Computer Methods in Applied Mechanics and Engineering 418 (2024) 116552

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

CECM: A continuous empirical cubature method with application to the dimensional hyperreduction of parameterized finite element models

J.A. Hernández [a,c,*], J.R. Bravo [a,b], S. Ares de Parga [a,b]

[a] Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), Barcelona, Spain
[b] Universitat Politècnica de Catalunya-BarcelonaTech (UPC), Department of Civil and Environmental Engineering (DECA), Barcelona, Spain
[c] Universitat Politècnica de Catalunya- BarcelonaTech (UPC), E.S. d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa (ESEIAAT), Terrassa, Spain

ARTICLE INFO

Keywords:
Empirical Cubature Method
Hyperreduction
Reduced-order modeling
Singular Value Decomposition
Quadrature

ABSTRACT

We propose a method for finding optimal quadrature/cubature rules with positive weights for parameterized functions in 1D, 2D or 3D spatial domains. The method takes as starting point the values of the functions at the Gauss points of a finite element (FE) mesh of the spatial domain for a representative sample of input parameters, and then construct an elementwise continuous orthogonal basis for such functions using the truncated Singular Value Decomposition (SVD) along with element polynomial fitting. To avoid possible memory bottlenecks in computing the SVD, we propose a Sequential Randomized SVD (SRSVD) in which the matrix is provided in a column-partitioned format, and which uses randomization to accelerate the processing of each individual block. After computing the basis functions, the method determines an exact integration rule for such functions, featuring as many points as functions, and in which the points are selected among the Gauss points of the FE mesh. Finally, the desired optimal rule is obtained by an sparsification process in which the algorithm zeroes one weight at a time while readjusting the positions and weights of the remaining points so that the constraints of the problem are satisfied. We apply this methodology to multivariate polynomials in cartesian domains to demonstrate that the method is indeed able to produce optimal rules – i.e., Gauss product rules –, and to a 2-parameters, 3D sinusoidal–exponential function to illustrate the use of the SRSVD in scenarios in which the standard SVD cannot handle the operation because of memory limitations. Lastly, the fact that the method does not require the analytical expression of the integrand functions – just their values at the FE Gauss points – makes it suitable for dealing with the so-called hyperreduction of parameterized finite element models. We exemplify this by showing its performance in the derivation of low-dimensional surrogate models in the context of the multiscale FE method. The Matlab source codes of both the CECM and the SRSVD, along with the scripts for launching the numerical tests, are openly accessible in the public repository https://github.com/Rbravo555/CECM-continuous-empirical-cubature-method.

* Corresponding author at: Centre Internacional de Mètodes Numèrics en Enginyeria (CIMNE), Barcelona, Spain.
E-mail address: joaquin.alberto.hernandez@upc.edu (J.A. Hernández).

https://doi.org/10.1016/j.cma.2023.116552
Received 11 August 2023; Received in revised form 12 October 2023; Accepted 16 October 2023
Available online 27 October 2023

1. Introduction

The present paper is concerned with a classical problem of numerical analysis: the approximation of integrals over 1D, 2D and 3D domains of parameterized functions as a weighted sum of the values of such functions at a set of m points $\{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_m\}$:

$$\int_{\Omega} f(\boldsymbol{x};\mu)\, d\Omega \approx \sum_{g=1}^{m} f(\boldsymbol{x}_g;\mu) w_g, \tag{1}$$

(with m as small as possible). This problem is generally known as either quadrature (for 1D domains) or *cubature* (for higher dimensions), and has a long pedigree stretching back as far as C.F. Gauss, who devised in 1814 the eponymous quadrature rule for univariate polynomials.

1.1. Cubature problem in hyperreduced-order models

The recent development of the so-called *hyperreduced-order models* (HROMs) for parameterized finite element (FE) analyses [1,2] has sparked the resurgence of interest in this classical problem. Indeed, a crucial step in the construction of such HROMs is the solution of the cubature problem associated to the evaluation of the nonlinear term(s) in the pertinent governing equations. For instance, in a Galerkin-based structural HROM, the nonlinear term is typically the projection of the nodal FE internal forces $\boldsymbol{F}^h \in \mathbb{R}^{N_{dof}}$ (here N_{dof} denotes the number of degrees of freedom of the FE model) onto the span of the displacement modes, i.e.: $\boldsymbol{F} = \boldsymbol{\phi}^T \boldsymbol{F}^h$, $\boldsymbol{\phi} \in \mathbb{R}^{N_{dof} \times n}$ being the matrix of displacement modes. The basic premise in these HROMs is that the number of modes is much smaller that the number of FE degrees of freedom ($n \ll N_{dof}$). This in turn implies that the internal forces per unit volume will also reside in a space of relatively small dimensions (independent of the size of the underlying FE mesh), and therefore, its integral over the spatial domain will be, in principle, amenable to approximation by an efficient cubature rule, featuring far less points than the original FE-based rule. The challenge lies in determining the minimum number of cubature points necessary for achieving a prescribed accuracy, as well as their location and associated *positive* weights. The requisite of positive weights arises from the fact that, in a Galerkin FE framework, the Jacobian matrix of the discrete system of equations is a weighted sum of the contribution at each FE Gauss point. Thus, if the Jacobian matrices at point level are positive definite, the global matrix is only guaranteed to inherit this desirable attribute if the cubature weights are positive [2].

Before delving into the description of the diverse approaches proposed to date to deal with this cubature problem in the context of HROMs, it proves convenient to formally formulate the problem in terms of a generic parameterized vector-valued function $\boldsymbol{a} : \Omega \times D \to \mathbb{R}^n$. Let $\Omega = \cup_{e=1}^{N_{el}} \Omega^e$ be a finite element partition of the spatial domain $\Omega \subset \mathbb{R}^d$ ($d = 1, 2$ or 3). For simplicity of exposition, assume that all elements are isoparametric and of the same order of interpolation, possessing r Gauss points each. Suppose we are given the values of the integrand functions for P instantiations of the input parameters ($\{\boldsymbol{\mu}_i\}_{i=1}^{P} = D^s \subset D$) at all the Gauss points of the discretization. The integral of the function over Ω for each $\boldsymbol{\mu}_j$ ($j = 1, 2 \dots P$) can be calculated by the corresponding element Gauss rule as

$$b_k = \sum_{e=1}^{N_{el}} \int_{\Omega^e} a_i(\boldsymbol{x}, \boldsymbol{\mu}_j)\, d\Omega = \sum_{e=1}^{N_{el}} \sum_{g=1}^{r} a_i(\bar{\boldsymbol{x}}_g^e, \boldsymbol{\mu}_j) W_g^e, \qquad k = (j-1)n + i; \quad j = 1, 2 \dots P; \quad i = 1, 2 \dots n. \tag{2}$$

Here, $\bar{\boldsymbol{x}}_g^e \in \Omega^e$ denotes the position of the gth Gauss point of element Ω^e, whereas $W_g^e > 0$ is the product of the Gauss weight and the Jacobian of the isoparametric transformation at such a point. Each b_k ($k = 1, 2 \dots Pn$) is therefore considered as the "exact" integral, that is, the reference value we wish to approximate. The above expression can be written in a compact matrix form as

$$\boldsymbol{b}_{FE} = \boldsymbol{A}_{FE}^T \boldsymbol{W}_{FE}, \tag{3}$$

where $\boldsymbol{b}_{FE} \in \mathbb{R}^{nP}$ is the vector of "exact" integrals defined in Eq. (2), $\boldsymbol{A}_{FE}$ is the matrix obtained from evaluating the integrand functions at all the FE Gauss point, $\boldsymbol{X}_{FE} = \{\{\bar{\boldsymbol{x}}_g^e\}_{g=1}^{r}\}_{e=1}^{N_{el}}$, while $\boldsymbol{W}_{FE}$ designates the vector of FE weights, formed by gathering all the Gauss weights in a single column vector. Each column of $\boldsymbol{A}_{FE}$ is the discrete representation of a scalar-valued integrand function, and thus the total number of columns is equal to the number of sampling parameters P times the number of integrand functions per parameter, n. The number of rows of $\boldsymbol{A}_{FE}$, on the other hand, is equal to the total number of integration points ($M = N_{el} \cdot r$). In terms of element contributions, matrix $\boldsymbol{A}_{FE}$ is expressible as

$$\boldsymbol{A}_{FE} = \begin{bmatrix} \boldsymbol{A}_{FE}^{(1)} \\ \boldsymbol{A}_{FE}^{(2)} \\ \vdots \\ \boldsymbol{A}_{FE}^{(N_{el})} \end{bmatrix}_{N_{el} \cdot r \times nP} \quad \text{where} \quad \boldsymbol{A}_{FE}^{(e)} = \begin{bmatrix} a_1(\bar{\boldsymbol{x}}_1^e, \boldsymbol{\mu}_1) & a_2(\bar{\boldsymbol{x}}_1^e, \boldsymbol{\mu}_1) & \cdots & a_n(\bar{\boldsymbol{x}}_1^e, \boldsymbol{\mu}_1) & \cdots & a_n(\bar{\boldsymbol{x}}_1^e, \boldsymbol{\mu}_P) \\ a_1(\bar{\boldsymbol{x}}_2^e, \boldsymbol{\mu}_1) & a_2(\bar{\boldsymbol{x}}_2^e, \boldsymbol{\mu}_1) & \cdots & a_n(\bar{\boldsymbol{x}}_2^e, \boldsymbol{\mu}_1) & \cdots & a_n(\bar{\boldsymbol{x}}_2^e, \boldsymbol{\mu}_P) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_1(\bar{\boldsymbol{x}}_r^e, \boldsymbol{\mu}_1) & a_2(\bar{\boldsymbol{x}}_r^e, \boldsymbol{\mu}_1) & \cdots & a_n(\bar{\boldsymbol{x}}_r^e, \boldsymbol{\mu}_1) & \cdots & a_n(\bar{\boldsymbol{x}}_r^e, \boldsymbol{\mu}_P) \end{bmatrix}_{r \times nP} \tag{4}$$

(here $\boldsymbol{A}_{FE}^{(e)} \in \mathbb{R}^{r\times nP}$ denotes the block matrix corresponding to the r Gauss points of element Ω^e). The same notational scheme is used for the vector of FE weights:

$$\boldsymbol{W}_{FE} = \begin{bmatrix} \boldsymbol{W}_{FE}^{(1)} \\ \boldsymbol{W}_{FE}^{(2)} \\ \vdots \\ \boldsymbol{W}_{FE}^{(N_{el})} \end{bmatrix}_{N_{el}\cdot r\times 1} \quad \text{where} \quad \boldsymbol{W}_{FE}^{(e)} = \begin{bmatrix} W_1^e \\ W_2^e \\ \vdots \\ W_r^e \end{bmatrix}_{r\times 1} . \tag{5}$$

