Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 345 (2019) 363–381
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

Smart finite elements: A novel machine learning application

German Capuano, Julian J. Rimoli*

School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

Received 21 June 2018; received in revised form 9 October 2018; accepted 16 October 2018
Available online 19 November 2018

Highlights

- We use machine learning to find the forces corresponding to the element's state.
- The method avoids the complex task of finding the internal displacement field.
- We increase the performance of the method by enforcing known physical constraints.
- The method is not tied to any particular machine learning algorithm.
- The method does not impose any restriction on the solver of choice.

Abstract

Many multiscale finite element formulations can become computationally expensive because they rely on detailed models of the element's internal displacement field. This issue is exacerbated in the presence of nonlinear problems, where numerical iterations are generally needed. We propose a method that utilizes machine learning to generate a direct relationship between the element state and its forces, which avoids the complex task of finding the internal displacement field and eliminates the need for numerical iterations. 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.

Keywords: Machine learning; Finite elements; Multiscale models

1. Introduction

The finite element method is extremely popular because it solves boundary value problems in a straightforward and systematic way, even in cases involving complex geometries. Unfortunately, the method is computationally prohibitive under many circumstances, including but not limited to problems with discontinuities, singularities, and multiple relevant scales. Some alternative methods that tackle its deficiencies are the global-local methods [1], residual free bubbles [2,3], the variational multiscale method [4], the discontinuous enrichment method [5,6], methods based on

* Corresponding author.
E-mail address: rimoli@gatech.edu (J.J. Rimoli).

https://doi.org/10.1016/j.cma.2018.10.046
0045-7825/

partition of unity and the generalized finite element method [7–9], and multiscale methods [10–13]. These methods share a common trait: they expand or modify the solution space to better fit the particular problem. However, they differ in the type of assumptions they make about the possible solutions. In the most basic cases, a known handbook solution is used to enrich the representation basis. This is the typical scenario when using the extended finite element method for crack propagation analysis [14]. In some other cases, the methods are used in conjunction with assumptions over the distribution of features, such as periodicity or volume representation [15,16]. While these 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 [17].

Computational mechanics is not the exception. Machine learning has been used to formulate multiscale elements [18,19], to enhance the performance of traditional elements [20], to extract constitutive manifolds [21,22] and to produce a data-driven solver [23]. 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 3.1). 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 existing finite element formulations. That is, we use a machine learning algorithm on data extracted from individual elements 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 outline of the article is as follows. First, we review in Section 2 the typical machine learning algorithms and procedures applicable to our method. In Section 3 we define the formulation of our method and provide a simple example of a linear regression model. In Section 4 we provide two nonlinear case studies that make extensive use of these techniques. Finally, in Section 5 we summarize the main findings and provide concluding remarks, including possible directions for further research.

2. 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 $\mathcal{D} = \left\{x^i, r^i\right\}_{i=1}^{N_s}$, 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 dataset 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 [17,24] provide a good introduction to the topic, and the article in reference [25] 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 and support vector regression. Most implementations of these methods produce only single outputs ($r^i \in \mathbb{R}$) so we focus only on that scenario. To produce multiple outputs, which is necessary in our work, one can simply produce multiple models, one per output component.

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 ith row is the vector x^i, and R is a vector whose ith 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 $\mathcal{O}(p)$.

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 [17,24] 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. 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 Fig. 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

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

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 [26], 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 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 $T_{fe}(\cdot)$, 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 $\hat{f}_e^i = T_{fe}(f_e^i)$. 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 $\bar{f}_g = T_{fe}^{-1}(\hat{f}_e)$. Thus, we can write

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

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 [27] and the displacement field as the field of interest. However, the method is equally applicable to any other formulation and type of problem.

3.1. Example: 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 [11], 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.

From the nodes at the boundary of the fine-scale mesh, some are selected as the coarse scale nodes of the element (see Fig. 1). 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 ith row and 0

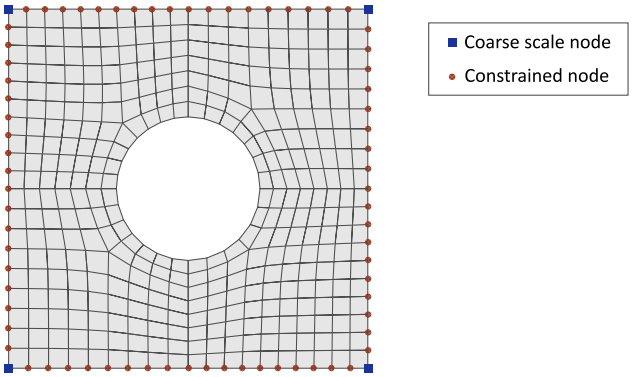

Fig. 1. Example of a geometric multiscale finite element with the fine-scale model represented as a mesh in its interior.

