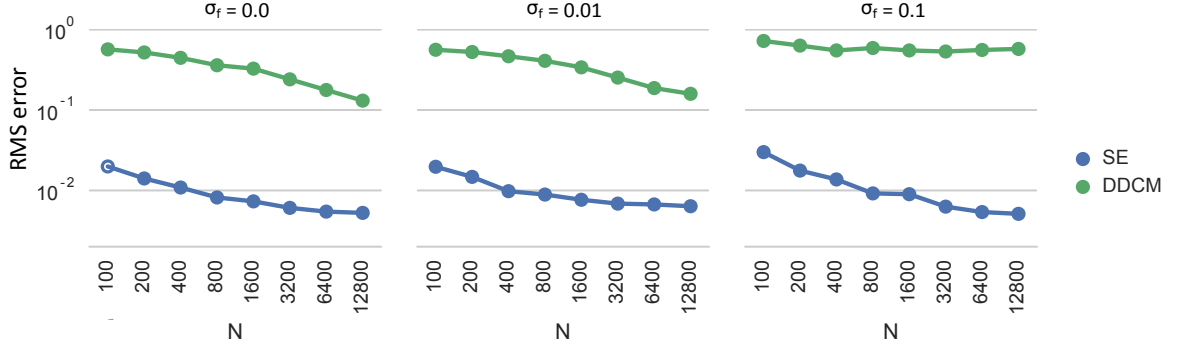


Figure 3.10: Comparison of the RMS Error of the stress in the structure ϵ_{RMS}^σ for smart elements (SE) and data-driven computational mechanics (DDCM) as function of the training sample size N . The output in the training samples have a normally distributed error with a standard deviation σ_f . Each point in the plot represents the average error from 10 models with different training samples.

In Figure 3.10 we show the error in the structure when the forces in the training sample have a normally distributed error with a standard deviation σ_f . The smart elements used in that case have hyperparameters $C = 1e5$, $\gamma = 2$ and $\epsilon = 1000$. We also show in the same figure the error obtained using the method proposed by Kirchdoerfer et al.[30], which is named data-driven computational mechanics. Their method has a hyperparameter C_e that defines their error function. Since it is not possible to use traditional testing to obtain a reasonable value for that hyperparameter, we tested a number of cases and present here the one that yields the smallest error in the structure, which is $C_e = 1e10$.

We found that our method produces a smaller error than its data-driven counterpart by at least an order of magnitude, while remaining less susceptible to errors in the training sample. Comparing with Figure 3.9 we can also see that choosing different values of ϵ and γ could reduce the error in the smart elements even further, but doing so would increase the computational cost. For example, Figure 3.8 indicates that for $N = 3200$ and $\gamma = 1$, the prediction cost is reduced by a factor of 8 by choosing $\epsilon = 1000$ instead of $\epsilon = 10$.

We believe that the method by Kirchdoerfer et al. could achieve similar levels of error than ours when combined with a constitutive manifold construction like the one proposed by Ibañez et al. [28]. However, the main advantage of our method remains in its simplicity. We use traditional finite element solvers and techniques in combination with simple, publicly available and popular machine learning libraries [59] to achieve solutions that are acceptable for most engineering applications.

3.3.2 Nonlinear Multiscale problem

In this case study, we focus on the multiscale finite element method for nonlinear problems formulated by Efendiev et al. [22]. This method is similar to the geometric multiscale finite element method introduced in Section 3.2.1 in the sense that it uses a fine-scale model to determine the deformation within the element. It is possible but not required to use the finite element method to obtain the fine-scale solution. One difference, however, is that the test functions in this case are traditional polynomial shape functions. The method works for nonlinear models, but it requires the solution of the nonlinear static fine-scale problem within each element for each iteration step. In the case of a large number of elements in a dynamic simulation with a large number of time steps, the computational cost becomes prohibitive. We can use smart elements to produce an approximated model with a reduced computational cost.

We study a simple 2D structure in plane strain shown in Figure 3.11. It is a rectangular beam of size 10 cm by 5 cm with a uniform array of circular holes of diameter 0.4 cm separated by a distance equal to 1 cm . The material of the structure is Neo-Hookean, which has a strain energy density U given by [55]

$$U = \frac{\mu}{2}(\bar{I}_1 - 3) + \frac{\kappa}{2}(J - 1)^2$$

where $\bar{I}_1 = I_1/J^{2/3}$, I_1 is the first invariant of the right Cauchy-Green deformation tensor

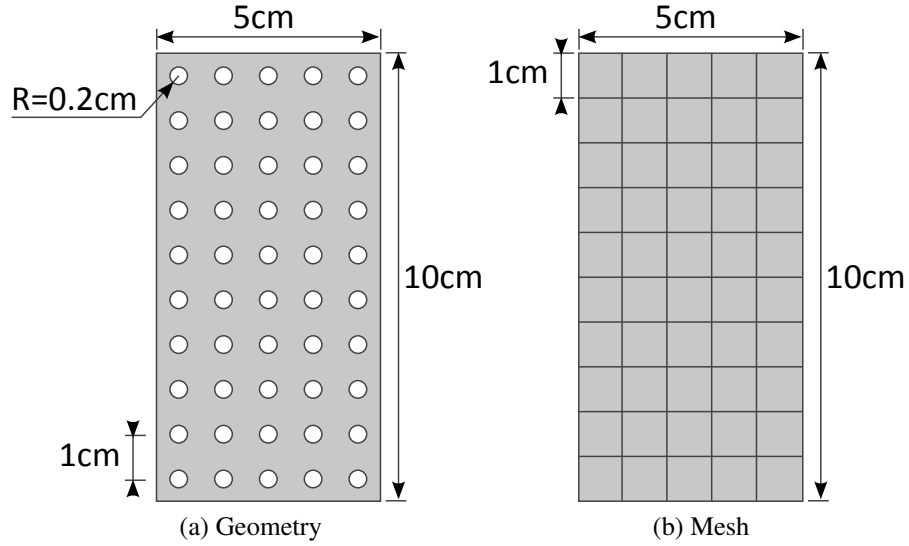


Figure 3.11: A short bar used in example 3.3.2. It contains a uniform array of 5 by 10 holes of diameter 0.4 cm . The mesh corresponding to this geometry does not contain the holes since they are considered part of the fine scale.

C , $J = \det(F)$ is the Jacobian of the deformation tensor F , μ is the shear modulus of the material and κ is the bulk modulus of the material. The particular material properties for our model are $\mu = 0.9091 \text{ MPa}$ and $\kappa = 0.8333 \text{ MPa}$, which correspond to $E = 2 \text{ MPa}$ and $\nu = 0.1$.

The model is dynamic and undamped, and is initially at rest. We solve the system using an explicit central difference solver from the initial time $t_0 = 0 \text{ s}$ to the end time $t_{max} = 0.5 \text{ s}$.

We compare a multiscale finite element (MsFE) model with smart element models that use a MsFE as their base element. The coarse scale mesh in the MsFE is composed by a grid of square elements with side length 1 cm , and they have the fine-scale mesh from Example 3.2.1, which is shown in Figure 3.4. The fine-scale mesh is composed of traditional quadrilateral 2D elements with bilinear shape functions. The smart elements only possess a coarse scale mesh, which matches that of the MsFE model.

We test three different load cases. In all the scenarios the bottom of the beam is clamped and a non-follower load is distributed uniformly over the top surface of the beam. In Case

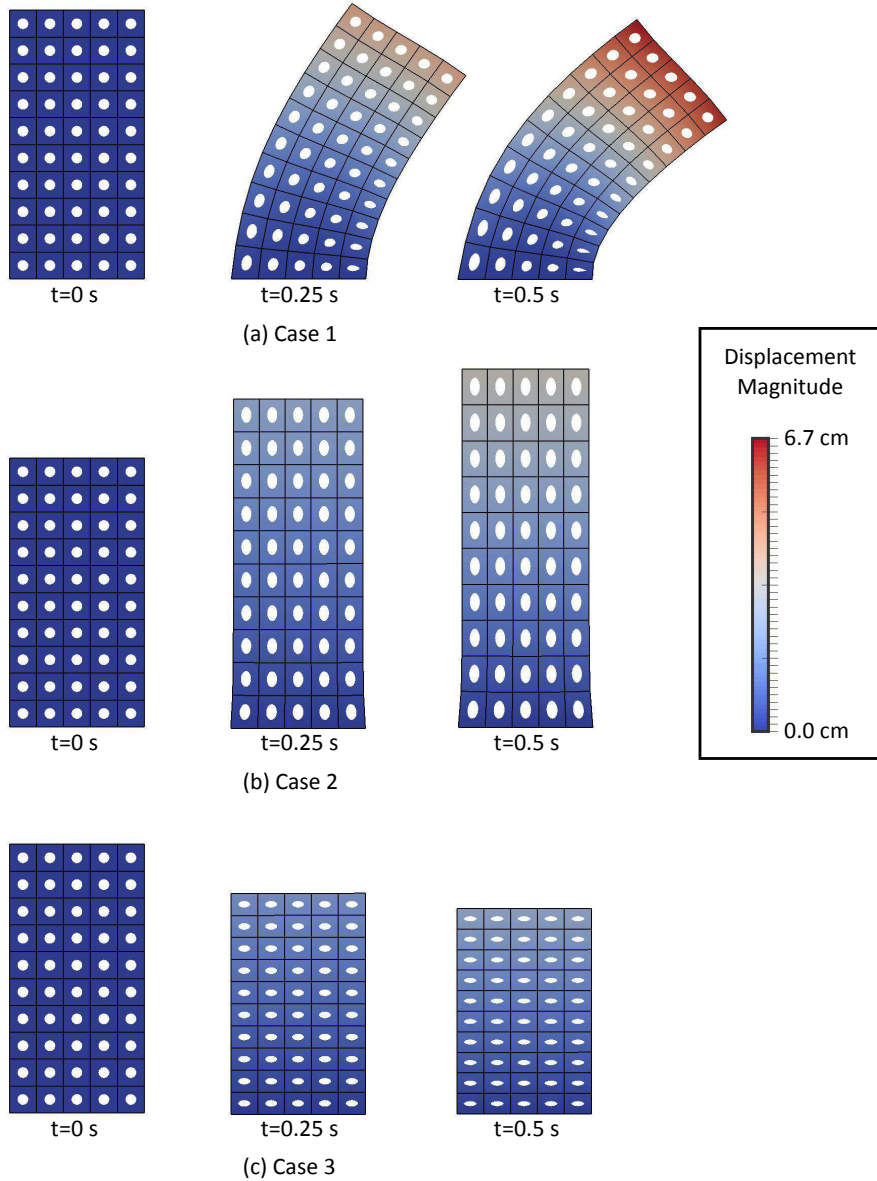


Figure 3.12: Comparison of the deformed configuration of a multiscale finite element model and a smart model for the three load cases in three different time instants. The first is displayed as a colored surface while the latter is displayed as a black wireframe. The shown smart model corresponds to ordinary polynomial regression of order 6 (Ord 6). Note that in the figure the exact MsFE solutions and the smart element solutions are superimposed.