The cubature problem consists in finding a set of points $\boldsymbol{X} := \{\boldsymbol{x}_g\}_{g=1}^m$ ($\boldsymbol{x}_g \in \Omega$) and associated *positive* weights $\{\omega_g\}_{g=1}^m$ (with m *as small as possible*) such that the vector of "exact" integrals $\boldsymbol{b}_{FE}$ is approximated to some desired level of accuracy $0 \le \epsilon_b \le 1$, i.e.:

$$\|\boldsymbol{A}^T(\boldsymbol{X})\boldsymbol{\omega} - \boldsymbol{b}_{FE}\|_2 \le \epsilon_b \|\boldsymbol{b}_{FE}\|_2. \tag{6}$$

Here, $\|\bullet\|_2$ is the standard Euclidean norm, whereas $\boldsymbol{A}(\boldsymbol{X})$ and $\boldsymbol{\omega}$ denote the matrix of the integrand evaluated at the set of points $\boldsymbol{X}$ and their associated weights, respectively:

$$\boldsymbol{A}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{A}(\boldsymbol{x}_1) \\ \boldsymbol{A}(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{A}(\boldsymbol{x}_m) \end{bmatrix} := \begin{bmatrix} a_1(\boldsymbol{x}_1,\boldsymbol{\mu}_1) & a_2(\boldsymbol{x}_1,\boldsymbol{\mu}_1) & \cdots & a_n(\boldsymbol{x}_1,\boldsymbol{\mu}_1) & \cdots & a_n(\boldsymbol{x}_1,\boldsymbol{\mu}_P) \\ a_1(\boldsymbol{x}_2,\boldsymbol{\mu}_1) & a_2(\boldsymbol{x}_2,\boldsymbol{\mu}_1) & \cdots & a_n(\boldsymbol{x}_2,\boldsymbol{\mu}_1) & \cdots & a_n(\boldsymbol{x}_2,\boldsymbol{\mu}_P) \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_1(\boldsymbol{x}_m,\boldsymbol{\mu}_1) & a_2(\boldsymbol{x}_m,\boldsymbol{\mu}_1) & \cdots & a_n(\boldsymbol{x}_m,\boldsymbol{\mu}_1) & \cdots & a_n(\boldsymbol{x}_m,\boldsymbol{\mu}_P) \end{bmatrix}_{m\times Pn} \quad \boldsymbol{\omega} = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_m \end{bmatrix}_{m\times 1} . \tag{7}$$

Remark 1.1. A remark concerning notation is in order here. In Eq. (7), $\boldsymbol{A}(\boldsymbol{x})$ ($\boldsymbol{x} \in \Omega$) represents a vector-valued function that returns the n entries of the integrand function $\boldsymbol{a}(\boldsymbol{x})$ for the P samples of the input parameters, in the form of row matrix (i.e., $\boldsymbol{A}(\boldsymbol{x}) \in \mathbb{R}^{1\times nP}$). On the other hand, when the argument of $\boldsymbol{A}$ is not a single point, but a collection of m points $\boldsymbol{X} := \{\boldsymbol{x}_g\}_{g=1}^m$, then $\boldsymbol{A}(\boldsymbol{X})$ represents a matrix with as many rows as points in the set, i.e. $\boldsymbol{A}(\boldsymbol{X}) \in \mathbb{R}^{m\times nP}$. According to this notational convention, the matrix defined in Eq. (4) can be compactly written as $\boldsymbol{A}_{FE} = \boldsymbol{A}(\boldsymbol{X}_{FE})$.

1.2. State-of-the-art on cubature rules for HROMs

The first attempts to solve the above described cubature problem in the context of reduced-order modeling were carried by the computer graphics community. The cubature scheme proposed by An et al. in Ref. [3] in 2010 for dealing with the evaluation of the internal forces in geometrically nonlinear models may be regarded as the germinal paper in this respect. An and co-workers [3] addressed the cubature problem (6) as a *best subset selection problem* (i.e., the desired set of points is considered a subset of the entire set of Gauss points, $\boldsymbol{X} \subset \boldsymbol{X}_{FE}$). They proposed a greedy strategy that incrementally constructs the set of points by minimizing the norm of the residual of the integration at each iteration, while enforcing the positiveness of the weights. Subsequent papers in the computer graphics community (see Ref. [4] and references therein) revolved around the same idea, and focused fundamentally in improving the efficiency of the scheme originally proposed by An et al. [3]—which turned out to be ostensibly inefficient, for it solves a nonnegative-least squares problem, using the standard Lawson–Hanson algorithm [5], each time a new point enters the set.

Interesting re-interpretations of the cubature problem came with the works of Von Tycowicz et al. [6] and Pan et al. [7]—still within computer graphics circles. Both works recognized the analogy between the discrete cubature problem and the quest for *sparsest solution of underdetermined systems of equations*, a problem which is common to many disciplines such as signal processing, inverse problems, and genomic data analysis [8]. Indeed, if we regard the vector of reduced weights $\boldsymbol{\omega}$ as a sparse vector of the same length as $\boldsymbol{W}_{FE}$, then the *best subset selection* problem can be posed as that of minimizing the nonzero entries of $\boldsymbol{\omega}$:

$$\min_{\boldsymbol{\omega}\ge\boldsymbol{0}} \|\boldsymbol{\omega}\|_0, \quad \text{subject to } \|\boldsymbol{A}_{FE}^T\boldsymbol{\omega} - \boldsymbol{b}_{FE}\|_2 \le \epsilon_b\|\boldsymbol{b}_{FE}\|_2 \tag{8}$$

where $\|\cdot\|_0$ stands for the ℓ_0 pseudo-norm—the number of nonzero entries of the vector. It is well-known [9] that this problem is computationally intractable (NP hard), and therefore, recourse to either suboptimal greedy heuristic or convexification is to be made. Von Tycowicz et al. [6] adapted the algorithm proposed originally in Ref. [10] for compressed sensing applications (called *normalized iterative hard thresholding*, abbreviated NIHT) by incorporating the positive constraints, reporting significant improvements in performance with respect to the original NNLS-based algorithm of An et al. [3]. The work by Pan et al. [7], on the other hand, advocated an alternative approach – also borrowed from the compressed sensing literature, see Ref. [11] – based on the *convexification* of problem (8). Such a convexification consists in replacing the ℓ_0 pseudo-norm by the ℓ_1 norm—an idea that, in turn, goes back to the seminal paper by Chen et al. [12]. In doing so, the problem becomes tractable, and can be solved by standard Linear Programming (LP) techniques.

Cubature schemes did not enter the computational engineering scene until the appearance in 2014 of the *Energy-Conserving Mesh Sampling and Weighting* (ECSW) *scheme* proposed by C. Farhat and co-workers [1]. The ECSW is, in essence, a nonnegative least squares method (NNLS), very much aligned to the original proposal by An et al. [3], although much more algorithmically efficient. Indeed, Farhat and co-workers realized that the NNLS itself produces sparse approximations, and therefore it suffices to introduce a control-error parameter inside the standard NNLS algorithm—rather than invoking the NNLS at each greedy iteration, as proposed originally in An's paper [3]. The efficiency of the ECSW was tested against other sparsity recovery algorithms by Farhat's

team in Ref. [13], arriving at the conclusion that, if equipped with an updatable QR decomposition for calculating the unrestricted least-squares problem of each iteration, the ECSW outperformed existing implementations based on convexification of the original problem. It should be pointed out that, although the ECSW is a mesh sampling procedure, and therefore, the entities selected by the ECSW are finite elements rather than single Gauss points, the formulation of the problem is rather similar to the one described in the foregoing: the only differences are that, firstly, each element contribution $\boldsymbol{A}_{FE}^{(e)}$ in Eq. (4) collapses into a single row obtained as the weighted sum of the Gauss points rows; and, secondly, the vector of FE weights $\boldsymbol{W}_{FE}$ is replaced by an all-ones vector.

The Empirical Cubature Method, hereafter referred to as Discrete Empirical Cubature Method (DECM), introduced by the first author in Ref. [2] for parameterized finite element structural problems, also addresses the problem via a greedy algorithm, in the spirit of An's approach [3], but exploits the fact that deriving a cubature rule for integrating the set of functions contained column-wise in matrix $\boldsymbol{A}_{FE}$ is equivalent to deriving a cubature rule for a set of *orthogonal bases* for such functions. Ref. [2] demonstrates that this brings two salient advantages in the points selection process. Firstly, the algorithm invariably converges to zero integration error when the number of selected points is equal to the number of orthogonal basis functions; and secondly, the algorithm need not enforce the positiveness of the weights at each iteration. Furthermore, Ref. [2] recognizes that the cubature problem is ill-posed when $\boldsymbol{b}_{FE} \approx \boldsymbol{0}$ – this occurs, for instance, in self-equilibrated structural problems, such as computational homogenization [14,15] – and shows that this can be overcome by enforcing the sum of the reduced weights to be equal to the volume of the domain. In Ref. [16], the first author proposed an improved version of the original DECM, in which the local least-squares problem at each iteration are solved by rank-one updates.

Another approach also introduced recently in the computational engineering community is the *Empirical Quadrature Method* (EQM), developed by A. Patera and co-workers [17–19]. It should be noted that the name similarity with the above described Empirical Cubature Method is only coincidental, for the EQM is not based on the nonnegative least squares method, like the ECM, but rather draws on the previously mentioned ℓ_1 convexification of problem (8). Thus, in the EQM, the integration rule is determined by linear programming techniques, as in the method advocated in the work by Pan et al. [7] for computer graphics applications.

Remark 1.2. The preceding literature survey solely includes hyperreduction methods based on the efficient evaluation of the integrals appearing in the full-order FE equations via cubature rules with positive weights. Nevertheless, it should be mentioned that there is another important class of hyperreduction methods, which actually predates cubature-based schemes in computational engineering circles. Such methods may be termed interpolation-based approaches, for they rely on the approximation of the full-order nonlinear terms, either assembled, see Refs. [20,21], or unassembled, see Ref. [22] as a linear expansion of SVD basis vectors, the coefficients in this expansion being determined by interpolation at selected entries of the basis matrix (or more generally, by least-squares fitting [23]). For a comprehensive review of such interpolation-based methods, the reader is referred to Ref. [24]. It should be also pointed out that, according to the comparative performance study conducted by Farhat and co-workers in Ref. [25], cubature-based schemes with positive weights tend to be more robust than interpolation-based approaches, because they preserve the numerical stability properties of the original full-order model.