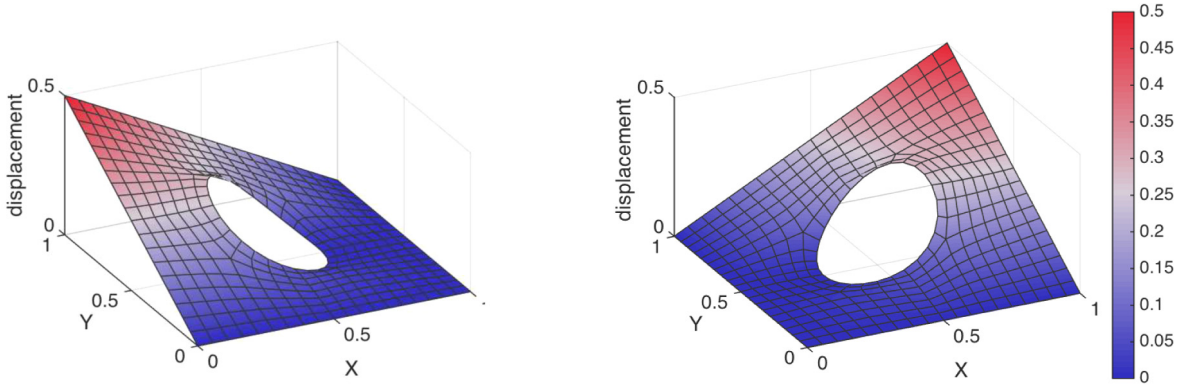

Fig. 2. Example of shape functions used in the geometric multiscale finite element method.

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 ith column of T. Two of such solutions are shown in Fig. 2. Once this procedure is performed for all degrees of freedom the stiffness matrix of the element K_e can be obtained using

$$K_e = T^T K_f T \tag{1}$$

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 $\mathcal{D}_e = \left\{\phi_e^i, f_e^i\right\}_{i=1}^{N_s}$ 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 (1).

3.2. 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 4.

3.2.1. 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 [28]. 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 [29], or we can derive it from geometric considerations [30]. 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 [30])

$$u_{ed}^I = R_e(u_e^I - u_{et} + X_e^I - X_{ec}) - (X_e^I - X_{ec}) \tag{2}$$

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}.

3.2.2. 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

$$R_e^i(f_e) = 0 \tag{3}$$

where R_e^i is some known function of the internal forces, $i = 1, \ldots, q_e$, 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). 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 4.1.

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 $i = 1, \ldots, q_e$. 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 4.2.

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), 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.

4. 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 [23]. 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.

4.1. 3D truss structure

In this case study, we follow a static 3D truss problem suggested by Kirchdoerfer et al. [23]. The structure is composed of cubic truss cells of size $1\,m$ and its configuration is given in Fig. 3 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 Fig. 4.