1 the load has a magnitude of 2.5 kN and is applied in the direction of $[1, -1]^T$. In Case 2 the load has a magnitude of 20 kN and is in the direction $[0, 1]^T$. In Case 3 the load has a magnitude of 20 kN and is in the direction $[0, -1]^T$. In addition, in Case 3 we constrain the

lateral movement of the sides of the beam to avoid bifurcations in the solution. We apply the loads over a time interval starting at t_0 and ending at $t_{load} = 0.4 s$ with a magnitude varying according to

$$P(\tau) = P_{max}\tau^3 * (10 - 15\tau + 6\tau^2) \quad (3.12)$$

Where $P(\tau)$ is the magnitude of the load as a function of a non-dimensional time $\tau = t/t_{load}$, and P_{max} is the maximum load magnitude for the particular load case. After $t = t_{load}$ the magnitude of the load remains constant and equal to P_{max} . Note that this load distribution is smooth in the sense that both the first and second derivatives are continuous everywhere. For simplicity, we use non-dimensional magnitudes and assume that they correspond to meters for distances and Newtons for forces.

For this particular problem, the input vector λ_e of the smart elements contains only the nodal displacements. It is not necessary to include other information because all other parameters are the same across elements. With the exception of linear regression models, we consider only corotational elements and we use all 8 components of the corotational displacement as the input vector $\hat{\lambda}_e$. To calculate the rotation, we apply a polar decomposition to the deformation gradient at the center of the element, which we obtain using the bilinear test functions of the base element. We also use the average nodal position as the center of the element and the average nodal displacement as the translation of the element. Then we use Expression 3.10 to obtain the corotational displacements.

We analyze cases with and without balanced internal forces: in the former, we use the predicted nodal forces directly; in the latter, we subtract the out-of-balance forces and moments from the predicted nodal forces to enforce equilibrium. To achieve it, we assume that the error \bar{e}_e is a linear combination of 3 vectors, as described in Section 3.2.2. Two of the vectors correspond to forces equal to $[1, 0]^T$ and $[0, 1]^T$ applied to all nodes in the element. The third vector represents a moment in counter-clockwise direction generated with two pairs of opposite forces, each one applied on a pair of opposite nodes.

We test linear regression models that correspond to ordinary least squares regression of order 1. Their samples are of size 8 and contain nodal displacements extracted from a uniform distribution $u_i \sim U(-1e-6, 1e-6)$ as the input, and the resulting nodal forces on a single MsFE as the outputs. Since these displacements are small, the induced rotations are also small and the material behaves linearly for practical purposes. From each linear model, we produce a corotational and a non-corotational element. The only difference in both cases is the input vector during prediction, which contains the corotational displacements u_{ed}^I for the former, and the displacements u_e^I for the latter. We can use a model trained with non-corotational displacements within an element that is corotational because the rotations and displacements in the training sample are small.

We also test a variety of nonlinear corotational models based on support vector regression, ordinary least squares regression of order 6, and a combination of SVR and linear regression. Their samples are of size 10000 and contain corotational displacements as inputs and the corresponding forces in a MsFE as outputs. To generate the data, we obtain the nodal displacements in a fixed reference frame using a uniform distribution $u_i \sim U(-5e-3, 5e-3)$, and extract the corotational displacements from those values. The outputs are the nodal forces of the MsFE for those corotational displacements.

The polynomials in the ordinary least squares regression models do not possess an independent term to avoid non-zero forces at zero displacement. In the case of SVR we use radial basis functions as the kernel, and $\epsilon = 20$, $C = 1e5$ and $\gamma = 1e4$ as the hyperparameters. The mixed models use the corotational linear regression model to obtain the forces when $\|\lambda_e\| < 4e-4$ and the SVR model to obtain the forces when $\|\lambda_e\| > 8e-4$. The transition when $4e-4 < \|\lambda_e\| < 8e-4$ is given by

$$f_i = f_i^L(\lambda_e) * (1 - P(\tau)) + f_i^{SVR}(\lambda_e) * P(\tau) \quad (3.13)$$

where f_i , f_i^L and f_i^{SVR} are the forces predicted by the mixed model, the linear regression model, and the SVR model, respectively, $\tau = (\|\lambda_e\| - 4e-4)/4e-4$, and $P(\tau)$ is

given by equation 3.12 using $P_{max} = 1$.

We generate 10 data samples for linear regression models and 10 data samples for other models, and reuse them for the different load scenarios and learning algorithms. We measure the error in the models using

$$\epsilon_{max}^u = \frac{1}{u_{max}} \max_t \left(\sum_{i=1}^m \frac{\|u_i^r(t) - u_i(t)\|^2}{m} \right)^{\frac{1}{2}}$$

where $u_{max} = \max_t(\max_i(\|u_i^r(t)\|))$, and $u_i(t)$ and $u_i^r(t)$ are the vectors containing the displacements at node i and time t in the smart model and the reference MsFE model, respectively. We show the deformed configuration of the models for some selected time instants in Figure 3.12, and the results comparing the error on the different smart elements in Figure 3.13. Although the problem is dynamic, we point out that the stress waves in the model are much larger than the element length. As such, the dynamic effects inside the fine scale are negligible, and the assumptions made in MsFEM hold valid for this particular problem. It is worth mentioning that the same would apply to any traditional FEM formulation.

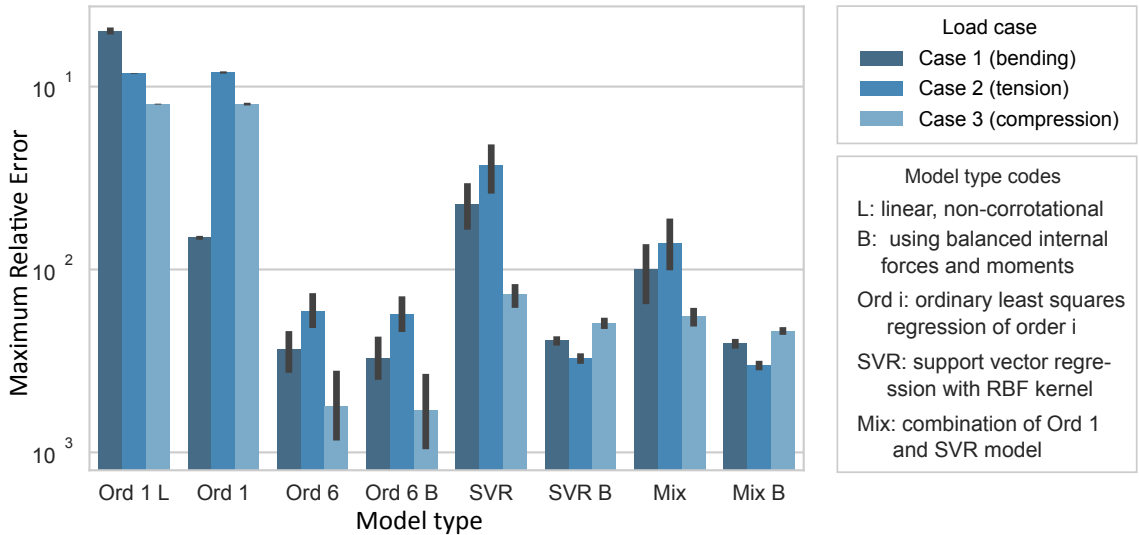


Figure 3.13: Maximum Relative Error ϵ_{max}^u in the displacements for different smart element types and load scenarios.

As expected, the linear regression model (Ord 1 L) produces large errors. In particular, the bending case involves large rotations which lead to artificial volume changes in the elements. The corotational version of that same model (Ord 1) solves the problem of finite rotations and the resulting error is very small for bending, but the error remains unchanged in the other cases. Without considering the smart elements with balanced internal forces, the one that produces the smallest error is the ordinary regression with polynomials of order 6 (Ord 6). In general, SVR performs poorly because the error in the predicted forces of the elements is absolute and not relative. That means that the models have an initial out-of-balance force that is instantly applied to the structure. Since the model is undamped, the vibrations generated at the beginning of the simulation remain throughout the entire simulation. Although the mixed linear-SVR model (Mix) solves that issue, its error remains larger than in ordinary regression of order 6. One of the main advantages of this mixed element is that for small deformations the force prediction cost is the same than for ordinary regression of order 1, which is considerably lower than both SVR and higher order ordinary regression. While balancing the internal nodal forces produces improvements in all the models tested, it has the greatest effect on the SVR-based models. In those cases, we observe similar error than in ordinary regression of order 6, and at the same time, a much lower variance. That means that the SVR balanced models (SVR B and Mix B) are more independent of the particular training sample and produce more predictable levels of error.