1.3. Efficiency of best subset selection algorithms

The *best subset selection* algorithms described in the foregoing vary in the way the corresponding optimization problem is formulated, and also in computational performance (depending on the nature and size of the problem under consideration), yet all of them have something in common: none of them is able to provide the optimal solution, not even when the optimal integration points are contained in the set of FE Gauss points. We have corroborated this claim by examining the number of points provided by all these methods when the integrand is a 1D polynomial in the interval $\Omega = [-1, 1]$. In Fig. 1, we show, for the case of polynomials of order $P = 5$, the location of the points and the associated weights provided by[1]: (1) the nonnegative least-squares method (NNLS); (2) the linear-programming based method (LP); (3) the Discrete Empirical Cubature Method (DECM); (4) and the normalized iterative hard thresholding (NIHT). We also show in each Figure the 3-points optimal Gauss rule, which in this case is $\omega_1^* = \omega_3^* = 5/9$, $\omega_2^* = 8/9$, and $x_1^* = -x_3^* = \sqrt{3/5}$ $x_2^* = 0$. The employed spatial discretization features $N_{el} = 1000$ elements, with one Gauss point per element (located at the midpoint), and it was arranged in such a way that 3 of the corresponding element midpoints coincide with the optimal Gauss points. It can be seen that, as asserted, none of the four schemes is able to arrive at the optimal quadrature rule. Rather, the four methods provide quadrature rules with $m = P + 1 = 6$ points, that is, with as many points as functions to be integrated; in the related literature, these rules are known as *interpolatory quadrature rules* [27]. Different experiments with different initial discretizations and/or polynomial orders led invariably to the same conclusion (i.e., all of them produce interpolatory rules).

1.4. Goal and methodology

Having described the capabilities and limitations of existing cubature methods in the context of HROMs, we now focus on the actual goal of the present paper, which is to enhance such methods so that they produce rules close to the optimal ones—or at least rules featuring far less points than integrand functions. Our proposal in this respect draws inspiration from the *elimination* algorithm advocated, apparently independently, by Bremer et al. [28] and Xiao et al. [29] in the context of the so-called *Generalized Gaussian*

[1] The NNLS and LP analyses can be carried out by calling standard libraries (here we have used the Matlab functions *lsqnonneg* and *linprog*, respectively), the ECM algorithm is given in Ref. [16], whereas for the NIHT we have used the codes given in Ref. [26].

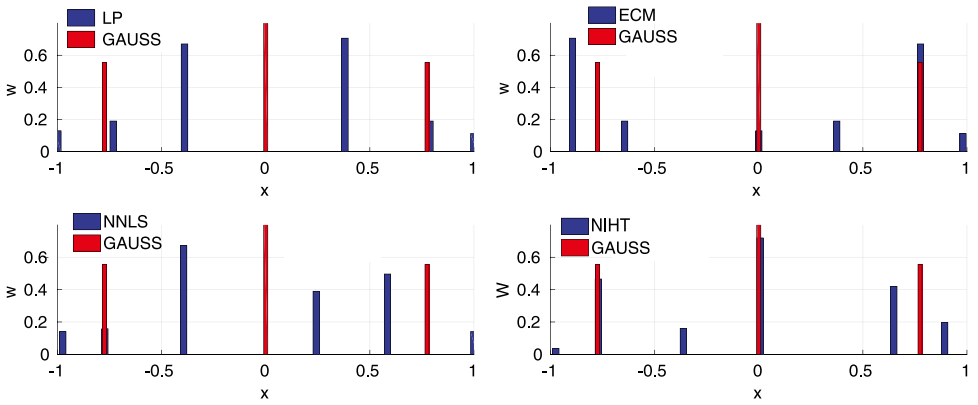


Fig. 1. Location of the points and magnitude of the weights of the integration rules for polynomial of order $P = 5$ in $\Omega = [-1, 1]$ provided by the: (a) Linear programming-based strategy (LP); (b) Discrete Empirical Cubature Method (DECM); (c) Non-negative least-squares (NNLS). (d) Normalized iterative hard thresholding (NIHT). Note that in the 4 cases, the number of integration points is equal to the number of functions to be integrated (i.e., monomials) $m = P+1 = 6$. The optimal solution is provided by the Gaussian quadrature rule of $m^* = 3$ points, also displayed in each of the four graphs.

Rules (see Refs. [30–32]), which, as its name indicates, is a research discipline that seeks to extend the scope of the quadrature rule originally developed for polynomials by C.F. Gauss. To the best of the authors' knowledge, cross-fertilization between this field and the field of hyperreduction of parameterized finite element models has not yet taken place. This lack of cross-fertilization may be attributed to the fact that the former is fundamentally concerned with parametric families of functions whose analytical expression is known, while the latter concentrates in huge databases of *empirical* functions (i.e., functions derived from *computational experiments*), whose values are only given at certain points of the spatial domain (the Gauss points of the FE mesh). The present work, thus, appears to be the first attempt to combine ideas from these two related disciplines.

The intuition behind the elimination algorithm presented in Refs. [28,29] goes as follows. Consider, for instance (the same arguments can be used with either the LP or NIHT approaches), the points and weights provided by the *interpolatory* DECM rule shown in Fig. 1.b. Observe that the distribution of weights is rather irregular, being the difference between the largest and smallest weights more pronounced than in the case of the optimal rule—for instance, the smallest weight is only 5% of the total length of the domain. This suggests that we may get rid of some of the points in the initial set, on the grounds that, as their contribution to the integration error is not significant (relatively small weights), a slight "readjustment" of the positions and weights of the remaining points may suffice to return the integration error to zero. Since we cannot know a priori how many points in total can be eliminated, this operation must be carried out carefully, removing one point at a time.

1.4.1. Sparsification problem

Although inspired by this elimination scheme, our approach addresses the problem from a different perspective, more in line with the *sparsification* formulation presented in expression (8), in which the goal is to drive to zero as many weights as possible. To understand how our sparsification scheme works, it proves useful to draw a physical analogy in which the integration points are regarded as *particles* endowed with *nonnegative masses* (the weights), and which are subject to nonlinear conservation equations (the integration conditions). At the beginning, the particles have the positions and masses (all positive) determined by one of the interpolatory cubature rules discussed previously in Section 1.3. The goal is to, progressively, drive to zero the mass of as many particles as possible, while keeping the remaining particles within the spatial domain, and with nonnegative masses. To this end, at each step, we reduce the mass of the particle that least contributes to the conserved quantities, and then calculate the position and masses of the remaining particles so that the nonlinear conservation equations are satisfied.

For solving the nonlinear balance equations using standard methods (i.e., Newton's), it is necessary to have a continuous (and differentiable) representation of the integrand functions. In contrast to the cases presented in Refs. [28,29], in our case, the analytical expression of such functions are in general not available. To overcome this obstacle, we propose to construct local polynomial interpolatory functions using the values of the integrand functions at the Gauss points of each finite element traversed by the particles.

Another crucial difference of our approach with respect to Refs. [28,29] is the procedure to solve the nonlinear equations at each step. Due to the underdetermination of such equations, there are an infinite number of possible configurations of the system for the majority of the steps. Both Refs. [28,29] use the pseudo-inverse of the Jacobian matrix, a fact that is equivalent to choosing the (non-sparse) ℓ_2 minimum-norm solution [9] in each iteration. By contrast, here we employ sparse solutions, with as many nonzero entries as functions to be integrated. The rationale for employing this sparse solution is that, on the one hand, it minimizes the number of particles that move at each iteration, and consequently, diminish the computational effort of tracking the particles through the mesh; and, on the other hand, it reduces the overall error inherent to the recovery of the integrand functions via interpolation.

It should be stressed that we do not employ a specific strategy for directly enforcing the positiveness of the masses (weights). Rather, we force the constant function to appear in the set of integrand functions; in our physical analogy, this implies that one

of the balance equations is the *conservation of mass*. Since the total mass of the system is to be conserved, reducing the mass of one particle leads to an increase in the overall mass of the remaining particles, and this tends to ensure that their masses remain positive. On the other hand, when a particle attempts to leave the domain, we return it back to its previous position, and proceed with the solution scheme. If convergence is not achieved, or the constraints are massively violated, we simply abandon our attempt of reducing the weight of the current controlled particle, and move to the next particle in the list. The process terminates – hopefully at the optimum – when we have tried to make zero the masses of all particles.

We choose as initial interpolatory rule – over the other methods discussed in Fig. 1 – the Discrete Empirical Cubature Method, DECM. The reason for this choice is twofold. Firstly, we have empirically found that the DECM gives points that tend to be close to the optimal ones—for instance, in Fig. 1.b two of the points calculated by the DECM practically coincide with the optimal Gauss points $x_1 = \sqrt{3/5}$ and $x_2 = 0$. Secondly, the DECM does not operate directly on the sampling matrix $\boldsymbol{A}_{FE}$ defined in Eq. (4), but rather on an orthogonal basis matrix for its column space [16]. As a consequence, the cubature problem translates into one of integrating orthogonal basis functions, and this property greatly facilitates the convergence of the nonlinear problem alluded to earlier. The combination of the DECM followed by the continuous search process will be referred to hereafter as the *Continuous Empirical Cubature Method* (CECM).

1.5. Sequential randomized SVD (SRSVD)

We use the ubiquitous Singular Value Decomposition (SVD) to determine the orthogonal basis matrix for the column space of $\boldsymbol{A}_{FE}$ required by the DECM. Computationally speaking, the SVD is by far the most memory-intensive operation of the entire cubature algorithm. For instance, in a parametric function of dimension $n = 10$, with $P = 100$ parametric samples and a mesh of $N_{el} = 10^6$ linear hexahedra elements featuring $n_g = 8$ Gauss points each, matrix $\boldsymbol{A}_{FE}$ occupies 64 Gbytes of RAM memory. To overcome such potential memory bottlenecks, we have devised a scheme for computing the SVD in which the matrix is provided into a column-partitioned format, with the submatrices being processed one at a time. In contrast to other partitioned schemes, such as the one proposed by the first author in Ref. [2] or the partitioned Proper Orthogonal Decomposition of Ref. [33], which compute the SVD of the entire matrix from the individual SVDs of each submatrices, our scheme addresses the problem in an incremental, sequential fashion: at each increment, the current basis matrix (for the column space of $\boldsymbol{A}_{FE}$) is enriched with the left singular vectors coming from the SVD of the *orthogonal complement*. The advantage of this sequential approach over the concurrent approaches in Refs. [2,33] is that it exploits the existence of linear correlations across the blocks. For instance, in a case in which all submatrices are full rank, and besides, a linear combination of the first submatrix (this may happen when analyzing periodic functions), our sequential approach would require performing a single SVD—that of the first matrix. By contrast, the concurrent approaches in Refs. [2,33], would not only need to calculate the SVD of all the submatrices, but they would not provide any benefit at all in terms of computer memory (in fact the partitioned scheme would end up being more costly than the standard one-block implementation). Lastly, to accelerate the performance of each SVD on the orthogonal complement of the submatrices, we employ a modified version of the randomized blocked SVD proposed by Martinsson et al. [34], using as prediction for the rank of a given submatrix that of the previous submatrix in the sequence.

1.6. Organization of the paper

The paper is organized as follows. The determination of the orthogonal basis functions and their gradients by using the SVD of the sampling matrix are discussed in Section 2. Although an original contribution of the present work, we have relegated the description of the Sequential Randomized SVD (SRSVD) algorithm to Appendix (in order not to interrupt the continuity of the presentation of the cubature algorithm, which constitutes the primary focus of this paper). On the other hand, the computation of the interpolatory cubature rule by the Discrete Empirical Cubature Method, DECM, is presented in Section 3, and the solution of the continuous sparsification problem in Sections 4 and 5. Except for the DECM, which can be found in the original Ref. [16], we provide the pseudo-codes of all the algorithms involved in both the cubature and the SRSVD. Likewise, we have summarized all the implementation steps in Box 5.1 of Section 5.3. The logic of the proposed methodology can be followed without the finer details from the information in this Box.