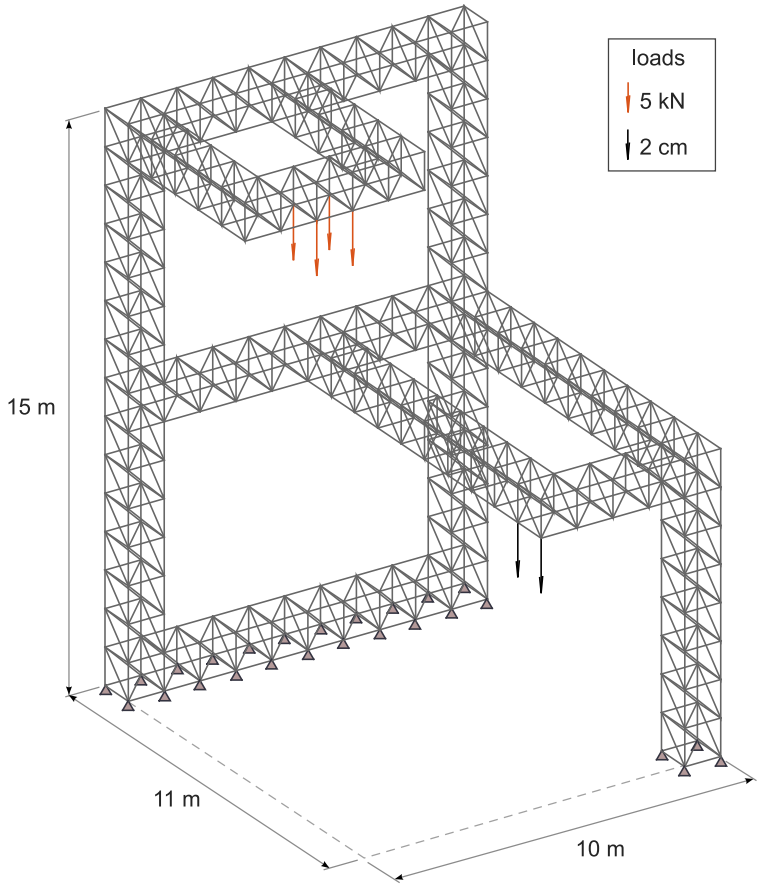

Fig. 3. Example truss structure with its applied loads and imposed displacements.

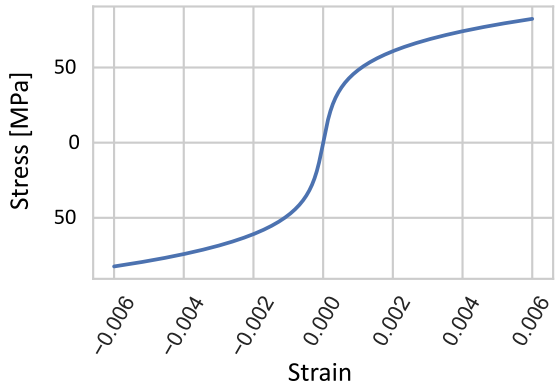

Fig. 4. Stress–strain relationship of the material in the truss structure. Adapted from Kirchdoerfer et al. [23].

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 [31].

The corotational formulation of the truss elements implicitly follows the ideas from Section 3.2 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 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 lengths of bar e, and $x_e^I \in \mathbb{R}^3$ and $X_e^I \in \mathbb{R}^3$ are the current and initial positions 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_{fe}^{-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

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

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 Fig. 5, 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.