In addition to the reduced computational cost, smart elements present one additional advantage with respect to their base element. When obtaining the static solution to the fine-scale problem in a MsFE, it is possible that the model does not converge. If the amount of elements in the coarse scale model is large, ensuring that all fine-scale models converge for large deformations is difficult. Smart elements tend to be more stable because, once trained, they do not need to solve the fine-scale problem. If the base element fails during the sample generation, that particular training case can be discarded and the behavior during prediction is estimated from other training cases in the proximity.

A more generic smart element

The example in the previous section considers only displacements as inputs of the smart element. Now we solve a problem with the same structure, loads and material properties ($\mu = 0.9091 MPa$ and $\kappa = 0.8333 MPa$) as in the previous section, but we consider a more generic smart element whose inputs also include the material properties μ and κ . That is, the smart element now automatically works for a range of material properties. We extract the displacements from the same probability distribution used in the previous case, and the material properties from the uniform distributions $\mu \sim U(5e5, 5e6)$ and $\kappa \sim U(5e5, 5e6)$. Since the displacements and the material properties possess different orders of magnitude, we scale all the components of the input vector λ_e so that they fit in the range $[-1, 1]$. Finally, since the number of components is larger than in the previous section, we increase the size of the training sample to $N_s = 20000$.

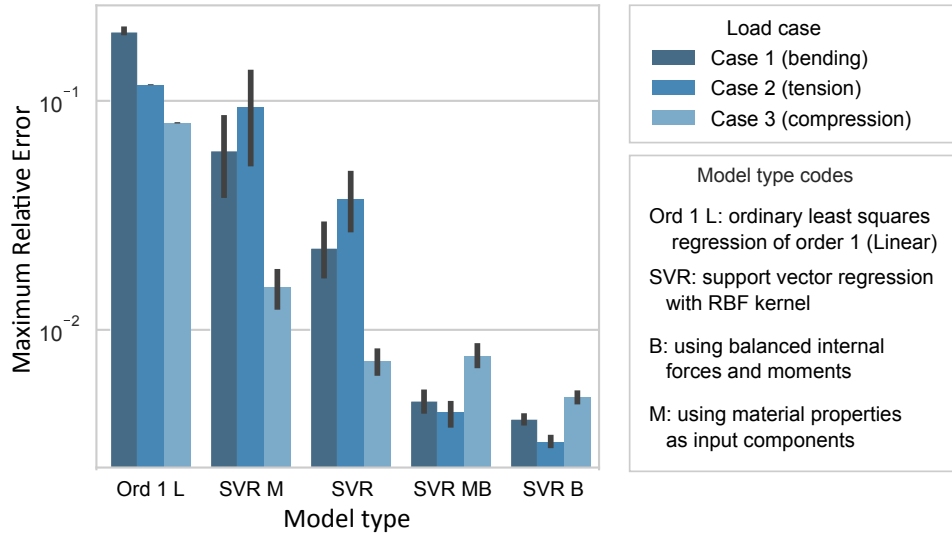


Figure 3.14: Comparison of the Maximum Relative Error ϵ_{max}^u in the displacements of the smart elements when including or excluding consideration of the material properties in the input vector.

In this case we only consider an SVR model and a balanced SVR model with hyperparameters $\epsilon = 20$, $\gamma = 0.5$, and $C = 2e4$. We generated the models using 10 different samples and tested the resulting 10 models in all 3 load scenarios. The results are shown in

Figure 3.14, where we compare the error with some of the previous models. As expected, the observed error and variance are slightly larger than for the SVR models that did not consider the material properties as inputs. In the particular case of the non-balanced element, the increase in error over the already poor performance of SVR means that the smart element does not perform significantly better than in the linear regression case. In the case of balanced elements, the increase in error and its variance is not significant and the element (SVR MB) considerably outperforms the linear regression model. In summary, we see that by simply increasing the sample size, we can obtain a smart element that works for a range of material properties without sacrificing much in terms of accuracy or precision.

3.4 Application to material modeling

Although the technique we developed was originally intended only to model finite elements, we can also apply it to material modeling. Consider the less general case of either a triangular finite element or the case of a material point in a homogenized multi-scale finite element. In the former case, if the shape functions are linear, then the strain field is constant throughout the element. In the latter, the fine scale model at each integration point is evaluated as if the strain surrounding it is uniform. Those two cases are just examples where the state of the material is a single strain value that has a one to one correspondence to the deformation components of the displacement in the model u_{ed} . That is, we can generate a reversible map $\epsilon(u_{ed})$, where ϵ are the strains. Note also that the number of u_{ed} components is always equal to the number of strain components. In the same way, the number of independent components of the force vector in the element model has the same length than the stress components, and we can find a reversible map from one to the other. This means that if we can find a surrogate model for such an element $\hat{f}_e(u_{ed})$ and the mappings from the strains to the displacements $u_{ed}(\epsilon)$, and from the forces to the stresses $\sigma(\hat{f}_e)$, we can also produce a surrogate material model $\bar{\sigma}(\epsilon) = \sigma \circ \hat{f}_e \circ u_{ed}(\epsilon)$.

Producing such a surrogate material model can be useful, but we can also achieve a

similar result in a much easier way. If we produce the mappings $\epsilon(u_{ed})$ and $\sigma(\hat{f}_e)$, we can modify the dataset $\mathcal{D} = \{u_{ed}^i, \hat{f}_e^i\}_{i=1}^{N_s}$ from the previous sections to generate another set $\mathcal{D}_m = \{\epsilon^i, \sigma^i\}_{i=1}^{N_s}$, where $\epsilon^i = \epsilon(u_{ed}^i)$ and $\sigma^i = \sigma(\hat{f}_e^i)$. Then we can train the machine learning model using this new dataset to directly obtain a surrogate for the material. In some cases, we can also produce the values of σ^i in the dataset directly and without having to first obtain the nodal forces. We show an example of this process in Section 3.5.3 where we analyze a history dependent multi-scale material model.

3.5 History dependent problems

In previous sections, we used machine learning methods to predict the forces of finite elements using information from their current state. Through examples, we demonstrated that for many problems that approach provides excellent results. In this section we expand that concept to consider the history of the element's state. This will allow us to solve history dependent problems in a more effective way.

Our objective in this section is the same than in the rest of the thesis. We want to obtain the forces that drive the degrees of freedom of an element. Those degrees of freedom are a chosen subset of values from the state of the element, which contains all the information in the domain of the element. For example, the state of the element may contain the distribution of displacement, strain, stress, plastic deformation, damage, temperature, and any other field or information inside that domain, and the degrees of freedom could be just a finite number of those strain values. In history-independent problems, a particular value of the degrees of freedom produces a unique value for the driving forces, and it is assumed that all other information in the domain of that element can be obtained as function of the degrees of freedom. In history-dependent problems, the situation is more complex. The expression of the driving forces may depend on an arbitrary amount of information from the entire state of the element. That information is usually problem dependent, and it is frequently difficult to figure what kind of information is relevant. For example, in the case of an RVE

that has an elastoplastic matrix with stiff particles, the average stress in the element depends on the distribution of plastic deformation in the entire domain. We can extract some measures of that plastic deformation, but this should be done carefully as we may lose relevant information in the process. As a consequence, for many history-dependent problems we cannot directly apply our method from Section 3.2. Doing so would require us to carefully study the particular problem at hand and to make specific assumptions about the element response. This is against the philosophy of our method because we would need a new formulation for each particular problem. Even assuming that we do know how to choose the relevant properties, if they are a large number, they would be expensive to predict using machine learning, and predicting them would be wasteful because that information is not necessarily relevant to the user.

Although we cannot obtain the driving forces just from the current values of the degrees of freedom, we could use the history of those values to perform a simulation of the element up to its current state and extract any desired information. Continuing with the previous example of an RVE, if we knew the history of the strain imposed to the cell, we could replicate the experiment by generating an undeformed cell and imposing those strains to its boundary. As a result we would reach the current configuration of the element and we could extract any desired measure of the plastic deformation. This means that the information of the problem description together with the history of the degrees of freedom contains all the information about the current state of the element.

Next in this section, we propose to use the history of the degrees of freedom or a subset of that sequence as the input of our smart elements. This kind of input has enough information to predict the driving forces even in history-dependent problems. More interestingly, we will show later in this chapter that this method allows us to recover localization information from multiscale problems, something that is lost in homogenization alternatives. In addition, we will show ways to systematically generate the necessary data and suggest machine learning algorithms that are appropriate for this kind of problem.

3.5.1 Method

Assume that we possess a computationally-expensive history-dependent finite element (FE) or representative volume element (RVE) type that we want to work with. If the finite element model that contains them is large, the use of those high-fidelity elements may be prohibitive, and we would like to reduce their computational cost. In particular, we want to follow the ideas from Section 3.2 and use a surrogate model to achieve our objective. That is, given a function that produces the element's driving forces, we would like to extract data from that function and use the data to train a machine learning model. That machine learning model should be a computationally-cheaper approximation of the driving force function of the high-fidelity element.

In this particular case, we are dealing with a history-dependent problem so we need to consider the time t into account. We use a discrete time sequence $t_s = \{t_i\}_{i=0}^{N_t}$, where $t_i \in \mathbb{R}$ increases with the time index $i \in \mathbb{N}$. The vector containing DOFs of the element at time t_i is $x_i \in \mathbb{R}^{n_x}$ which includes the displacements or strains depending on whether we are considering a FE or a RVE, respectively. In addition, that vector can contain any other known state variable or parameter that we deem important. The driving forces for the DOFs of the element at time t_i are given by $y_i \in \mathbb{R}^{n_y}$. Since we are considering a history dependent problem, y_i is a function of the sequence of inputs up to that instant $\{x_j\}_{j=0}^i$. This means that every y_i function is different because the length of their input sequence increases with time.