Sections 6.1 and 6.2 are devoted to the numerical validation by comparison with the (optimal) quadrature and cubature rules of univariate and multivariate Lagrange polynomials. The example presented in Section 6.3, on the other hand, is intended to illustrate the performance of the method in scenarios where the proposed SRSVD becomes essential—because the integrand matrix exhausts the memory capabilities of the computer at hand. Finally, the application of the proposed CECM to the hyperreduction of a multiscale finite element model is explained in Section 6.4.

2. Orthogonal basis for the integrand

2.1. Basis matrix via SVD

As pointed out in the foregoing, our cubature method does not operate directly on the integrand sampling matrix $\boldsymbol{A}_{FE}$, defined in Eq. (4), but on a basis matrix for its column space, denoted henceforth by $\boldsymbol{U} \in \mathbb{R}^{M\times p}$. Since $\boldsymbol{U}$ will be a linear combination of the columns of $\boldsymbol{A}_{FE}$, which are in turn the discrete representation of the scalar integrand functions we wish to integrate, it follows that the columns of $\boldsymbol{U}$ themselves will be the discrete representations of basis functions for such integrand functions. These basis

functions will be denoted hereafter by $u_i : \Omega \to \mathbb{R}$ $(i = 1, 2 \dots p)$. In analogy to Eq. (4), we can write $\boldsymbol{U}$ in terms of such basis functions as

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{U}^{(1)} \\ \boldsymbol{U}^{(2)} \\ \vdots \\ \boldsymbol{U}^{(N_{el})} \end{bmatrix}_{M \times p} \quad \text{where} \quad \boldsymbol{U}^{(e)} := \begin{bmatrix} \boldsymbol{u}(\bar{\boldsymbol{x}}_1^e) \\ \boldsymbol{u}(\bar{\boldsymbol{x}}_2^e) \\ \vdots \\ \boldsymbol{u}(\bar{\boldsymbol{x}}_r^e) \end{bmatrix}_{r \times p}. \tag{9}$$

while

$$\boldsymbol{u}(\boldsymbol{x}) := \begin{bmatrix} u_1(\boldsymbol{x}) & u_2(\boldsymbol{x}) & \cdots & u_p(\boldsymbol{x}) \end{bmatrix}_{1 \times p}. \tag{10}$$

We shall require these basis functions to be $L_2(\Omega)$-orthogonal, i.e.:

$$\int_\Omega u_i u_j \, d\Omega = \delta_{ij}, \qquad i, j = 1, 2 \dots p, \tag{11}$$

δ_{ij} being the Kronecker delta. By evaluating the above integral using the FE-Gauss rule (as done in Eq. (2)), we get:

$$\int_\Omega u_i u_j \, d\Omega = \sum_{e=1}^{N_{el}} \sum_{g=1}^{r} u_i(\bar{\boldsymbol{x}}_g^e) W_g^e u_j(\bar{\boldsymbol{x}}_g^e) = \boldsymbol{U}_i^T \, \mathrm{diag}(\boldsymbol{W}_{FE}) \boldsymbol{U}_j, \qquad i, j = 1, 2 \dots p. \tag{12}$$

In the preceding equation, $\boldsymbol{U}_i$ and $\boldsymbol{U}_j$ represents the ith and jth columns of $\boldsymbol{U}$, while $\mathrm{diag}(\boldsymbol{W}_{FE})$ stands for a diagonal matrix containing the entries of the vector of FE weights $\boldsymbol{W}_{FE}$ (defined in Eq. (5)). The above condition can be cast in a compact form as

$$\boldsymbol{U}^T \, \mathrm{diag}(\boldsymbol{W}_{FE}) \boldsymbol{U} = \boldsymbol{I}, \tag{13}$$

$\boldsymbol{I}$ being the $p \times p$ identity matrix. The preceding equation reveals that orthogonality in the $L_2(\Omega)$ sense for the basis functions u_i translates into orthogonality for the columns of $\boldsymbol{U}$ in the sense defined by the following inner product

$$\langle \boldsymbol{v}_1, \boldsymbol{v}_2 \rangle_W := \boldsymbol{v}_1^T \, \mathrm{diag}(\boldsymbol{W}_{FE}) \boldsymbol{v}_2 \tag{14}$$

$(\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^M)$.

In order to determine $\boldsymbol{U}$ from $\boldsymbol{A}_{FE}$, we compute first the (truncated) Singular Value Decomposition of the *weighted* matrix defined by

$$\bar{\boldsymbol{A}} := \mathrm{diag}(\sqrt{\boldsymbol{W}_{FE}}) \boldsymbol{A}_{FE} \tag{15}$$

that is:

$$\bar{\boldsymbol{A}} = \bar{\boldsymbol{U}} \boldsymbol{S} \boldsymbol{V}^T + \bar{\boldsymbol{E}}, \tag{16}$$

symbolyzed in what follows as the operation:

$$[\bar{\boldsymbol{U}}, \boldsymbol{S}, \boldsymbol{V}] = \mathtt{SVD}(\bar{\boldsymbol{A}}, \epsilon_{svd}). \tag{17}$$

Here, $\bar{\boldsymbol{U}} \in \mathbb{R}^{M \times p}$, $\boldsymbol{S} \in \mathbb{R}^{p \times p}$ and $\boldsymbol{V} \in \mathbb{R}^{dP \times p}$ are the matrices of left-singular vectors, singular values and right-singular vectors, respectively. The matrix of singular values is diagonal with $S_{ii} \geq S_{i-1,i-1} > 0$, while the matrices of left-singular and right-singular vectors obey the orthogonality conditions

$$\bar{\boldsymbol{U}}^T \bar{\boldsymbol{U}} = \boldsymbol{I}, \qquad \boldsymbol{V}^T \boldsymbol{V} = \boldsymbol{I}. \tag{18}$$

Matrix $\bar{\boldsymbol{E}}$ in Eq. (16), on the other hand, represents the truncation term, which is controlled by a user-specified tolerance $0 \leq \epsilon_{svd} \leq 1$ such that

$$\|\bar{\boldsymbol{E}}\|_F \leq \epsilon_{svd} \|\bar{\boldsymbol{A}}\|_F \tag{19}$$

(here $\|\bullet\|_F$ denotes the Frobenius norm). The desired basis matrix $\boldsymbol{U}$ is computed from $\bar{\boldsymbol{U}}$ as

$$\boldsymbol{U} = \mathrm{diag}(\sqrt{\boldsymbol{W}_{FE}})^{-1} \bar{\boldsymbol{U}}. \tag{20}$$

It can be readily seen that, in doing so, the $\boldsymbol{W}_{FE}$-orthogonality condition defined in Eq. (13) is satisfied. Multiplication of both sides of Eq. (16) allows us to write

$$\boldsymbol{A}_{FE} = \boldsymbol{U} \boldsymbol{S} \boldsymbol{V}^T + \boldsymbol{E}, \tag{21}$$

where

$$\boldsymbol{E} := \mathrm{diag}(\sqrt{\boldsymbol{W}_{FE}})^{-1} \bar{\boldsymbol{E}}. \tag{22}$$

Notice that, by virtue of the definition of Frobenius norm, and by using the preceding expression, we have that

$$\|\bar{\boldsymbol{E}}\|_F^2 = \mathrm{tr}\,(\bar{\boldsymbol{E}}^T \bar{\boldsymbol{E}}) = \mathrm{tr}\,(\boldsymbol{E}^T \, \mathrm{diag}(\boldsymbol{W}_{FE}) \boldsymbol{E}) = \|\boldsymbol{E}\|_W^2, \tag{23}$$

where tr $(\bullet)$ stands for the trace operator, and $\|\bullet\|_W$ designates the norm induced by the inner product introduced in Eq. (14). Since the same reasoning can be applied to $\|\bar{\boldsymbol{A}}\|_F$, we can alternatively write the truncation condition (19) as

$$\|\boldsymbol{E}\|_W \leq \epsilon_{svd}\|\boldsymbol{A}_{FE}\|_W. \tag{24}$$

Remark 2.1. When $\boldsymbol{A}_{FE}$ is too large to be processed as a single matrix, we shall use, rather than the standard SVD (17), the sequential randomized SVD alluded to in the introductory section (1.5):

$$[\bar{\boldsymbol{U}}, \boldsymbol{S}, \boldsymbol{V}] = \texttt{SRSVD}([\bar{\boldsymbol{A}}_1, \bar{\boldsymbol{A}}_2, \dots \bar{\boldsymbol{A}}_s], \epsilon_{svd}) \tag{25}$$

(here $[\bar{\boldsymbol{A}}_1, \bar{\boldsymbol{A}}_2, \dots \bar{\boldsymbol{A}}_s]$ stands for a partition of the weighted matrix $\bar{\boldsymbol{A}}$). The implementation details are provided in Algorithm 6 of Appendix.

2.2. Constant function

We argued in Section 1.4 that the efficiency of the proposed search algorithm relies on one fundamental requirement: the volume of the domain is to be exactly integrated—i.e., the sum of the cubature weights must be equal to the volume of the domain $V = \int_\Omega d\Omega$. If the integrand functions are provided as a collection of analytical expressions, this can be achieved by incorporating a constant function in such a collection, with the proviso that the value for the constant should be sufficiently high so that the SVD regards the function as representative within the sample.

The same reasoning applies when the only data available is the empirical matrix $\boldsymbol{A}_{FE}$: in this case, we may make $\boldsymbol{A}_{FE} \leftarrow [\boldsymbol{A}_{FE}, c\mathbf{1}]$, where $\mathbf{1}$ is an all-ones vector and $c \in \mathbb{R}$ the aforementioned constant. Alternatively, to make the procedure less contingent upon the employed constant c, we may expand, rather than the original matrix $\boldsymbol{A}_{FE}$, the basis matrix $\boldsymbol{U}$ itself. To preserve column-wise orthogonality, we proceed by first computing the component of the all-ones vector orthogonal to the column space of $\boldsymbol{U}$ (with respect to the inner product (14)):

$$\boldsymbol{v} = \mathbf{1} - \boldsymbol{U}\boldsymbol{U}^T \operatorname{diag}(\boldsymbol{W}_{FE})\mathbf{1} = \mathbf{1} - \boldsymbol{U}\boldsymbol{U}^T\boldsymbol{W}_{FE}. \tag{26}$$

If $\|\boldsymbol{v}\|_2 \approx 0$, then no further operation is needed (the column space of $\boldsymbol{U}$ already contains the all-ones vector); otherwise, we set $\boldsymbol{v} \leftarrow \boldsymbol{v}/\|\boldsymbol{v}\|_2$, and expand $\boldsymbol{U}$ as $\boldsymbol{U} \leftarrow [\boldsymbol{v}, \boldsymbol{U}]$.