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 3.2.1. We

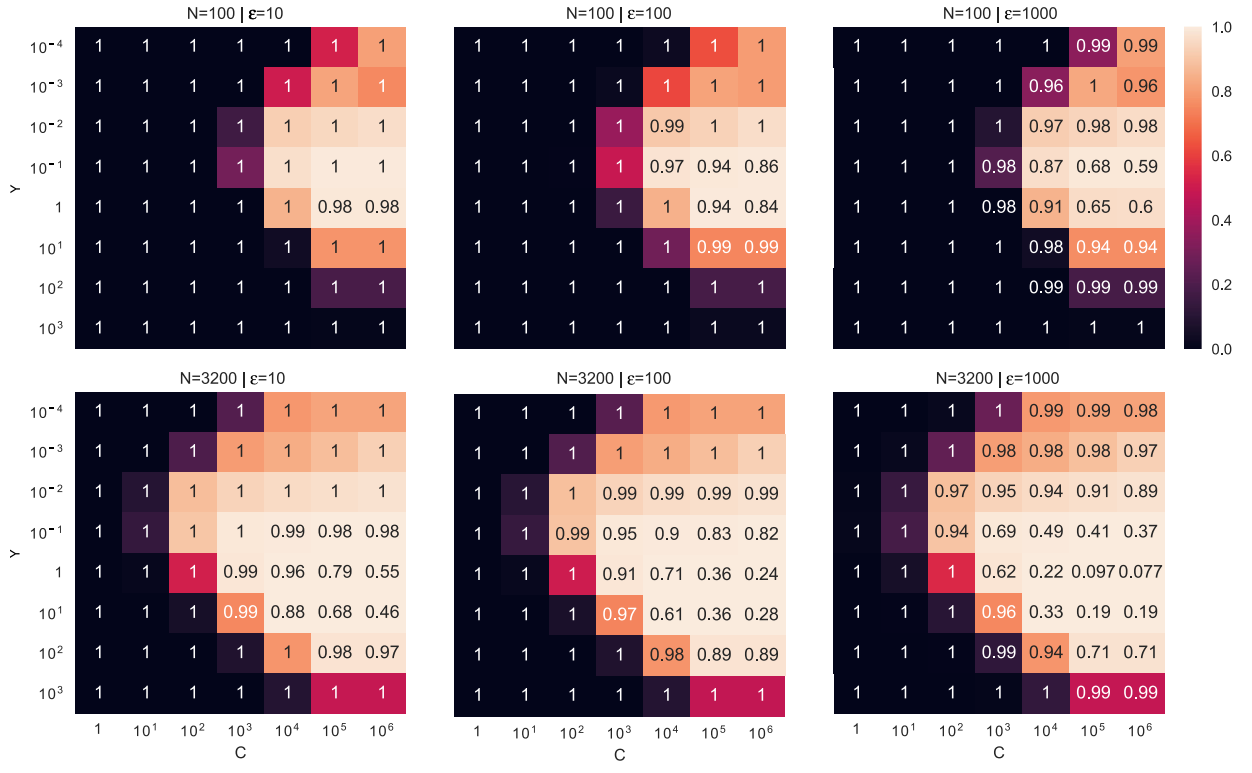

Fig. 5. Heatmaps for hyperparameter fine tuning. The colormap indicates the R^2 score for a test dataset of size 10000 and the value inside each box indicates the sparsity of the solution. The R^2 values have been limited to a minimum of 0 in the color visualization. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

show the error in the truss structure in Fig. 6 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

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

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.

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.

In Fig. 7 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. [23], 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 $C_e = 1e10$.

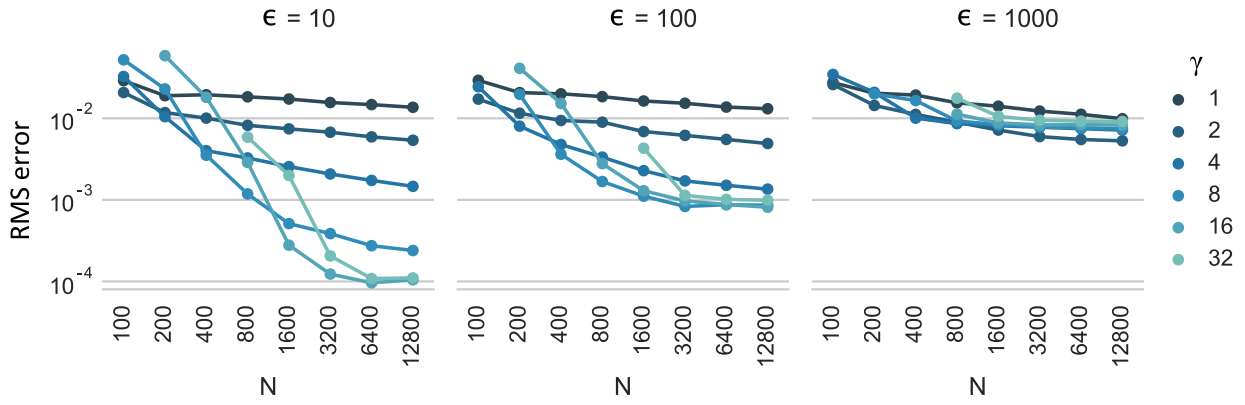

Fig. 6. RMS Error of the stress in the structure for different values of ϵ and γ depending on the size of the training sample N. Each point represents the average over the converged models obtained from 10 different training samples.

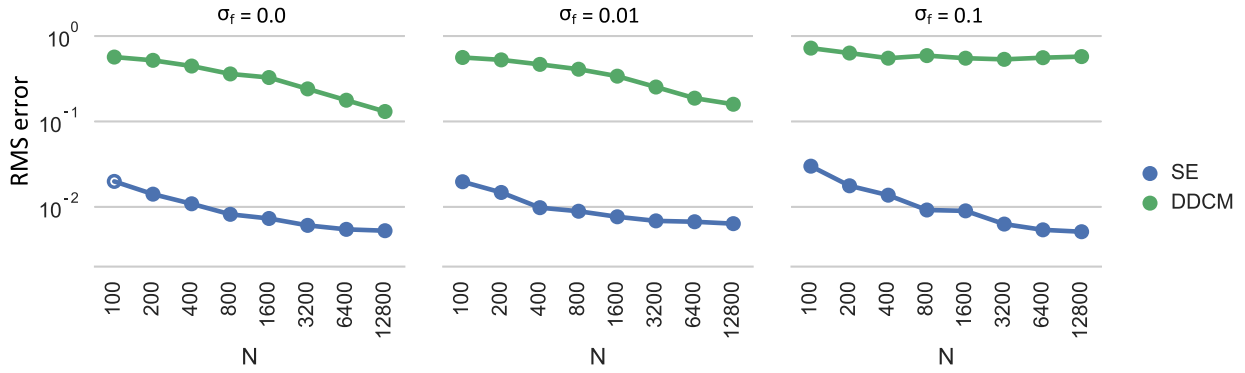

Fig. 7. Comparison of the RMS Error of the stress in the structure ϵ^{σ}_{RMS} for smart elements (SE) and data-driven computational mechanics (DDCM) as function of the training sample size N. The output in the training samples have a normally distributed error with a standard deviation σ_f. Each point in the plot represents the average error from 10 models with different training samples.