After consecutively applying each input sequence $\{x_j\}_{j=0}^i$ to its corresponding driving force function y_i for all time increments, what we obtain is a sequence of input vectors $x = \{x_j\}_{j=0}^{N_t}$ and a sequence of output vectors $y = \{y_j\}_{j=0}^{N_t}$. These two sequences represent the complete history of our element, and when compared with the data from previous sections, they are equivalent to a single input-output pair (x, y) . To train a machine learning model we need a dataset that contains many of those pairs, which we usually represent as $\mathcal{D} = \{x^k, y^k\}_{k=1}^{N_s}$ where x^k and y^k are particular instances of the sequences x and y , respectively.

If the input sequence x^k is random, we say that each pair (x^k, y^k) is a realization of the process. In next section, we present a way to generate random input sequences x^k assuming that they are realizations of a Gaussian process (see Section 3.1.3).

What we now need is to form the output sequence y^k that corresponds to that input sequence x^k . The only way to generate such data is to actually generate the specified high-fidelity element, perform the history dependent simulation from time t_0 to t_{N_i} and to track the variables that form our output vector y_i^k . In that simulation, we need to impose our input vector x_i^k onto the state of the element. Since there is no interaction with external sources, our input vector contains enough information to perform a single-element simulation. The output variables can be variables at the macro level, such as average stress, but they can also correspond to local information such as the maximum von Mises stress or plastic deformation. In fact, we can store as output any variable that can be measured during the simulation.

Once we produced the dataset \mathcal{D} , we need to train the smart element, which is the surrogate of the high-fidelity element. However, since the length of the input sequence changes over time, using conventional machine learning methods is not effective and it is better to use specialized techniques that take into account temporal information. Problems where we have a sequence of inputs and we want to produce a sequence of outputs are called sequence-to-sequence. At the time of this writing, the dominant machine learning method for sequence-to-sequence problems is recurrent neural networks (see Section 3.2), but in principle we could use any specialized method. Training a recurrent model is similar to training a conventional one in the sense that we provide them a dataset in order to produce a function $\bar{y}(x)$. The difference is that in this case, the function has a number of hidden parameters or a memory, which makes it behave as if the i -th call to that function is an approximation to $y_i(x_i)$. The configuration of the particular machine learning model is problem dependent, so we provide later a simple example to illustrate the process in Section 3.5.3.

3.5.2 Input sequence generation

In the previous section, we proposed to use the history of the DOFs of the elements as the input of our smart elements and recurrent neural networks as their machine learning algorithm for history dependent problems. In this section we show how to generate those input sequences for the training dataset.

The input in our sequence-to-sequence problem is simply a sequence of vectors that represents the evolution of the DOFs of our model over time. That is, given a sequence of time instants $t_s = \{t_i\}_{i=0}^{N_t}$, we need a sequence of vectors $x_s = \{x_i\}_{i=0}^{N_t}$ where $x_i \in \mathbb{R}^{n_x}$ are the DOFs at time t_i . In addition, we want these sequences to be representative of possible deformation paths of our elements. This means that the values at different instants of time are not independent, and the closer two instants are, the more correlated their values should be.

We propose to generate the sequence x_s by assuming that it belongs to a Gaussian process $x(t) \sim GP(m(t), \kappa(t, t'))$, and to use the known properties of the state variables to specify the characteristics of the process. Those characteristics are the mean function $m(t)$, the covariance function $\kappa(t, t')$ and its characteristic length, the amplitude of the process, and the initial conditions of the system. In our particular problem, the state vector contains the strains or displacements of our system, and we need to choose accordingly. Once the process is defined, we can choose any desired time sequence t_s and produce a realization of the process by evaluating it at those time increments to produce x_s . Next, we describe how to choose each of the characteristics of our Gaussian process.

First, we need to specify the mean function and the covariance function of the process. Unless we have knowledge about the expected average movement of the system, we can safely assume that the mean function $m(t)$ is zero. This does not mean that the vectors x_s that we produce have an average value of zero, only that its expected value is zero at any instant. The covariance function is the most important decision because it determines the kind of behavior our process has. We can choose any of the typical functions mentioned in

Section 3.1.3, or any other function that we believe is a good fit. For example, if we believe that the system has a smooth behavior, then a squared exponential covariance function could be a good fit. Since we are assuming a particular covariance function, it is a good idea to test that the realizations of the process are consistent with the possible behavior of x_s .

After we define the covariance function, we need to define its hyperparameters. In most of the typical cases, they possess a property called characteristic length. To define its value we can look at the number of expected deformation cycles in the simulations we want to perform. For example, if the kind of simulation we need is not cyclic, the covariance function could have a characteristic length that is in the same order of magnitude than the simulation length t_{N_t} . If the state variables may perform several cycles, then the characteristic length should be considerably shorter than the simulation so that the variables have an opportunity to perform several cycles.

Another hyperparameter that we need to define is the amplitude of the process, which defines the spread of the possible values in the function. We know that for any single time instant t_i the distribution of the state x_i is Gaussian and that its standard deviation is given by the covariance function. Thus, we can define the amplitude of the process to ensure that with some desired probability, our sample does not reach the maximum value allowed on the state variables. For example, if the state of the system is defined by the strains, we could choose that with 99% probability, our system is below the ultimate strain.

Using a Gaussian process with the previous distribution is still not sufficient since we also need to specify the initial conditions. The easiest way to enforce those constraints on the process is to use the Gaussian process regression detailed in Section 3.1.3. If the initial condition of the system is a particular value x_0 , we can use the data $\mathcal{D} = \{0, x_0\}$ and fit our system using Expression 3.5. If we need to also constrain the derivative of the state variable, we can do so approximately by specifying another point slightly before the initial time. For example, assuming that we want to have a derivative and initial value of zero we

could fit the pairs $\{0, 0\}$ and $\{-t_s, 0\}$, where t_s is much shorter than the simulation time. We show an example of this procedure in Section 3.5.3, where we generate the sequence of macroscopic strains that are applied to the fine-scale model in a multi-scale material.

If the initial conditions are complex and the suggested method does not work, one can simply draw samples from the original distribution, and only keep the cases that fit our desired constraints. This procedure is inefficient, and it is unlikely to yield samples that satisfy our constraints with precision, so we should only use it as a last resort.

Finally, if we desire to have a rich set of samples that represent the behavior of the system under many different circumstances, we can simply produce them using a combination of sources. We could select a number of covariance functions and designate each of them to a proportion of the samples, and then use probability distributions over their parameters so that the samples represent a wide range of situations.

3.5.3 Example: Multi-scale Material

In this problem, we study a simple material model that homogenizes the plane strain fine-scale finite element model from Figure 3.15. Just like with any other material model, given the strain, we want to obtain the corresponding stress. The strain is an input imposed by the surrounding material onto to the cell boundary, and the stress is the average value over the fine-scale model. To enforce the specified strain value, we constrain the boundary of the finite element model so that its nodes follow the imposed strain. For this particular problem, we are also interested in the values of the maximum plastic equivalent strain and the maximum von Mises stress in the fine-scale model.

The fine-scale problem structure is defined on a square domain of side length $l_s = 2mm$ and possesses a particle of stiff material embedded into a softer matrix. The matrix has an elastic perfectly-plastic material [60] with elastic modulus $E_m = 100MPa$, Poisson's ratio $\nu_m = 0.1$, and yield stress $\sigma_{ym} = 0.5MPa$. The particle has a circular shape with radius $r_p = 0.6mm$, and its material is elastic, with elastic modulus $E_p = 500MPa$, and Poisson's

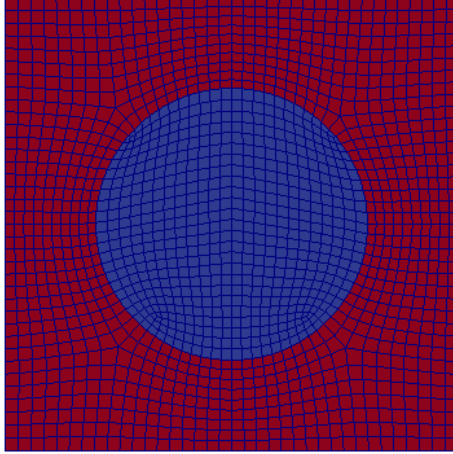


Figure 3.15: Illustration of a 2D plane strain finite element mesh. It contains two materials: a soft elastoplastic matrix (red) and a stiffer elastic particle (blue).

ratio $\nu_p = 0.3$. Our objective is to generate a dataset extracted from that fine-scale model and to train a surrogate machine learning model using that data.

To simulate the interior of the fine-scale model we use an explicit central difference solver. Although this problem is history dependent, for low-frequency loads it is not rate dependent. This means that we can use a dimensionless time scale, and rescale it to the desired magnitude when necessary. The dimensionless time for this problem is $\tau = t/t_{char}$, where t is the time and t_{char} is the characteristic length of the process. Using a dimensionless time scale is not only convenient in this procedure, but also more efficient. If we perform simulations of the fine-scale model using the same time scale as for the coarse-scale model, and we use an explicit central difference solver for the fine scale, we would be required to take an extremely large number of time increments. The reason is that the total simulation time is dominated by the requirements of the coarse-scale, but the fine-scale time increments are much shorter than the coarse-scale time increments. By using a dimensionless time, we can simply perform the fine-scale simulation using the shortest simulation time that does not produce oscillations from dynamic effects.