Lemma 2.1. *If the column space of the basis matrix $\boldsymbol{U}$ contains the all-ones (constant) vector, then*

$$\boldsymbol{E}^T\boldsymbol{W}_{FE} = \mathbf{0}, \tag{27}$$

that is, the integrals of the functions whose discrete representation are the truncation matrix $\boldsymbol{E}$ in the SVD (21) are all zero.

Proof. By construction, the truncation term $\boldsymbol{E}$ admits also a decomposition of the form $\boldsymbol{E} = \boldsymbol{U}_\perp\boldsymbol{S}_\perp\boldsymbol{V}_\perp^T$, where $\langle \boldsymbol{U}_\perp, \boldsymbol{U}_\perp\rangle_W = \boldsymbol{I}$, $\langle \boldsymbol{U}_\perp, \boldsymbol{U}\rangle_W = \mathbf{0}$ and $\boldsymbol{V}_\perp^T\boldsymbol{V} = \boldsymbol{I}$. Thus, replacing this decomposition into Eq. (27), we arrive at

$$\boldsymbol{E}^T\boldsymbol{W}_{FE} = \boldsymbol{V}_\perp\boldsymbol{S}_\perp(\boldsymbol{U}_\perp^T\boldsymbol{W}_{FE}) = \mathbf{0}. \tag{28}$$

The proof boils down thus to demonstrate that $\boldsymbol{U}_\perp^T\boldsymbol{W}_{FE} = \mathbf{0}$. This follows easily from the condition that $\langle \boldsymbol{U}_\perp, \boldsymbol{U}\rangle_W = \mathbf{0}$. Indeed, since the all-ones vector pertains to the column space of $\boldsymbol{U}$, the matrix of trailing modes $\boldsymbol{U}_\perp$ is also orthogonal to the all-ones vector, hence

$$\langle \boldsymbol{U}_\perp, \mathbf{1}\rangle_W = \mathbf{0} \;\Rightarrow\; \boldsymbol{U}_\perp^T \operatorname{diag}(\boldsymbol{W}_{FE})\mathbf{1} = \boldsymbol{U}_\perp^T\boldsymbol{W}_{FE} = \mathbf{0}. \tag{29}$$

2.3. Evaluation of basis functions

During the weight-reduction process, it is necessary to repeatedly evaluate the basis functions, as well as their spatial gradient, at points, in general, different from the Gauss points of the mesh.

2.3.1. Integrand given as analytical expression

If the analytical expressions for the integrand functions $\boldsymbol{A}(\boldsymbol{x})$ and their spatial derivatives $\dfrac{\partial \boldsymbol{A}(\boldsymbol{x})}{\partial x_i}$ $(i = 1, 2 \dots d)$ are available, these evaluations can be readily performed by using the singular values and right-singular vectors of decomposition (21) as follows:

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{A}(\boldsymbol{x})\boldsymbol{V}\boldsymbol{S}^{-1}, \tag{30}$$

and

$$\frac{\partial \boldsymbol{u}(\boldsymbol{x})}{\partial x_i} = \frac{\partial \boldsymbol{A}(\boldsymbol{x})}{\partial x_i}\boldsymbol{V}\boldsymbol{S}^{-1}, \qquad i = 1 \dots d. \tag{31}$$

Proof. Post-multiplication of both sides of Eq. (21) by $\boldsymbol{V}$ leads to

$$\boldsymbol{A}_{FE}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\boldsymbol{V} + \boldsymbol{E}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\boldsymbol{V} + \boldsymbol{U}_\perp\boldsymbol{S}_\perp\boldsymbol{V}_\perp^T\boldsymbol{V}, \tag{32}$$

where we have used the matrices introduced in the proof of Lemma 2.1. By virtue of the orthogonality conditions $\boldsymbol{V}^T\boldsymbol{V} = \boldsymbol{I}$ and $\boldsymbol{V}_\perp^T\boldsymbol{V} = \boldsymbol{0}$, the above equation becomes $\boldsymbol{A}_{FE}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{S}$; postmultiplication by $\boldsymbol{S}^{-1}$ finally leads to $\boldsymbol{U} = \boldsymbol{A}_{FE}\boldsymbol{V}\boldsymbol{S}^{-1}$. This equation holds, not only for $\boldsymbol{A}_{FE} = \boldsymbol{A}(\boldsymbol{X}_{FE})$, but for any $\boldsymbol{A}(\boldsymbol{x})$, as stated in Eq. (30). □

Remark 2.2. Eq. (31) indicates that the gradient of the jth basis function depends inversely on the jth singular value. Negligible singular values, thus, may give rise to inordinately high gradients, causing convergence issues during the nonlinear readjustment problem. To avoid these numerical issues, the SVD truncation threshold ϵ_{svd} (see Expression (24)) should be set to a sufficiently large value (typically $\epsilon_{svd} \geq 10^{-6}$).

2.3.2. Interpolation using Gauss points

In general, however, the analytical expression for the integrand functions are not available, and therefore, the preceding equations cannot be employed for retrieving the values of the orthogonal basis functions. This is the case encountered when dealing with FE-based reduced-order models, where the only information we have at our disposal is the value of the basis functions at the Gauss points of the finite elements, represented by submatrices $\boldsymbol{U}^{(e)} \in \mathbb{R}^{r\times p}$ $(e = 1, 2 \dots N_{el})$ in Eq. (9).

In a FE-based reduced-order model, at element level, the integrand functions are, in general, a nonlinear function of the employed nodal shape functions. It appears reasonable, thus, to use also polynomial interpolatory functions to estimate the values of the basis functions at other points of the element using as interpolatory points, rather than the nodes of the element, their Gauss points. If we denote by $\boldsymbol{N}^{(e)} : \Omega^e \to \mathbb{R}^{1\times r}$ the r interpolatory functions (arranged as a row matrix), then we can write

$$\boldsymbol{u}(\boldsymbol{x}) = \boldsymbol{N}^{(e)}(\boldsymbol{x})\boldsymbol{U}^{(e)}, \qquad \boldsymbol{x} \in \Omega^e. \tag{33}$$

Likewise, the spatial derivatives can be determined as

$$\frac{\partial \boldsymbol{u}(\boldsymbol{x})}{\partial x_i} = \boldsymbol{B}_i^{(e)}(\boldsymbol{x})\boldsymbol{U}^{(e)}, \qquad \boldsymbol{x} \in \Omega^e, \quad i = 1 \dots d \tag{34}$$

where

$$\boldsymbol{B}_i^{(e)} := \frac{\partial \boldsymbol{N}^{(e)}}{\partial x_i}, \quad i = 1 \dots d. \tag{35}$$

The level of accuracy of this estimation will depend on the number of Gauss point per element with respect to the order of the nodal shape functions, as well as the distorsion of the physical domain Ω^e with respect to the parent domain—which is the cause of the aforementioned nonlinearity. It may be argued that if the element has no distorsion, the evaluation of the integrand via Eq. (33) will be exact if the proper number of Gauss points is used. For instance, in a small-strains structural problem, if the element is a 4-noded bilinear quadrilateral, with no distorsion (i.e., a rectangle), and the term to be integrated is the virtual internal work, then the integrand is represented exactly by a quadratic.[2] polynomial. Such a polynomial possesses $(2+1)^2 = 9$ monomials, and therefore a 3×3 Gauss rule would be needed. Notice that this is an element integration rule with one point more per spatial direction than the integration standard rule for bilinear quadrilateral elements (2×2). Nevertheless, numerical experience shows that in general it is not necessary to "overintegrate" the element provided that a sufficiently dense mesh is used in areas where high gradients are likely to appear. This will be corroborated in the example presented in Section 6.4 Likewise, in this regard, it should be pointed out that the proposed algorithm for solving the "readjustment" step will not work if the integrand functions are constant over the element. Thus, linear triangles/tetrahedra for integrating internal forces in structural problems cannot be used in the proposed approach.

The expression for $\boldsymbol{N}^{(e)}$ and $\boldsymbol{B}_i^{(e)}$ in terms of the coordinates of the Gauss points $\{\bar{\boldsymbol{x}}_1^e, \bar{\boldsymbol{x}}_2^e \dots \bar{\boldsymbol{x}}_r^e\}$ can be obtained by the standard procedure used in deriving FE shape functions, see e.g. Ref. [35]. Firstly, we introduce the mapping $\boldsymbol{\varphi}^e : \Omega^e \to \Omega^{e'}$ defined by

$$[\boldsymbol{x}']_i = \frac{1}{L_i}[\boldsymbol{x} - \bar{\boldsymbol{x}}_0^e]_i, \qquad i = 1 \dots d \tag{36}$$

where $[\bullet]_i$ symbolyzes the ith component of the argument, $\bar{\boldsymbol{x}}_0^e = (\sum_{g=1}^r \bar{\boldsymbol{x}}_g^e)/r$ is the centroid of the Gauss points, and L_i is a scaling length defined by

$$L_i = \max_{g=1\dots r}(|[\bar{\boldsymbol{x}}_g^e - \bar{\boldsymbol{x}}_0^e]_i|), \qquad i = 1 \dots d. \tag{37}$$

The expression of the shape functions in terms of the scaled positions of the Gauss points $\bar{\boldsymbol{X}}' = \{\bar{\boldsymbol{x}}'_1, \bar{\boldsymbol{x}}'_2 \dots \bar{\boldsymbol{x}}'_r\}$, where $\bar{\boldsymbol{x}}'_g = \boldsymbol{\varphi}^e(\bar{\boldsymbol{x}}_g^e)$, is given by

$$\boldsymbol{N}^{(e)}(\boldsymbol{x}') = \boldsymbol{P}(\boldsymbol{x}')\boldsymbol{P}^{-1}(\bar{\boldsymbol{X}}'). \tag{38}$$

Here, $\boldsymbol{P}(\boldsymbol{x}') \in \mathbb{R}^{1\times r}$ is the row matrix containing the monomials up to the order corresponding to the number and distribution of Gauss points at point $\boldsymbol{x}' = \boldsymbol{\varphi}^e(\boldsymbol{x})$; for instance, for the case of a 2D $q \times q$ rule, where $r = q^2$, this row matrix adopts the form

$$\boldsymbol{P}(\boldsymbol{x}') := [x'^0_1 x'^0_2, x'^1_1 x'^0_2 \cdots x'^i_1 x'^j_2 \cdots x'^{q-1}_1 x'^{q-1}_2]_{1\times r}. \tag{39}$$

[2] Virtual work is the product of virtual strains (which are linear in a 4-noded rectangular) and stresses (which are therefore also linear).

The other matrix appearing in Eq. (38), $\boldsymbol{P}(\bar{\boldsymbol{X}}') \in \mathbb{R}^{r\times r}$, known as the *moment matrix* [35], is formed by stacking the result of applying the preceding mapping to the set of scaled Gauss points $\bar{\boldsymbol{X}}'$. Provided that the element is not overly distorted (no negative Jacobians in the original isoparameteric transformation), the invertibility of $\boldsymbol{P}(\bar{\boldsymbol{X}}')$ is guaranteed thanks to the coordinate transformation Eq. (36)—which ensures that the coordinates of all points range between −1 and 1, therefore avoiding scaling issues in the inversion.