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 Fig. 6 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, Fig. 5 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. [21]. 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 [32] to achieve solutions that are acceptable for most engineering applications.

4.2. Nonlinear Multiscale problem

In this case study, we focus on the multiscale finite element method for nonlinear problems formulated by Efendiev et al. [12]. This method is similar to the geometric multiscale finite element method introduced in Section 3.1 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

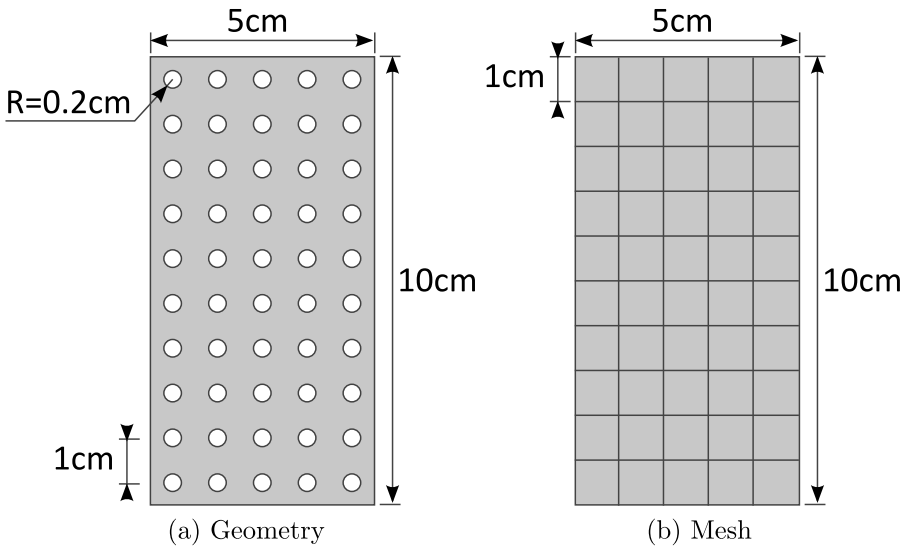

Fig. 8. A short bar used in Section 4.2. It contains a uniform array of 5 by 10 holes of diameter 0.4 cm. The mesh corresponding to this geometry does not contain the holes since they are considered part of the fine scale.

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 Fig. 8. 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 [28]

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

where $\bar{I}_1 = I_1/J^{2/3}$, I_1 is the first invariant of the right Cauchy–Green deformation tensor 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 $\mu = 0.9091$ MPa and $\kappa = 0.8333$ MPa, which correspond to $E = 2$ 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 $t_0 = 0\,s$ to the end time $t_{max} = 0.5\,s$.

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 Section 3.1, which is shown in Fig. 1. 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 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 had 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

$$P(\tau) = P_{max}\tau^3 * (10 - 15\tau + 6\tau^2) \tag{4}$$

where $P(\tau)$ is the magnitude of the load as a function of a non-dimensional time $\tau = t/t_{load}s$, 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 (2) 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^I_{ed} for the former, and the displacements u^I_e 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

$$f_i = f_i^L(\lambda_e) * (1 - P(\tau)) + f_i^{SVR}(\lambda_e) * P(\tau) \tag{5}$$

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 Eq. (4) 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

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

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 Fig. 9, and the results comparing the error on the different smart elements in Fig. 10. 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.