The first step is to generate the sequence of strains for each data sample using the procedure from Section 3.5.2. We use an independent Gaussian process for each component

of the strain and assume that the coarse-scale model is loaded slowly, meaning that our imposed strains over the material do not have high-frequency components. This assumption lead us to choose a squared exponential covariance function for the Gaussian process. Since we are working with dimensionless time, the characteristic time scale of the input is 1, and the total simulation time is 4. For simplicity, we choose that the strain components have the same amplitude, which is equal to the yield strain $e_y = s_y/E$. We could have, for example, chosen a smaller magnitude for the shear component. In addition, we constrain the Gaussian processes using an initial value and a first time derivative equal to zero so that the fine-scale model is initially undeformed and at rest. We generate 10000 such samples for the training dataset and 2000 for validation.

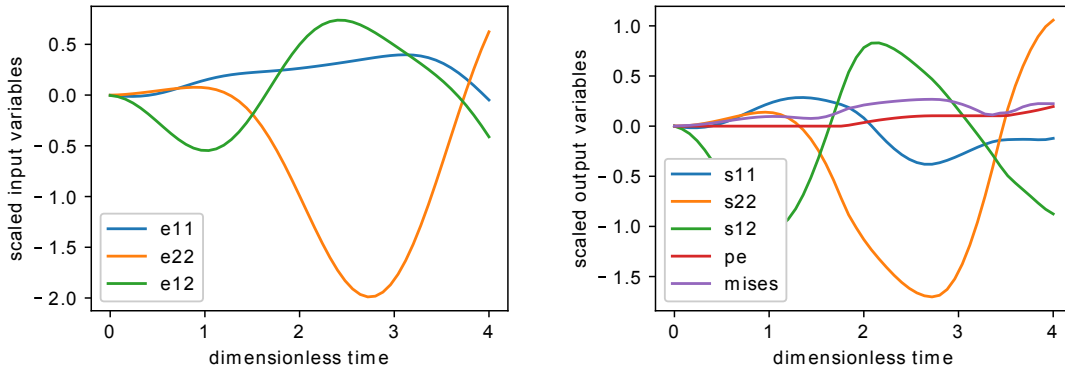


Figure 3.16: Example of input and output sequences for a single sample in the training set. (left) Input sequences. The curves represent the average strain components (e_{11} , e_{22} , e_{12}) as a function of the time. (right) Output sequences. The curves represent the average stress components (s_{11} , s_{22} , s_{12}), the maximum equivalent plastic strain (pe) and the maximum von Mises stress ($mises$) as a function of the time.

The next step is to use each generated input sequence (the strains) to simulate the fine-scale finite element model described above, and then extract the desired output values from the resulting deformed configuration. In this particular problem, we need to extract the average stress, the maximum von Mises stress, and the maximum equivalent plastic strain. As a result, the output vector for each instant in the output sequence has length five. The

number of time increments on each simulation depends on the conditional stability of the central difference solver. In this case, that number is larger than what we need to train the machine learning model, so we extract 51 uniformly distributed data points over the simulation span. Once we obtain the output sequences we normalize both the input and output components. For the input, we normalize the standard deviation to 1. For the output, we normalize the standard deviation of the average stresses to 1 and the standard deviation of the other components to 0.1. Having a low error in the stress values is critical for obtaining accurate results when using the neural network as a surrogate material in other simulations. The other components of the output do not need to be as accurate because they would not affect the deformation of the coarse-scale model and are only used for post-processing and failure analysis. This unbalanced normalization allows us to spend more resources on reducing the error of the most important output variables while producing a reasonable estimation of the others. We show an example of the input and output sequences in Figure 3.16.

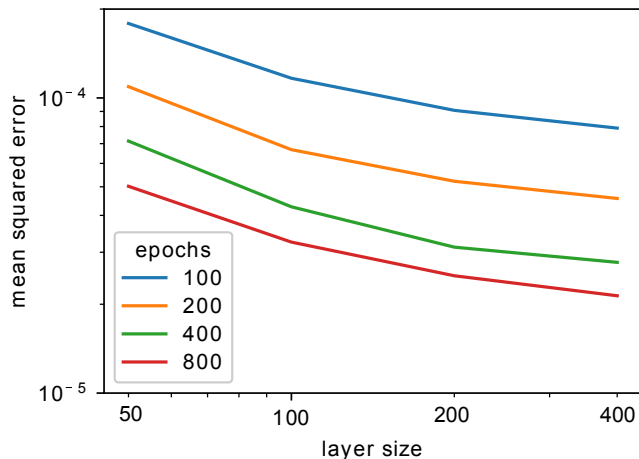


Figure 3.17: Error of recurrent neural networks trained on material data for different layer sizes and number of training epochs. In all cases, the error indicated for epoch i represents the best score achieved during training until that epoch.

For the machine learning model, we generate a number of recurrent neural networks consisting of three LSTM layers followed by a linear time-distributed fully-connected feed-

forward layer. All the LSTM layers in a given model have the same size N_L , and we choose different values for that size in order to analyze the behavior of the layer size on the error. The final layer has size 5 which is equal to the output size. We choose a mean squared error metric and optimize the model using Adam, which is an adaptive gradient descent method [61]. We train the model using with mini-batches of size 20 and performed a total of 800 epochs (passes over the entire dataset) so that we can compare the effect of the training epochs on the error.

In Figure 3.17, we show the error of our neural networks on a test dataset of size 2000 for all the different layer sizes and a varying number of training epochs. In all of the cases, the error over the test set is similar to the error on the training set (not shown in the plot), thus they do not indicate overfitting. The plot shows that we can considerably reduce the prediction error by increasing either the layer size or the number of epochs, but because of their diminishing returns, the optimal solution given a specific training cost should be achieved by increasing a combination of both. On the other hand, from the perspective of computational prediction cost, it is always better to increase the number of epochs than the layer size because that cost is only dependent on the latter. We also show some examples of the output values on test data and compare them with their prediction in Figures 3.18 and 3.19 for the case with layer size 400 and trained over the 800 epochs. Those plots show that the behavior of all output variables is predicted with great accuracy. From a practical perspective, the error in the average stress is small enough to be acceptable for most applications. The relative error in the other components is perceivable, but they are capable of showing the behavior trends, and they could be a reasonable choice in many circumstances including failure analysis.

In conclusion, recurrent neural networks are a promising method for generating accurate surrogate models for history dependent multi-scale materials. The main additional difficulty is the need to generate input sequences instead of input values, but we were able to easily overcome that obstacle using Gaussian process regression. In addition, we suc-

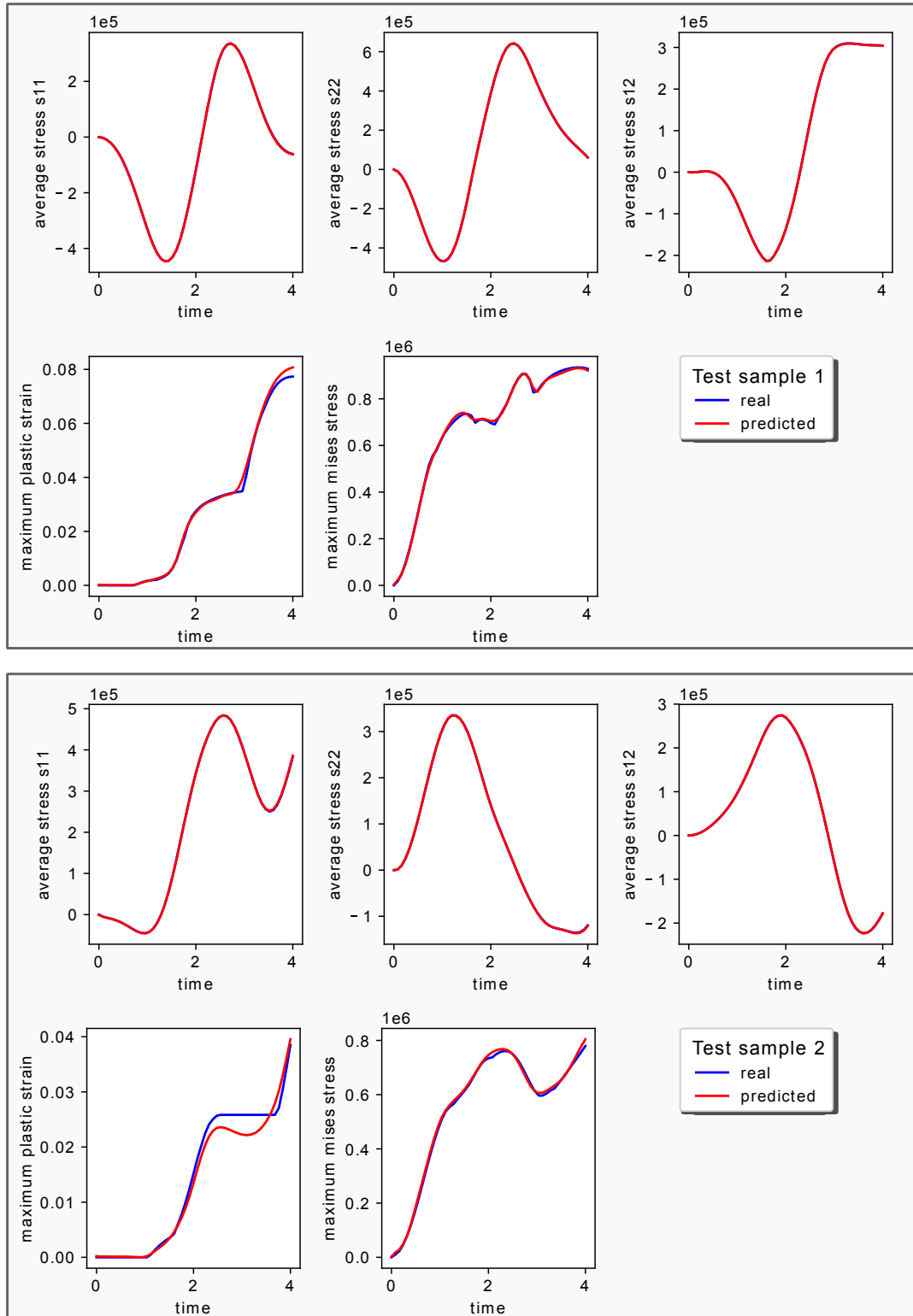


Figure 3.18: Selected output components test samples 1 and 2 compared with the machine learning prediction.