As for the gradient of the shape functions in Eq. (35), by applying the chain rule, we get that

$$\boldsymbol{B}_i^{(e)} = \frac{\partial \boldsymbol{P}(\boldsymbol{x}')}{\partial x'_j}\frac{\partial x'_j}{\partial x_i}\boldsymbol{P}^{-1}(\bar{\boldsymbol{X}}') = \frac{1}{L_i}\frac{\partial \boldsymbol{P}(\boldsymbol{x}')}{\partial x'_i}\boldsymbol{P}^{-1}(\bar{\boldsymbol{X}}'). \tag{40}$$

3. Discrete empirical Cubature method (DECM)

Once the orthogonal basis matrix $\boldsymbol{U}$ has been computed by the weighted SVD outlined in Section 2.1, the next step consists in determining an *interpolatory cubature rule* (featuring as many points as functions to be integrated) for the basis functions $\boldsymbol{u} : \Omega \to \mathbb{R}^{1\times p}$. As pointed out in Section 1.4, we employ for this purpose the Empirical Cubature Method, proposed by the first author in Ref. [2], and further refined in Ref. [16]. We call it here *Discrete* Empirical Cubature Method, DECM, to emphasize that the cubature points are selected among the Gauss points of the mesh. The DECM, symbolized in what follows as the operation

$$[\boldsymbol{z}, \boldsymbol{w}^*] \leftarrow \mathrm{DECM}(\boldsymbol{U}, \boldsymbol{W}_{FE}), \tag{41}$$

takes as inputs the basis matrix $\boldsymbol{U} \in \mathbb{R}^{M\times p}$ and the vector of positive FE weights $\boldsymbol{W}_{FE} \in \mathbb{R}^M$; and returns a set of p indexes $\boldsymbol{z} \subset \{1, 2 \dots M\}$ and a vector of *positive* weights $\boldsymbol{w}^*$ such that

$$\boldsymbol{U}(\boldsymbol{z}, :)\boldsymbol{w}^* = \boldsymbol{b}. \tag{42}$$

Here, $\boldsymbol{U}(\boldsymbol{z}, :)$ denotes, in the so-called "colon" notation [36] (the one used by Matlab), the submatrix of $\boldsymbol{U}$ formed by the rows corresponding to indexes $\boldsymbol{z}$, while $\boldsymbol{b} \in \mathbb{R}^p$ is the vector of "exact" integrals of the basis functions, that is:

$$\boldsymbol{b} = \int_\Omega \boldsymbol{u}^T \, d\Omega = \boldsymbol{U}^T \boldsymbol{W}_{FE}. \tag{43}$$

The points associated to the selected rows will be denoted hereafter by $\boldsymbol{X}^* = \{\boldsymbol{x}_1^*, \boldsymbol{x}_2^* \dots \boldsymbol{x}_p^*\}$ ($\boldsymbol{X}^* \subset \boldsymbol{X}_{FE}$). Hence, according to the notational convention introduced in Remark 1.1, $\mathbb{P}_{\boldsymbol{z}}\boldsymbol{U}$ may be alternatively expressed as

$$\mathbb{P}_{\boldsymbol{z}}\boldsymbol{U} = \boldsymbol{u}(\boldsymbol{X}^*) = \begin{bmatrix} \boldsymbol{u}(\boldsymbol{x}_1^*) \\ \boldsymbol{u}(\boldsymbol{x}_2^*) \\ \vdots \\ \boldsymbol{u}(\boldsymbol{x}_p^*) \end{bmatrix} = \begin{bmatrix} u_1(\boldsymbol{x}_1^*) & u_2(\boldsymbol{x}_1^*) & \cdots & u_p(\boldsymbol{x}_1^*) \\ u_1(\boldsymbol{x}_2^*) & u_2(\boldsymbol{x}_2^*) & \cdots & u_p(\boldsymbol{x}_2^*) \\ \vdots & \vdots & \ddots & \vdots \\ u_1(\boldsymbol{x}_p^*) & u_2(\boldsymbol{x}_p^*) & \cdots & u_p(\boldsymbol{x}_p^*) \end{bmatrix}_{p\times p} \tag{44}$$

Remark 3.1. It should be stressed that the solution to problem Eq. (42) is not unique. Rather, the number of possible solutions grows combinatorially with the ratio between the total number of Gauss points and the number of functions (M/p). The situation is illustrated in Fig. 2, where we graphically explain how the DECM works for the case of $M = 6$ Gauss points and polynomial functions up to order 1 (it can be readily shown that the orthogonal functions in this case are $u_1 = \sqrt{3/2}x$ and $u_2 = \sqrt{1/2}$, displayed in Fig. 2.a). The problem, thus, boils down to select $p = 2$ points out of $M = 6$, such that the resulting weights are positive. In Fig. 2.b, we plot each $\boldsymbol{u}(\bar{\boldsymbol{x}}_g)$ ($g = 1, 2 \dots 6$) along with the vector of exact integrals, which in this case is equal to $\boldsymbol{b} = [0, \sqrt{2}]^T$. It follows from this representation that, out of the $\binom{M}{p} = \binom{6}{2} = 15$ possible combinations, only 9 pairs are valid solutions. The DECM[3] chooses $\boldsymbol{x}_1^* = \bar{\boldsymbol{x}}_4$ and $\boldsymbol{x}_2^* = \bar{\boldsymbol{x}}_1$, which is the solution that yields the largest ratio between highest and lowest weight. Other possible solutions are, for instance, pairs $\{\bar{\boldsymbol{x}}_1, \bar{\boldsymbol{x}}_6\}$ and $\{\bar{\boldsymbol{x}}_2, \bar{\boldsymbol{x}}_6\}$—observe that in both cases, vector $\boldsymbol{b}$ lies in the cone[4] "positively spanned" by $\{\boldsymbol{u}(\bar{\boldsymbol{x}}_1), \boldsymbol{u}(\bar{\boldsymbol{x}}_6)\}$ and $\{\boldsymbol{u}(\bar{\boldsymbol{x}}_2), \boldsymbol{u}(\bar{\boldsymbol{x}}_6)\}$, respectively.

The reader interested in the points selection algorithm behind the DECM is referred[5] to Ref. [16], Appendix, Algorithm 7.

[3] The first vector $\boldsymbol{u}(\bar{\boldsymbol{x}}_4)$ is chosen because is the one which is most positively parallel to $\boldsymbol{b}$ (notice that, because of symmetry, it might have chosen $\boldsymbol{u}(\bar{\boldsymbol{x}}_3)$ as well). Then it orthogonally projects $\boldsymbol{b}$ onto $\boldsymbol{u}(\bar{\boldsymbol{x}}_4)$, giving $\boldsymbol{s}$, and then search for the vector which is more positively parallel to the residual $\boldsymbol{b} - \boldsymbol{s}$, which in this case is $\boldsymbol{u}(\bar{\boldsymbol{x}}_1)$.

[4] The cone positively spanned by a set of vectors is the set of all possible positive linear combinations of such vectors [37].

[5] It should be pointed that the notation employed in Ref. [16] is different from the one used here. The input of the DECM in Ref. [16] (which is called therein simply the *ECM*) is not $\boldsymbol{U}$, but the transpose of the weighted matrix $\bar{\boldsymbol{U}}$ (defined in Eq. (16)). Likewise the error threshold appearing in Ref. [16] is to be set to zero in order to produce an interpolatory rule.

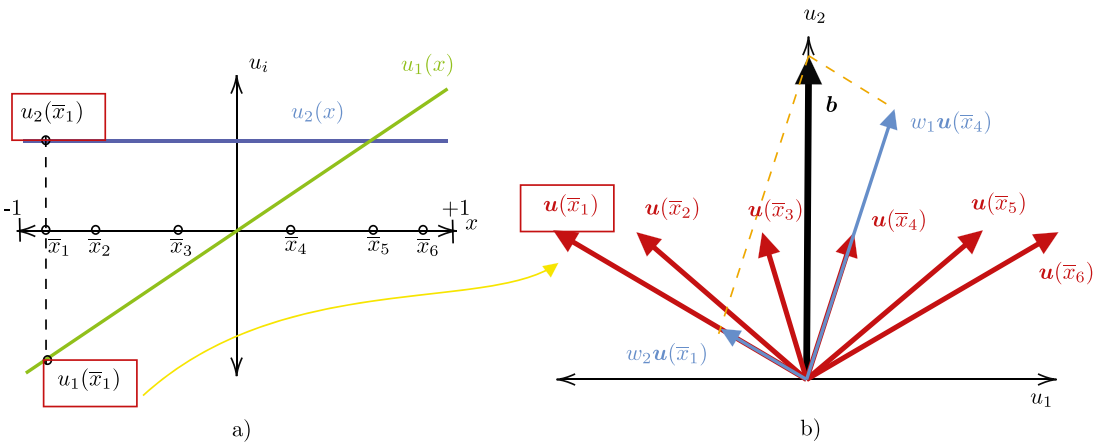


Fig. 2. Performance of the Discrete Empirical Cubature Method (DECM) [16], for the case of the orthonormal functions $u_1 = \sqrt{3/2}x$ and $u_2 = \sqrt{1/2}$ in the interval $\Omega = [-1, 1]$. The number of finite elements is $N_{el} = 1$, and the number of Gauss points $M = r = 6$. (a) Graph of the functions along with the location of the Gauss points $\bar{x}_6 = -\bar{x}_1 = 0.9325$, $\bar{x}_5 = -\bar{x}_2 = 0.6612$, $\bar{x}_4 = -\bar{x}_3 = 0.2386$. (b) Representation in the plane u_1u_2 of each vector $\boldsymbol{u}(\bar{\boldsymbol{x}}_g) = [u_1(\bar{\boldsymbol{x}}_g), u_2(\bar{\boldsymbol{x}}_g)]^T$ $(g = 1, 2 \dots 6)$, along with the vector of exact integrals $\boldsymbol{b} = [0, \sqrt{2}]^T$. The DECM chooses points $\bar{\boldsymbol{x}}_4$ and $\bar{\boldsymbol{x}}_1$, giving weights equal to $w_1^* = 1.5925$ and $w_2^* = 0.4075$.