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

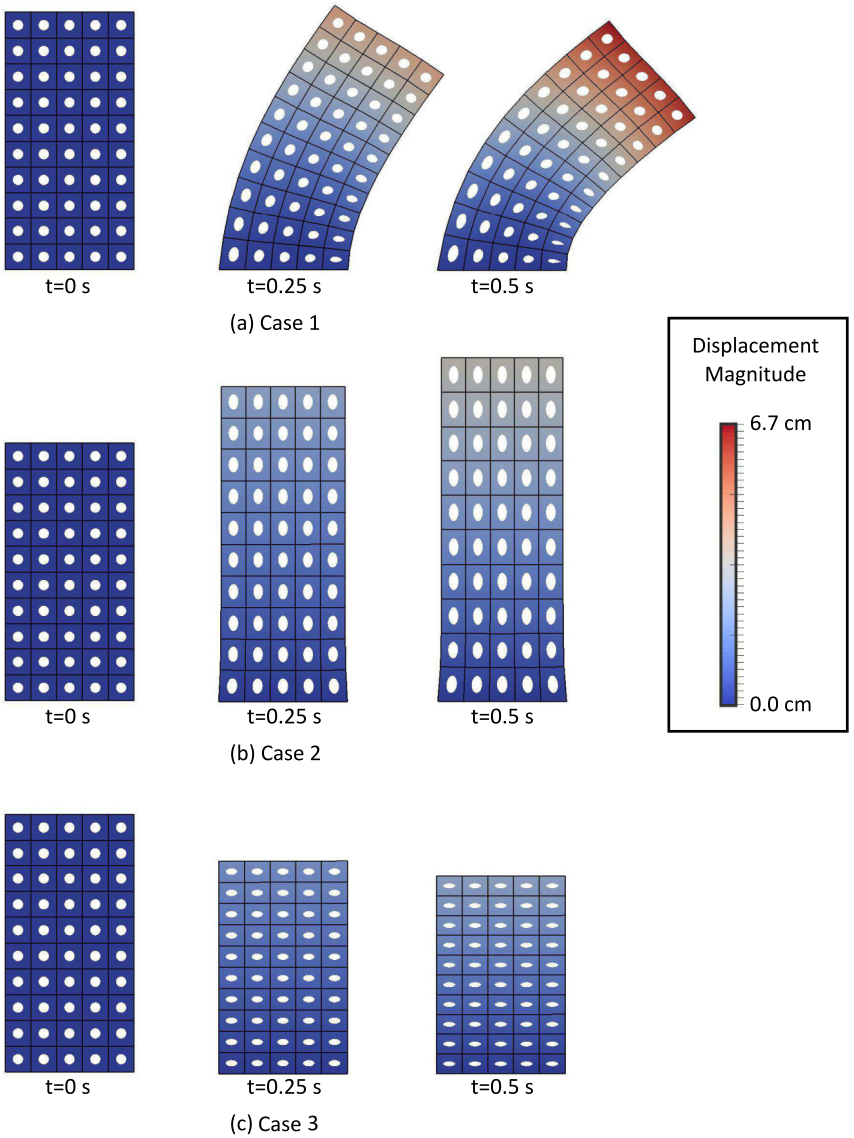

Fig. 9. Comparison of the deformed configuration of a multiscale finite element model and a smart model for the three load cases in three different time instants. The first is displayed as a colored surface while the latter is displayed as a black wireframe. The shown smart model corresponds to ordinary polynomial regression of order 6 (Ord 6). Note that in the figure the exact MsFE solutions and the smart element solutions are superimposed. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

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

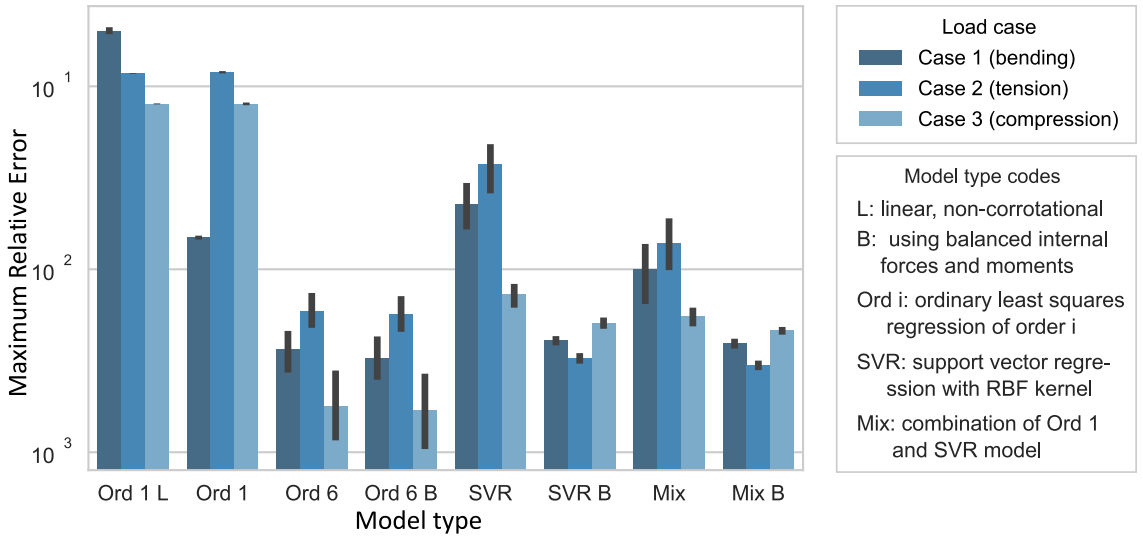

Fig. 10. Maximum Relative Error ϵ^{u}_{max} in the displacements for different smart element types and load scenarios.