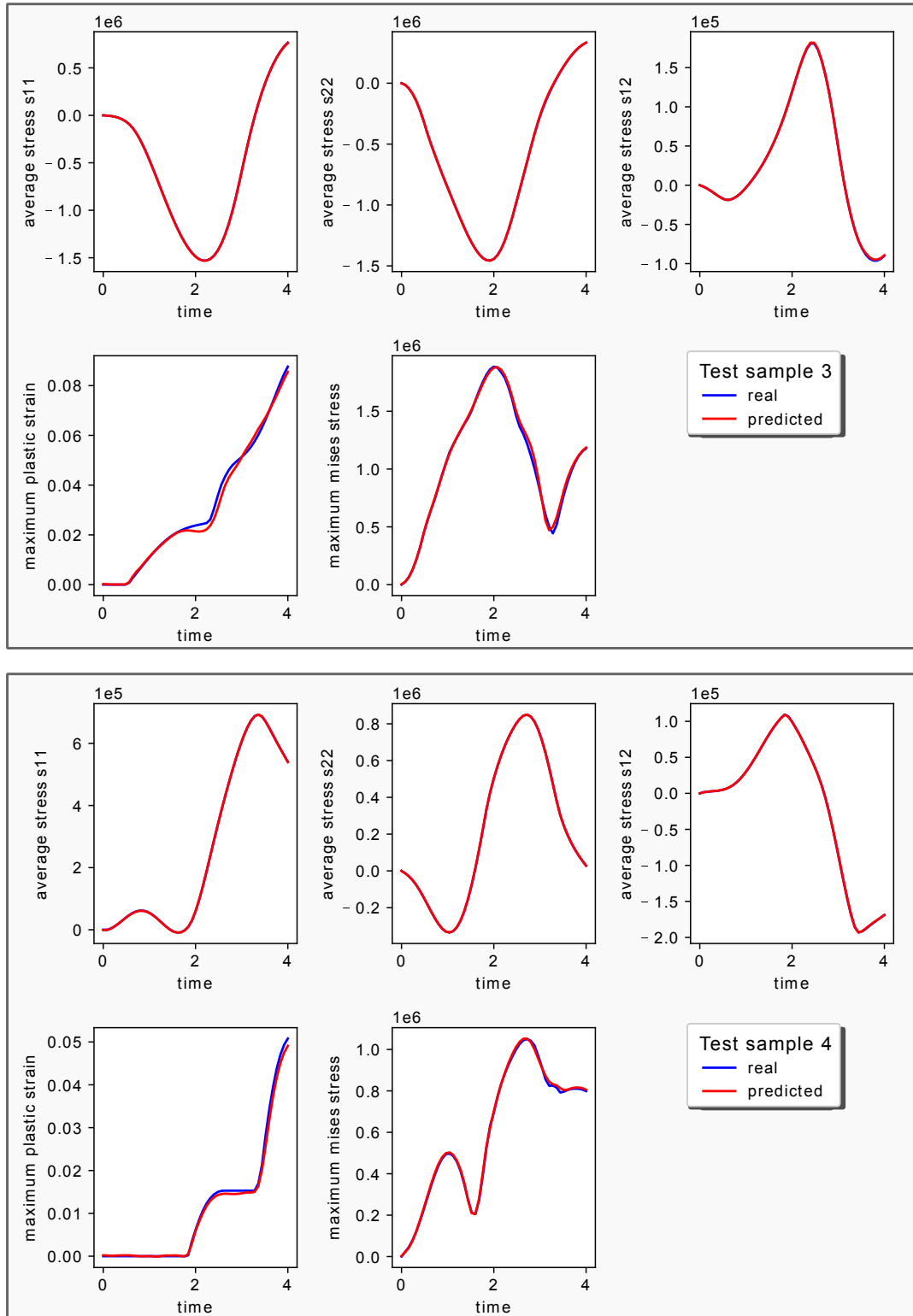


Figure 3.19: Selected output components for test samples 3 and 4 compared with the machine learning prediction.

cessfully predicted the output variables of interest, including localized information from the fine-scale model that is usually lost in homogenization methods.

3.5.4 Example: Multiscale Beam

In this section, we demonstrate the use of recurrent neural networks for history dependent problems by studying a multi-scale 2D beam. The structure has height $0.2m$ and width $0.1m$ and it has the metamaterial described in the previous section. That is, the material is formed by a uniform array of particles embedded in a softer matrix whose cell is shown in Figure 3.15.

In this particular problem, we solve the structure using a mesh with square elements of side length $2cm$ and using the simple material model described in the previous example. We compare the results with a finite element model using a surrogate material obtained using recurrent neural networks. This neural network has 3 LSTM layers of size 200 and it is trained for 800 epochs. The only difference with the recurrent model of layer size 200 from the previous section is that in this case, we trained it using 1001 datapoints per sample instead of 51.

The structure is clamped at the bottom and it has an instantaneous transversal load uniformly distributed over the top surface with a magnitude of $4kN/m$. Although the material is nonlinear, we did not consider geometric nonlinearity in this example. To solve the system we use a central difference explicit solver using time increments of length $5e - 6s$.

In Figure 3.20, we show the two finite element solutions at the instant of maximum displacement during the first period. The maximum error in the displacement is 0.5% which is low enough not to be perceivable to the naked eye in that image. Since the displacements depend on the stresses of the material model, this also indicates that the error in the stress output of the surrogate is low.

In Figure 3.21, we compare the state and outputs of the elements in the two finite

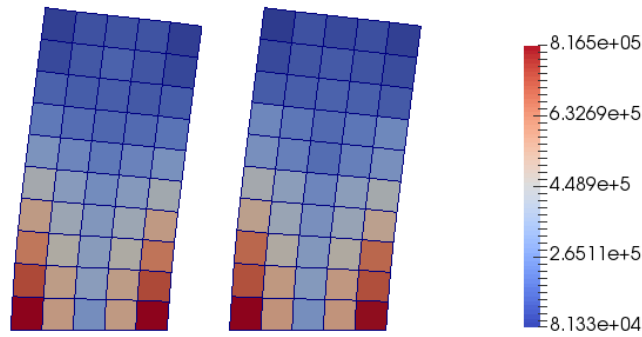


Figure 3.20: Comparison of the maximum von Mises stress in the fine scale of two multiscale finite element models. (right) Nonlinear multiscale finite element. (left) Surrogate model using recurrent neural networks. The displacement has been scaled 10 times for visualization purposes in both cases.

element models at the time instant of maximum displacement. One of them contains the high-fidelity multi-scale material model that we use as the baseline, and the other one contains the surrogate material model that we trained using recurrent neural networks. We observe a high correlation of the variables on the two models and a very small error in all the cases. In particular, the stress s_{11} and strain e_{11} have an extremely low error. The other components of the stress and strain have a larger relative error because they are affected by the dominant longitudinal magnitudes in the structure. The two bottom plots correspond to the maximum equivalent plastic strain and the maximum von Mises stress in the fine scale. Those variables also show remarkable accuracy, especially considering that they correspond to locations in the fine scale that may change over time. This is a valuable achievement because, at least to our knowledge, no other reduced-order technique can provide such information for arbitrary problems.