3.1. Relation between SVD truncation error and the DECM integration error

Let us examine now the error incurred in approximating the "exact" integrals $\boldsymbol{b}_{FE} = \boldsymbol{A}_{FE}^T \boldsymbol{W}_{FE}$ by the DECM cubature rule. This error may be expressed as

$$e_{decm} := \|\boldsymbol{A}_{FE}^T \boldsymbol{W}^* - \boldsymbol{A}_{FE}^T \boldsymbol{W}_{FE}\|_2 \tag{45}$$

where $\boldsymbol{W}^* := \mathbb{P}_{\boldsymbol{z}}^T \boldsymbol{w}^*$ (a vector of the same length as $\boldsymbol{W}_{FE}$, but with nonzero entries only at the indexes specified by $\boldsymbol{z}$). Inserting decomposition (21) in the preceding equation, we get

$$e_{decm} = \| \left(\boldsymbol{V}\boldsymbol{S}(\boldsymbol{U}^T \boldsymbol{W}^* - \boldsymbol{U}^T \boldsymbol{W}_{FE}) \right) + (\boldsymbol{E}^T \boldsymbol{W}^* - \boldsymbol{E}^T \boldsymbol{W}_{FE}) \|_2 \tag{46}$$

From condition (42), it follows that the term involving the basis matrix $\boldsymbol{U}$ vanishes; besides, since by construction the column space of $\boldsymbol{U}$ contains the all-ones vector, we have that, by virtue of Lemma 2.1, $\boldsymbol{E}^T \boldsymbol{W}_{FE} = \boldsymbol{0}$. Thus, Eq. (46) boils down to

$$e_{decm} = \|\boldsymbol{E}^T \boldsymbol{W}^*\|_2. \tag{47}$$

The truncation term $\boldsymbol{E}$ in the above condition is controlled by the SVD tolerance ϵ_{svd} appearing in inequality (24). Thus, the integration error e_{decm} may be lowered to any desired level by decreasing the SVD tolerance ϵ_{svd}. Numerical experience shows that for most problems $e_{decm}/\|\boldsymbol{b}_{FE}\|_2$ is slightly above, but of the same order of magnitude, as ϵ_{svd}.

4. Global sparsification problem

4.1. Formulation

We now concentrate our attention on the *sparsification* problem outlined in Section 1.4.1. The design variables in this optimization problem will be a vector of p weights

$$\boldsymbol{w} = [w_1, w_2 \dots w_p]^T, \qquad w_g \geq 0, \tag{48}$$

(recall that p is the number of orthogonal basis functions we wish to integrate), and the position of the associated points within the domain:

$$\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_p\}, \qquad \boldsymbol{x}_g \in \Omega. \tag{49}$$

With a minor abuse of notation, we shall also use $\boldsymbol{X}$ to denote the variable formed by stacking the position of the points into a column matrix, i.e.: $\boldsymbol{X} = [\boldsymbol{x}_1^T, \boldsymbol{x}_2^T \dots \boldsymbol{x}_p]^T$. On the other hand, we define the *integration residual* as

$$\boldsymbol{r} := \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} \tag{50}$$

that is, as the difference between the approximate and the exact integrals of the basis functions. In the preceding equation, $\boldsymbol{u}(\boldsymbol{X})$ designates the matrix formed by stacking the rows of all $\boldsymbol{u}(\boldsymbol{x}_g) \in \mathbb{R}^{1\times p}$ $(g = 1, 2 \dots p)$ into a single matrix, i.e.:

$$\boldsymbol{u}(\boldsymbol{X}) := \begin{bmatrix} \boldsymbol{u}(\boldsymbol{x}_1) \\ \boldsymbol{u}(\boldsymbol{x}_2) \\ \vdots \\ \boldsymbol{u}(\boldsymbol{x}_p) \end{bmatrix} = \begin{bmatrix} u_1(\boldsymbol{x}_1) & u_2(\boldsymbol{x}_1) & \cdots & u_p(\boldsymbol{x}_1) \\ u_1(\boldsymbol{x}_2) & u_2(\boldsymbol{x}_2) & \cdots & u_p(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ u_1(\boldsymbol{x}_p) & u_2(\boldsymbol{x}_p) & \cdots & u_p(\boldsymbol{x}_p) \end{bmatrix}_{p\times p} \tag{51}$$

With the preceding definitions at hand, the sparsification problem can be formulated as follows:

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{X}} \quad & \|\boldsymbol{w}\|_0 \\ \text{s.t.} \quad & \boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0} \\ & \boldsymbol{w} \geq \boldsymbol{0} \\ & \boldsymbol{X} \subset \Omega \end{aligned} \tag{52}$$

Recall that $\|\boldsymbol{w}\|_0$ stands for the number of nonzero entries of $\boldsymbol{w}$. Thus, the goal in the preceding optimization problem is to find the *sparsest* vector of *positive* weights, along with their associated positions within the domain, that render the *integration residual* $\boldsymbol{r}$ equal to zero.

Remark 4.1. The differences between the sparsification problem presented above and the one described in the introductory section (see Problem (8)) are three. Firstly, in the preceding problem, $\boldsymbol{w} \in \mathbb{R}^p$, i.e., the number of weights is equal to the number of basis functions to be integrated p, whereas in Problem (8), this number is equal to the total number of FE Gauss points M (it is assumed that $p \ll M$). Secondly, the integration residual in Problem (52) appears in the form of an equality constraint, while in Problem (8) appears as an inequality constraint. And thirdly, and most importantly, in Problem (52), the positions of the cubature points are considered design variables—in contrast to the situation encountered in Problem (8), in which the points are forced to coincide with the FE Gauss points, and thus, the only design variables are the weights.

4.2. Proposed sparsification algorithm

Algorithm 1: Proposed two-stage procedure for solving the sparsification problem (52).

Function $[\boldsymbol{X}, \boldsymbol{w}, \boldsymbol{\mathcal{E}}] \leftarrow$ `SPARSIFglo(`$\boldsymbol{X}^*$,$\boldsymbol{w}^*$,N_{steps},$\boldsymbol{\mathcal{A}}$,$\boldsymbol{\mathcal{M}}$`)`

Data: $\boldsymbol{X}^* = \{\boldsymbol{x}_1^*, \boldsymbol{x}_2^* \dots \boldsymbol{x}_p^*\}(\boldsymbol{x}_g^* \in \Omega)$ and $\boldsymbol{w}^* = [w_1^*, w_2^*, \dots, w_p^*]$ $(w_g^* > 0)$: DECM cubature rule (obtained by function (41)). $N_{steps} > 1$: number of steps used to solve the nonlinear problem (in the second stage) associated to the residual constraint $\boldsymbol{r} = \boldsymbol{0}$. $\boldsymbol{\mathcal{A}}$: Remaining variables controlling the solution of this nonlinear problem. $\boldsymbol{\mathcal{M}}$: data structures containing variables needed to evaluate the residual $\boldsymbol{r}$ at any point $\boldsymbol{x} \in \Omega$ (such as the basis matrix itself $\boldsymbol{U}$ and the vector of exact integrals $\boldsymbol{b}$, among others).

Result: *Sparsest* weight vector $\boldsymbol{w} = [w_1, w_2, \dots, w_p]$ $(w_g \geq 0)$ and associated positions $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2 \dots \boldsymbol{x}_p\}(\boldsymbol{x}_g \in \Omega)$ such that $\boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$.

$\boldsymbol{\mathcal{E}}^{old} \leftarrow \emptyset$ `// Initializations`

$N \leftarrow 1$ `// First stage. Number of steps employing in solving the nonlinear problem` $\boldsymbol{r} = \boldsymbol{0}$ `is set to 1.`

$[\boldsymbol{X}^{new}, \boldsymbol{w}^{new}, \boldsymbol{\mathcal{E}}^{new}] \leftarrow$ `SPARSIF(`$\boldsymbol{X}^*$,$\boldsymbol{w}^*$,N,$\boldsymbol{\mathcal{A}}$,$\boldsymbol{\mathcal{M}}$,$\boldsymbol{\mathcal{E}}^{old}$`)` `// SPARSIF() is described in Algorithm 2`

$[\boldsymbol{X}, \boldsymbol{w}, \boldsymbol{\mathcal{E}}] \leftarrow$ `SPARSIF(`$\boldsymbol{X}^{new}$,$\boldsymbol{w}^{new}$,N_{steps},$\boldsymbol{\mathcal{A}}$,$\boldsymbol{\mathcal{M}}$,$\boldsymbol{\mathcal{E}}^{new}$`)` `// Second stage. Number of steps equal to the value specified in the input arguments (typically` $N_{steps} \sim 20$`)`

The proposed approach for arriving at the solution of the preceding problem is to construct a sequence of p-points cubature rules

$$\{\boldsymbol{X}^0, \boldsymbol{w}^0\}, \{\boldsymbol{X}^1, \boldsymbol{w}^1\} \dots \{\boldsymbol{X}^i, \boldsymbol{w}^i\} \dots \{\boldsymbol{X}^{p-m}, \boldsymbol{w}^{p-m}\} \tag{53}$$

such that

$$\|\boldsymbol{w}^{k+1}\|_0 = \|\boldsymbol{w}^k\|_0 - 1 \tag{54}$$

that is, such that each weight vector in the sequence has one non-zero less than the previous one. The first element in the sequence will be taken as the cubature rule provided by the DECM (see Section 3):

$$\boldsymbol{X}^0 = \boldsymbol{X}^*, \qquad \boldsymbol{w}^0 = \boldsymbol{w}^*. \tag{55}$$

The algorithm proceeds from this initial point by the recursive application of an operation consisting in driving the weight of one single point to zero (while forcing the remaining points and weights to obey the constraints appearing in Problem (52)); this step will be symbolized hereafter as the function

$$[C, \boldsymbol{X}^{new}, \boldsymbol{w}^{new}, \boldsymbol{\mathcal{E}}^{new}] \leftarrow \texttt{MAKE1ZERO}(\boldsymbol{X}^{old}, \boldsymbol{w}^{old}, N_{steps}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}}, \boldsymbol{\mathcal{E}}^{old}). \tag{56}$$

This function takes as inputs a given cubature rule $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$, and tries to return a cubature rule $\{\boldsymbol{X}^{new}, \boldsymbol{w}^{new}\}$ with at least one less nonzero weight. The success of this operation is indicated by the output Boolean variable C ($C = false$ if it fails in producing a sparser cubature rule).

The other inputs in (56) are the following: (1) N_{steps}: number of steps used to solve the nonlinear problem associated to the residual constraint $\boldsymbol{r} = \boldsymbol{0}$. (2) $\mathcal{A}$: Remaining variables controlling the solution of this nonlinear problem (such as the convergence tolerance for the residual). (3) $\mathcal{M}$ and $\mathcal{E}^{old}$ are data structures containing the variables needed to evaluate the residual $\boldsymbol{r}$ at any point $\boldsymbol{x} \in \Omega$. $\mathcal{M}$ encompasses those variables that do not change during the execution (such as the basis matrix itself $\boldsymbol{U}$, the vector of exact integrals $\boldsymbol{b}$, see Eq. (43), the nodal coordinates of the FE mesh, the connectivity table, the Gauss coordinates, and in the case of analytical evaluation, the product $\boldsymbol{V}_s := \boldsymbol{V}\boldsymbol{S}^{-1}$ appearing in Eqs. (30) and (31)). $\mathcal{E}^{old}$, on the other hand, comprises element variables that are computed *on demand*,[6] such as the inverse of the moment matrix in Eq. (38) and the scaling factors in Eq. (36) (required for the interpolation described in Section 2.3.2).

Algorithm 2: Sparsification process, given an initial cubature rule $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$ and a number of steps N (invoked in Line 5 of Algorithm 1) .