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.

4.2.1. 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 $N_s = 20000$.

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 Fig. 11, 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.

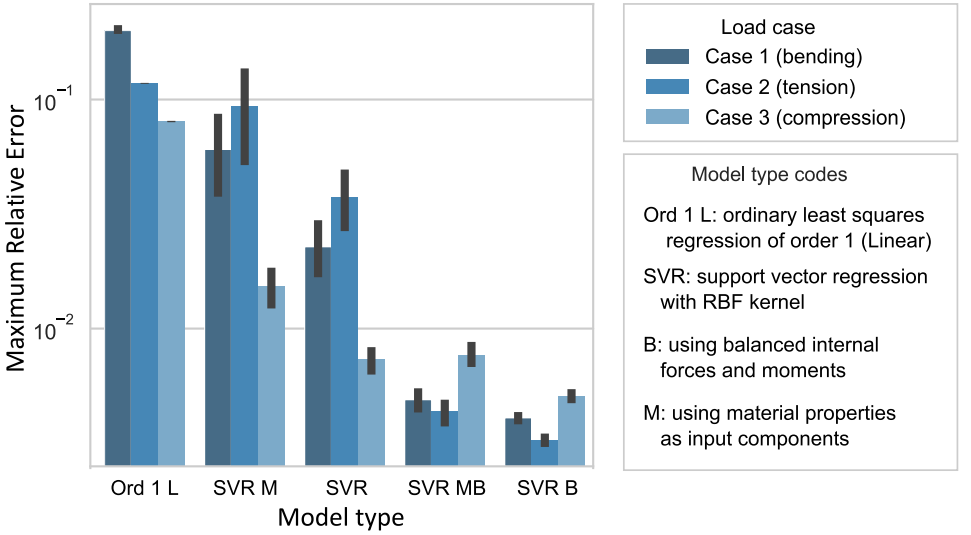

Fig. 11. Comparison of the Maximum Relative Error ϵ^{u}_{max} in the displacements of the smart elements when including or excluding consideration of the material properties in the input vector.

5. Summary and concluding remarks

We proposed 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 approximation model of the element. This process is known as surrogate modeling, and it allows 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. Then 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 combine 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 4.1. 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.