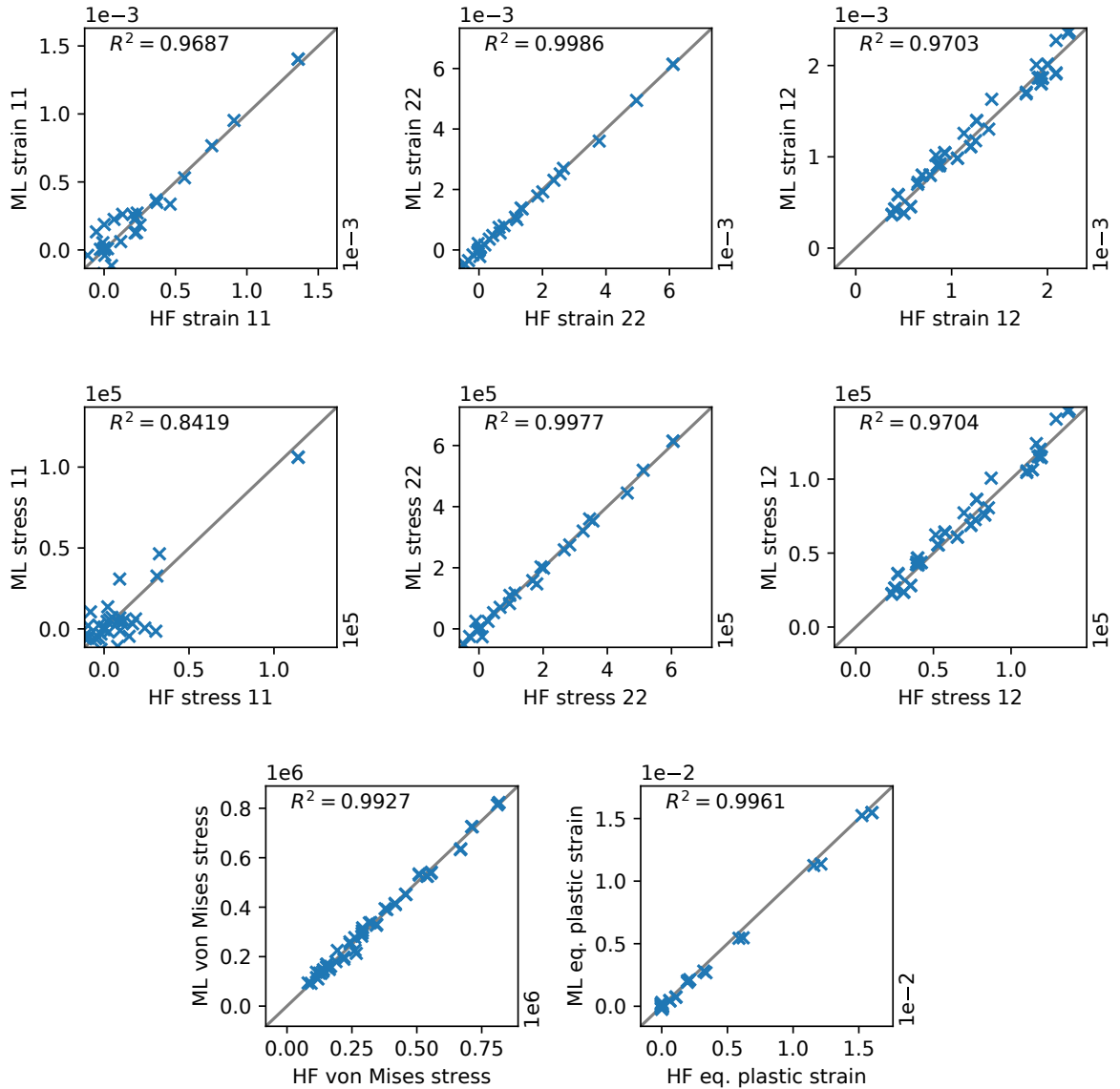


Figure 3.21: Comparison of the values observed on the elements of two multi-scale finite element models. One possesses a multi-scale material that contains a high-fidelity (HF) fine-scale model, and the other possesses a surrogate material generated using machine learning (ML). For reference, each plot also contains a line with unit slope.

CHAPTER 4

CONCLUSIONS

In the first part of this work, we presented a versatile finite element method aimed at modeling wave propagation in multi-scale structures. In our method, we split the set of shape functions into two subsets. On one side, the boundary shape functions provide a non-trivial solution at the boundaries and ensure continuity across elements. On the other, the interior shape functions provide a systematic way to control the stable time increments and to include the effects of heterogeneities and small features through the use of natural modes. We showed that boundary shape functions can be easily obtained through well established GMsFEM procedures, and that the interior shape functions can be obtained (1) analytically for simple configurations, and (2) through a modal analysis over the GMsFEM auxiliary mesh for more challenging geometries.

We tested the performance and convergence properties of the method through a series of selected problems. The results indicate that for wave propagation problems our method outperforms traditional FEM. More precisely, our method produced a smaller error for the same number of degrees of freedom, and its stable time increments were larger under the presence of small features.

We also provided an example application, where we used our method to study wave propagation on a two-dimensional periodic elastic domain. This kind of problem was particularly well suited for our approach due to the large difference between element sizes within the model. Consequently, our results were in very close agreement with those obtained through a traditional finite element model, but at a much lower computational cost.

This modal-based approach is similar to many other projection based reduced-order models, but we adapted it into a formulation of finite elements in which we can increase the amount of detail by including a larger amount of natural modes. Unfortunately, these modes

are only related to the linearized problem and a large number of them may be required in non-linear cases. Other projection based approaches struggle for the same reason: it is difficult to find or calculate a subspace that can represent every possible solution, and defining them in real time may be computationally expensive.

To address this issue, we proposed in the second part of our work a finite element method that focuses on finding a direct relationship between the inputs and outputs of its elements, thus avoiding the more complex tasks of finding the internal displacement field or performing numerical iterations. To achieve this goal, we first extracted data from an existing finite element and then fed that data to a machine learning algorithm to produce an approximate model of the element. This process is known as surrogate modeling, and it allows us to reduce the computational cost of a function at the expense of error in the output. We named the approximated models *smart elements*.

We also proposed two ways to improve the performance of the method. First, we showed that removing the rigid body component out of the displacement variables can considerably reduce the sample size and, consequently, the computational cost. Second, we used the internal equilibrium relationships on the elements to reduce the computational error. An additional consequence of these techniques is that our elements satisfy, by construction, frame indifference and conservation of linear and angular momentum. Although the selected physical constraint was given by the equilibrium equations, other relationships could also be used. For example, in some problems we could use dissipation relationships or other energy constraints to restrict the output of the smart elements. Using such constraints could further reduce the error in the internal forces of the elements and ensure that none of the desired physical relationships are violated.

Another strength of our method is that the learning algorithm is arbitrary. We can choose it depending on the particular type of problem to produce elements that fit a particular need. However, once trained, a smart element can be applied to many different scenarios without having to train the element again. We showed that it is possible to com-

bine different machine learning algorithms in one smart element to take advantage of each one's strengths. Modern machine learning techniques could take this a step further and produce general-use smart elements that are pre-trained for a variety of problem types.

We also demonstrated the use of the method through two in-depth study cases. In the first one, we solved a nonlinear 3D truss structure for varying levels of error in the element data, and we compared the results with a leading data-driven approach. These results indicate that our method produced accurate results even in the most challenging conditions: high sample error and low computational cost. In the second case, we solved a dynamic nonlinear 2D continuous multiscale structure under three different load scenarios. Here, we compared the effect of different machine learning algorithms on the smart elements, and we tested the effectiveness of load balancing. The results indicate that our method can produce low levels of error as long as the learning algorithm is appropriate for the particular case, and that the load balancing can considerably reduce the average and the variance of the error. We then solved the same problem using a smart element that was trained beforehand for a range of values in its material properties. This case served as an extra validation case and supported our idea of implementing general-use smart elements. A similar approach could be followed to pre-train the elements for other properties, such as geometric parameters or volume fraction.

The main downside in our method is that intermediate magnitudes of interest such as stresses and strains are not obtained during the evaluation of nodal forces. However, in most cases, these values can be obtained in a post-processing stage like in the example from Section 3.3.1. If that is not the case, a simple alternative is to add those variables of interest to the output of the training sample, so that they are predicted together with the forces. Since those values are only required on a small proportion of the solution steps, those components only need to be predicted on a small number of time increments.

Finally, we also showed how to apply our method in the case of history dependent problems. In those cases, one of the additional challenges is to be able to produce input

data. We proposed the use of Gaussian processes to generate the input sequences, and to use recurrent neural networks to model the sequence to sequence dataset generated in the element or material simulations. We demonstrated these techniques through two simple multiscale examples, and we showed that the proposed solution produces accurate results. In addition, our model was able to recover localized information for any desired variable of interest in the fine scale of the model.

In summary, we produced a method that uses machine learning algorithms to analyze data from existing finite element formulations and produce smart elements that closely replicate their behavior. Smart finite elements can be used in the same way than any other element (regular or multiscale). That is, they can be assembled using traditional techniques, and in conjunction with any conventional solver. As a result, we produced a flexible method that can reduce the computational cost of a variety of complex finite element formulations without sacrificing much in terms of accuracy.

REFERENCES

- [1] O. Lopez-Pamies, T. Goudarzi, and T. Nakamura, “The nonlinear elastic response of suspensions of rigid inclusions in rubber: I—an exact result for dilute suspensions,” *Journal of the Mechanics and Physics of Solids*, vol. 61, no. 1, pp. 1–18, 2013.
- [2] O. Lopez-Pamies, T. Goudarzi, and K. Danas, “The nonlinear elastic response of suspensions of rigid inclusions in rubber: II—a simple explicit approximation for finite-concentration suspensions,” *Journal of the Mechanics and Physics of Solids*, vol. 61, no. 1, pp. 19–37, 2013.
- [3] G. I. Taylor, “Plastic strain in metals,” *J. Inst. Metals*, vol. 62, pp. 307–324, 1938.
- [4] V. S. Deshpande and A. G. Evans, “Inelastic deformation and energy dissipation in ceramics: A mechanism-based constitutive model,” *Journal of the Mechanics and Physics of Solids*, 2008.
- [5] A. L. Gurson, “Continuum Theory of Ductile Rupture by Void Nucleation and Growth: Part I—Yield Criteria and Flow Rules for Porous Ductile Media,” *Journal of Engineering Materials and Technology*, 1977.
- [6] M. G. Geers, V. G. Kouznetsova, K. Matouš, and J. Yvonnet, “Homogenization methods and multiscale modeling: Nonlinear problems,” *Encyclopedia of Computational Mechanics Second Edition*, pp. 1–34, 2017.
- [7] J. Fish, K. Shek, M. Pandheeradi, and M. S. Shephard, “Computational plasticity for composite structures based on mathematical homogenization: Theory and practice,” *Computer Methods in Applied Mechanics and Engineering*, vol. 148, no. 1-2, pp. 53–73, 1997.
- [8] K. Matouš, H. Inglis, X. Gu, D. Rypl, T. Jackson, and P. H. Geubelle, “Multiscale modeling of solid propellants: From particle packing to failure,” *Composites science and technology*, vol. 67, no. 7-8, pp. 1694–1708, 2007.
- [9] H. Inglis, P. Geubelle, K. Matouš, H. Tan, and Y. Huang, “Cohesive modeling of dewetting in particulate composites: Micromechanics vs. multiscale finite element analysis,” *Mechanics of Materials*, vol. 39, no. 6, pp. 580–595, 2007.
- [10] K. Terada, J. Kato, N. Hirayama, T. Inugai, and K. Yamamoto, “A method of two-scale analysis with micro-macro decoupling scheme: Application to hyperelastic composite materials,” *Computational Mechanics*, vol. 52, no. 5, pp. 1199–1219, 2013.

- [11] J.-C. Michel and P. Suquet, “Nonuniform transformation field analysis,” *International journal of solids and structures*, vol. 40, no. 25, pp. 6937–6955, 2003.
- [12] A. K. Noor, “Global-local methodologies and their application to nonlinear analysis,” *Finite Elements in Analysis and Design*, vol. 2, no. 4, pp. 333–346, 1986.
- [13] F. Brezzi and A. Russo, “Choosing bubbles for advection-diffusion problems,” *Mathematical Models and Methods in Applied Sciences*, vol. 4, no. 04, pp. 571–587, 1994.
- [14] L. P. Franca, C. Farhat, A. P. Macedo, and M. Lesoinne, “Residual-free bubbles for the helmholtz equation,” *International journal for numerical methods in engineering*, vol. 40, no. 21, pp. 4003–4009, 1997.
- [15] T. J. Hughes, G. R. Feijóo, L. Mazzei, and J.-B. Quincy, “The variational multiscale method—a paradigm for computational mechanics,” *Computer methods in applied mechanics and engineering*, vol. 166, no. 1-2, pp. 3–24, 1998.
- [16] C. Farhat, I. Harari, and L. P. Franca, “The discontinuous enrichment method,” *Computer methods in applied mechanics and engineering*, vol. 190, no. 48, pp. 6455–6479, 2001.
- [17] C. Farhat, I. Harari, and U. Hetmaniuk, “The discontinuous enrichment method for multiscale analysis,” *Computer Methods in Applied Mechanics and Engineering*, vol. 192, no. 28, pp. 3195–3209, 2003.
- [18] J. M. Melenk and I. Babuška, “The partition of unity finite element method: Basic theory and applications,” *Computer methods in applied mechanics and engineering*, vol. 139, no. 1-4, pp. 289–314, 1996.
- [19] T. Strouboulis, I. Babuška, and K. Copps, “The design and analysis of the generalized finite element method,” *Computer methods in applied mechanics and engineering*, vol. 181, no. 1, pp. 43–69, 2000.
- [20] T.-P. Fries and T. Belytschko, “The extended/generalized finite element method: An overview of the method and its applications,” *International Journal for Numerical Methods in Engineering*, vol. 84, no. 3, pp. 253–304, 2010.
- [21] T. Y. Hou and X.-H. Wu, “A multiscale finite element method for elliptic problems in composite materials and porous media,” *Journal of computational physics*, vol. 134, no. 1, pp. 169–189, 1997.
- [22] Y. Efendiev, T. Y. Hou, V. Ginting, *et al.*, “Multiscale finite element methods for nonlinear problems and their applications,” *Communications in Mathematical Sciences*, vol. 2, no. 4, pp. 553–589, 2004.

- [23] J. Dolbow, N. Moës, and T. Belytschko, “An extended finite element method for modeling crack growth with frictional contact,” *Computer methods in applied Mechanics and engineering*, vol. 190, no. 51, pp. 6825–6846, 2001.
- [24] E. Alpaydin, *Introduction to machine learning*. MIT press, 2010.
- [25] P.-S. Koutsourelakis, “Stochastic upscaling in solid mechanics: An exercise in machine learning,” *Journal of Computational Physics*, vol. 226, no. 1, pp. 301–325, 2007.
- [26] P.-S. Koutsourelakis and E. Billionis, “Scalable bayesian reduced-order models for simulating high-dimensional multiscale dynamical systems,” *Multiscale Modeling & Simulation*, vol. 9, no. 1, pp. 449–485, 2011.
- [27] A. Oishi and G. Yagawa, “Computational mechanics enhanced by deep learning,” *Computer Methods in Applied Mechanics and Engineering*, 2017.
- [28] R. Ibañez, D. Borzacchiello, J. V. Aguado, E. Abisset-Chavanne, E. Cueto, P. Ladeveze, and F. Chinesta, “Data-driven non-linear elasticity: Constitutive manifold construction and problem discretization,” *Computational Mechanics*, pp. 1–14, 2017.
- [29] R. Ibañez, E. Abisset-Chavanne, J. V. Aguado, D. Gonzalez, E. Cueto, and F. Chinesta, “A manifold learning approach to data-driven computational elasticity and inelasticity,” *Archives of Computational Methods in Engineering*, vol. 25, no. 1, pp. 47–57, 2018.
- [30] T. Kirchdoerfer and M. Ortiz, “Data-driven computational mechanics,” *Computer Methods in Applied Mechanics and Engineering*, vol. 304, pp. 81–101, 2016.
- [31] F. Casadei and M. Ruzzene, “Application of the multi-scale finite element method to wave propagation problems in damaged structures,” *Health Monitoring of Structural and Biological Systems*, vol. 7984, 2011.
- [32] F. Casadei, J. J. Rimoli, and M. Ruzzene, “A geometric multiscale finite element method for the dynamic analysis of heterogeneous solids,” *Comput. Methods Appl. Mech. Eng.*, vol. 263, pp. 56–70, 2012.
- [33] R. J. Guyan, “Reduction of stiffness and mass matrices,” *AIAA journal*, vol. 3, no. 2, p. 380, 1964.
- [34] R. R. Craig and A. J. Kurdila, *Fundamentals of Structural Dynamics*. John Wiley and Sons, 2006.

- [35] F. Casadei, J. Rimoli, and M. Ruzzene, “Multiscale finite element analysis of wave propagation in periodic solids,” *Finite Elements in Analysis and Design*, vol. 108, pp. 81–95, 2016.
- [36] R. J. C. Jr. and M. C. C. Bampton, “Coupling of substructures for dynamic analyses,” *AIAA journal*, vol. 6, no. 7, pp. 1313–1319, 1968.
- [37] K. J. Bathe, *Finite Element Procedures*. Prentice Hall, 1996.
- [38] R. D. Cook, D. S. Malkus, M. E. Plesha, and R. J. Witt, *Concepts and Applications of Finite Element Analysis*. John Wiley & Sons, 2007, ISBN: 0470088214.
- [39] L. P. Franca and A. P. Macedo, “A two-level finite element method and its application to the helmholtz equation,” *International Journal for Numerical Methods in Engineering*, vol. 43, no. 1, pp. 23–32, 1998.
- [40] Y. Efendiev, J. Galvis, and X. H. Wu, “Multiscale finite element methods for high-contrast problems using local spectral basis functions,” *Journal of Computational Physics*, vol. 230, no. 4, pp. 937–955, 2011.
- [41] R. R. Craig, *Structural Dynamics*. John Wiley and Sons, 1981.
- [42] F. Casadei and J. J. Rimoli, “Anisotropy-induced broadband stress wave steering in periodic lattices,” *International Journal of Solids and Structures*, vol. 50, no. 9, pp. 1402–1414, 2013.
- [43] P. Celli and S. Gonella, “Low-frequency spatial wave manipulation via phononic crystals with relaxed cell symmetry,” *Journal of Applied Physics*, vol. 115, no. 10, p. 103 502, 2014.
- [44] X. N. Liu, G. K. Hu, G. L. Huang, and C. T. Sun, “An elastic metamaterial with simultaneously negative mass density and bulk modulus,” *Applied physics letters*, vol. 98, no. 25, 251907, 2011.
- [45] M. Schaeffer and M. Ruzzene, “Wave propagation in multistable magneto-elastic lattices,” *International Journal of Solids and Structures*, vol. 56, pp. 78–95, 2015.
- [46] M. J. S. Lowe and O. Diligent, “Low-frequency reflection characteristics of the s₀ lamb wave from a rectangular notch in a plate,” *The Journal of the Acoustical Society of America*, vol. 111, pp. 64–74, 2001.
- [47] K. P. Murphy, “Machine learning: A probabilistic perspective,” 2012.
- [48] A. I. Forrester and A. J. Keane, “Recent advances in surrogate-based optimization,” *Progress in Aerospace Sciences*, vol. 45, no. 1-3, pp. 50–79, 2009.

- [49] C. K. Williams and C. E. Rasmussen, *Gaussian processes for machine learning*, 3. MIT Press Cambridge, MA, 2006, vol. 2.
- [50] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, “Automatic differentiation in machine learning: A survey,” *Journal of Machine Learning Research*, vol. 18, pp. 1–43, 2018.
- [51] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [52] A. Forrester, A. Keane, *et al.*, *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.
- [53] T. Belytschko, W. K. Liu, B. Moran, and K. Elkhodary, *Nonlinear finite elements for continua and structures*. John Wiley & Sons, 2013.
- [54] F. Casadei, J. Rimoli, and M. Ruzzene, “A geometric multiscale finite element method for the dynamic analysis of heterogeneous solids,” *Computer Methods in Applied Mechanics and Engineering*, vol. 263, pp. 56–70, 2013.
- [55] A. F. Bower, *Applied mechanics of solids*. CRC press, 2009.
- [56] A. Mota, W. Sun, J. T. Ostien, J. W. Foulk, and K. N. Long, “Lie-group interpolation and variational recovery for internal variables,” *Computational Mechanics*, vol. 52, no. 6, pp. 1281–1299, 2013.
- [57] J.-M. Battini, “A non-linear corotational 4-node plane element,” *Mechanics Research Communications*, vol. 35, no. 6, pp. 408–413, 2008.
- [58] C. Kelley, *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, 1995. eprint: <http://epubs.siam.org/doi/pdf/10.1137/1.9781611970944>.
- [59] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, 27:1–27:27, 3 2011, Software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [60] J. C. Simo and T. J. Hughes, *Computational inelasticity*. Springer Science & Business Media, 2006, vol. 7.
- [61] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.