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), i.e., 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] Ahmed K. Noor, Global-local methodologies and their application to nonlinear analysis, Finite Elem. Anal. Des. 2 (4) (1986) 333–346.
[2] Franco Brezzi, Alessandro Russo, Choosing bubbles for advection-diffusion problems, Math. Models Methods Appl. Sci. 4 (04) (1994) 571–587.
[3] Leopoldo P. Franca, Charbel Farhat, Antonini P. Macedo, Michel Lesoinne, Residual-free bubbles for the Helmholtz equation, Internat. J. Numer. Methods Engrg. 40 (21) (1997) 4003–4009.
[4] Thomas J.R. Hughes, Gonzalo R. Feijóo, Luca Mazzei, Jean-Baptiste Quincy, The variational multiscale method–a paradigm for computational mechanics, Comput. Methods Appl. Mech. Engrg. 166 (1–2) (1998) 3–24.
[5] Charbel Farhat, Isaac Harari, Leopoldo P. Franca, The discontinuous enrichment method, Comput. Methods Appl. Mech. Engrg. 190 (48) (2001) 6455–6479.
[6] Charbel Farhat, Isaac Harari, Ulrich Hetmaniuk, The discontinuous enrichment method for multiscale analysis, Comput. Methods Appl. Mech. Engrg. 192 (28) (2003) 3195–3209.
[7] Jens M. Melenk, Ivo Babuška, The partition of unity finite element method: basic theory and applications, Comput. Methods Appl. Mech. Engrg. 139 (1–4) (1996) 289–314.
[8] Theofanis Strouboulis, Ivo Babuška, Kevin Copps, The design and analysis of the generalized finite element method, Comput. Methods Appl. Mech. Engrg. 181 (1) (2000) 43–69.
[9] Thomas-Peter Fries, Ted Belytschko, The extended/generalized finite element method: an overview of the method and its applications, Internat. J. Numer. Methods Engrg. 84 (3) (2010) 253–304.
[10] Thomas Y. Hou, Xiao-Hui Wu, A multiscale finite element method for elliptic problems in composite materials and porous media, J. Comput. Phys. 134 (1) (1997) 169–189.
[11] F. Casadei, J.J. Rimoli, M. Ruzzene, A geometric multiscale finite element method for the dynamic analysis of heterogeneous solids, Comput. Methods Appl. Mech. Engrg. 263 (2013) 56–70.
[12] Yalchin Efendiev, Thomas Y. Hou, Victor Ginting, et al., Multiscale finite element methods for nonlinear problems and their applications, Commun. Math. Sci. 2 (4) (2004) 553–589.
[13] German Capuano, Massimo Ruzzene, Julian J. Rimoli, Modal-based finite elements for efficient wave propagation analysis, Finite Elem. Anal. Des. 145 (2018) 10–19.
[14] John Dolbow, Nicolas Moës, Ted Belytschko, An extended finite element method for modeling crack growth with frictional contact, Comput. Methods Appl. Mech. Engrg. 190 (51) (2001) 6825–6846.
[15] H.M. Inglis, P.H. Geubelle, K. Matouš, H. Tan, Y Huang, Cohesive modeling of dewetting in particulate composites: micromechanics vs. multiscale finite element analysis, Mech. Mater. 39 (6) (2007) 580–595.
[16] Karel Matouš, Philippe H. Geubelle, Multiscale modelling of particle debonding in reinforced elastomers subjected to finite deformations, Internat. J. Numer. Methods Engrg. 65 (2) (2006) 190–223.
[17] Ethem Alpaydin, Introduction to Machine Learning, MIT press, 2010.
[18] Phaedon-Stelios Koutsourelakis, Stochastic upscaling in solid mechanics: an excercise in machine learning, J. Comput. Phys. 226 (1) (2007) 301–325.
[19] Phaedon-Stelios Koutsourelakis, Elias Bilionis, Scalable Bayesian reduced-order models for simulating high-dimensional multiscale dynamical systems, Multiscale Model. Simul. 9 (1) (2011) 449–485.
[20] Atsuya Oishi, Genki Yagawa, Computational mechanics enhanced by deep learning, Comput. Methods Appl. Mech. Engrg. (2017).
[21] Ruben Ibañez, Domenico Borzacchiello, Jose Vicente Aguado, Emmanuelle Abisset-Chavanne, Elias Cueto, Pierre Ladeveze, Francisco Chinesta, Data-driven non-linear elasticity: constitutive manifold construction and problem discretization, Comput. Mech. (2017) 1–14.
[22] Rubén Ibañez, Emmanuelle Abisset-Chavanne, Jose Vicente Aguado, David Gonzalez, Elias Cueto, Francisco Chinesta, A manifold learning approach to data-driven computational elasticity and inelasticity, Arch. Comput. Methods Eng. 25 (1) (2018) 47–57.
[23] Trenton Kirchdoerfer, Michael Ortiz, Data-driven computational mechanics, Comput. Methods Appl. Mech. Engrg. 304 (2016) 81–101.

[24] Kevin P. Murphy, Machine Learning: A Probabilistic Perspective, The MIT Press, 2012.
[25] Alexander I.J. Forrester, Andy J. Keane, Recent advances in surrogate-based optimization, Prog. Aerosp. Sci. 45 (1–3) (2009) 50–79.
[26] Alexander I.J. Forrester, András Sóbester, Andy J. Keane, Engineering Design Via Surrogate Modelling: A Practical Guide, John Wiley & Sons, 2008.
[27] Ted Belytschko, Wing Kam Liu, Brian Moran, Khalil Elkhodary, Nonlinear Finite Elements for Continua and Structures, John wiley & sons, 2013.
[28] Allan F. Bower, Applied Mechanics of Solids, CRC press, 2009.
[29] Alejandro Mota, WaiChing Sun, Jakob T. Ostien, James W. Foulk, Kevin N. Long, Lie-group interpolation and variational recovery for internal variables, Comput. Mech. 52 (6) (2013) 1281–1299.
[30] Jean-Marc Battini, A non-linear corotational 4-node plane element, Mech. Res. Commun. 35 (6) (2008) 408–413.
[31] C. Kelley, Iterative Methods for Linear and Nonlinear Equations, Society for Industrial and Applied Mathematics, 1995.
[32] Chih-Chung Chang, Chih-Jen Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.