1 **Function** $[\boldsymbol{X}, \boldsymbol{w}, \mathcal{E}^{new}] \leftarrow$ SPARSIF$(\boldsymbol{X}^{old}, \boldsymbol{w}^{old}, N, \mathcal{A}, \mathcal{M}, \mathcal{E}^{old})$

Data: $\boldsymbol{X}^{old} \subset \Omega$, $\boldsymbol{w}^{old} \in \mathbb{R}^p$. N and $\mathcal{A}$: Variables controlling the solution of the nonlinear problem $\boldsymbol{r} = \boldsymbol{0}$. $\mathcal{M}$ and $\mathcal{E}^{old}$: data structures containing variables needed to evaluate the residual $\boldsymbol{r}$ at any point $\boldsymbol{x} \in \Omega$.

Result: *Sparsest* weight vector $\boldsymbol{w} \in \mathbb{R}^p$ and associated positions $\boldsymbol{X}$ such that $\boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$. $\mathcal{E}^{new}$: updated structure data with information necessary for performing element interpolation.

2 $C \leftarrow true$

3 **while** $C = true$ **do**

4 $[C, \boldsymbol{X}^{new}, \boldsymbol{w}^{new}, \mathcal{E}^{new}] \leftarrow$ MAKE1ZERO$(\boldsymbol{X}^{old}, \boldsymbol{w}^{old}, N, \mathcal{A}, \mathcal{M}, \mathcal{E}^{old})$ // Described in Algorithm 3.

5 **if** $C = false$ **then return** // MAKE1ZERO() has failed to produce a cubature rule with one nonzero weight less.

6

7 **if** $w_i^{new} \geq \boldsymbol{0}$, $\forall i$ **then** $\boldsymbol{X} \leftarrow \boldsymbol{X}^{new}$; $\boldsymbol{w} \leftarrow \boldsymbol{w}^{new}$ // The weights solution $\boldsymbol{w}$ can only have positive weights

8

9 $\boldsymbol{X}^{old} \leftarrow \boldsymbol{X}^{new}$; $\boldsymbol{w}^{old} \leftarrow \boldsymbol{w}^{new}$; $\mathcal{E}^{old} \leftarrow \mathcal{E}^{new}$ // Update positions, weights and element interpolation data.

10 **end**

Due to its greedy or "myopic" character, the DECM tends to produce weights distribution in which most of the weights are relatively small in comparison with the total volume of the domain. We have empirically observed that the readjustment problem associated to the elimination of these small weights is moderately nonlinear, and in general, one step suffices to ensure convergence. However, as the algorithm advances in the sparsification process, the weights to be zeroed become larger, and, as a consequence, the readjustment problem becomes more nonlinear. In this case, to ensure convergence, it is necessary to reduce the weights progressively. To account for this fact, we have devised the two-stage procedure described in Algorithm 1. In the first stage (see Line 4), the sparsification process (sketched in turn in Algorithm 2) is carried out by decreasing the weight of each chosen weight in one single step. In the second stage, see Line 5, we take the cubature rule produced in the first stage, and try to further decrease the number of nonzero weights by using a higher number of steps $N_{steps} > 1$.

5. Local sparsification problem

After presenting the global sparsification procedure, we now focus on fleshing out the details of the fundamental building block of such a procedure, which is the above mentioned function MAKE1ZERO(), appearing in Line 4 of Algorithm 2.

The procedural steps are described in the pseudocode of Algorithm 3. Given a cubature rule $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$, with $\|\boldsymbol{w}^{old}\|_0 = m$ ($2 \leq m \leq p$), we seek a new cubature rule $\{\boldsymbol{X}, \boldsymbol{w}\}$ with $\|\boldsymbol{w}\|_0 = m - 1$. Notice that there are m different routes for eliminating a nonzero weight—as many as nonzero weights. It may be argued that the higher the contribution of a given point to the residual $\boldsymbol{r}$, the higher the difficulty of converging to feasible solutions using as initial point the cubature rule $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$. To account for this fact, we sort the indexes of the points with nonzero weights in ascending order according to its contribution to the residual (which is $s_i = w_i^{old}\|\boldsymbol{u}(\boldsymbol{x}_i^{old})\|_2$, see Line 4). The actual subroutine that performs the zeroing operation is SOLVERES() in Line 9. If this subroutine fails to determine a feasible solution in which the chosen weight is set to zero, then the operation is repeated with the next point in the sorted list, and so on until arriving at the desired sparser solution (if such a solution exists at all)

5.1. Modified Newton–Raphson algorithm

We now move to the above mentioned subroutine SOLVERES(), appearing in Line 9 of Algorithm 3—and with pseudo-code explained in Algorithm 4. This subroutine is devoted to the calculation of the position and weights of the remaining points when the weight of the chosen "control" point R (R $\in \{1, 2 \dots p\}$, $w_{\mathrm{R}}^{old} \neq 0$) is set to zero—by solving the nonlinear equation corresponding to the integration conditions $\boldsymbol{r}(\boldsymbol{X}, \boldsymbol{w}) = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$.

[6] In the proposed algorithm, we compute the necessary interpolation variables for each element of the mesh dynamically, as they are needed, rather than precomputing them all at once. Indeed, each time the position of the points is updated, we check which elements of the mesh contain the updated points. If all the elements have been previously visited, we use the information stored in $\mathcal{E}^{old}$ to perform the interpolation; otherwise, we compute the required interpolation variables for the new elements and update $\mathcal{E}^{old}$ into $\mathcal{E}^{new}$ with the new data.

Algorithm 3: Given a cubature rule $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$, this algorithm tries to determine a cubature rule $\{\boldsymbol{X}, \boldsymbol{w}\}$ with one additional zero weight (invoked in Line 5 of Algorithm 2).

1 **Function** $[C, \boldsymbol{X}, \boldsymbol{w}, \mathcal{E}] \leftarrow$ MAKE1ZERO$(\boldsymbol{X}^{old}, \boldsymbol{w}^{old}, N, \mathcal{A}, \mathcal{M}, \mathcal{E})$
  Data: Similar to inputs in Algorithm 2.
  Result: $\boldsymbol{X} \in \Omega$ and $\boldsymbol{w} \in \mathbb{R}^p$ such that $\boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$. If $C = true$, then $\|\boldsymbol{w}\|_0 = \|\boldsymbol{w}^{old}\|_0 - 1$ (i.e., the output vector weight $\boldsymbol{w}$ has one more zero that the input weight $\boldsymbol{w}^{old}$); otherwise, $\boldsymbol{w} \leftarrow \boldsymbol{w}^{old}$, $\boldsymbol{X} \leftarrow \boldsymbol{X}^{old}$
2 $\mathbf{F} \leftarrow$ Indexes nonzero entries of $\boldsymbol{w}^{old}$ // $\mathbf{F} \subset \{1, 2 \dots p\}$
3 $[\boldsymbol{u}^{old}_{\mathbf{F}}, \bullet, \bullet] \leftarrow$ EVALBASIS $(\boldsymbol{X}^{old}_{\mathbf{F}}, \mathcal{M}, \mathcal{E})$ // Evaluate basis functions $\boldsymbol{u}$ at points $\boldsymbol{X}^{old}_{\mathbf{F}} = \{\boldsymbol{X}^{old}_{\mathbf{F}(1)}, \boldsymbol{X}^{old}_{\mathbf{F}(2)} \dots \boldsymbol{X}^{old}_{\mathbf{F}(m)}\}$ (using the procedure described in Section 2.3) . $\boldsymbol{u}^{old}_{\mathbf{F}}$ is a $m \times p$ matrix, m being the number of nonzero entries of $\boldsymbol{w}^{old}$
4 $s_i \leftarrow w^{old}_i \|\boldsymbol{u}^{old}_{\mathbf{F}}(i, :)\|_2$ $(i = 1, 2 \dots m)$ // $\boldsymbol{u}^{old}_{\mathbf{F}}(i, :)$ denotes the i-th row of $\boldsymbol{u}^{old}_{\mathbf{F}}$
5 $\boldsymbol{i} \leftarrow$ Sort $\{s_1, s_2 \dots s_m\}$ in ascending order; return indexes describing the arragement
6 $C \leftarrow false$; $j \leftarrow 1$ // Initializations
7 **while** $j \leq m$ ***AND*** $C = false$ **do**
8 R $\leftarrow \boldsymbol{f}(j)$ // Index of candidate weight to be zeroed (R $\in \{1, 2 \dots p\}$)
9 $[C, \boldsymbol{X}, \boldsymbol{w}, \mathcal{E}] \leftarrow$ SOLVERES (R, $\boldsymbol{X}^{old}, \boldsymbol{w}^{old}, N, \mathcal{A}, \mathcal{M}, \mathcal{E}$) // See Algorithm 4
10 $j \leftarrow j + 1$ // If $C = false$, trying next candidate.
11 **end**

Algorithm 4: Given a cubature rule $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$ and an index R $\in \{1, 2 \dots p\}$, where $w^{old}_{\mathrm{R}} \neq 0$, this algorithm determines a cubature rule $\{\boldsymbol{X}, \boldsymbol{w}\}$ with one additional zero at point R ($w_{\mathrm{R}} = 0$) .

1 **Function** $[C, \boldsymbol{X}, \boldsymbol{w}, \mathcal{E}] \leftarrow$ SOLVERES$(R, \boldsymbol{X}^{old}, \boldsymbol{w}^{old}, N, \mathcal{A}, \mathcal{M}, \mathcal{E})$
  Data: R $\in \{1, 2 \dots p\}$: Candidate index. Remaining inputs: similar to inputs of function in Algorithm 3.
  Result: $\boldsymbol{X} \in \Omega$ and $\boldsymbol{w} \in \mathbb{R}^p$ such that $\boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$. If $C = true$, then $\|\boldsymbol{w}\|_0 = \|\boldsymbol{w}^{old}\|_0 - 1$ and $w_{\mathrm{R}} = 0$
2 $n \leftarrow 1$; $C \leftarrow true$; $w^{ref} \leftarrow w^{old}_{\mathrm{R}}$; $\boldsymbol{X} \leftarrow \boldsymbol{X}^{old}$; $\boldsymbol{w} \leftarrow \boldsymbol{w}^{old}$ // Initializations
3 **while** $n \leq N$ ***AND*** $C = true$ **do**
4 $w_{\mathrm{R}} = w^{ref}(1 - n/N)$
5 $[C, \boldsymbol{X}, \boldsymbol{w}, \mathcal{E}, \mathbf{P}] \leftarrow$ NEWTONRmod $(\boldsymbol{X}, \boldsymbol{w}, \mathrm{R}, \mathcal{A}, \mathcal{M}, \mathcal{E})$ // Modified Newton-Raphson, see Algorithm 5
6 $n \leftarrow n + 1$
7 **end**
8 **if** $C = false$ **then** $\boldsymbol{X} \leftarrow \boldsymbol{X}^{old}$; $\boldsymbol{w} \leftarrow \boldsymbol{w}^{old}$ // No feasible solution found.
9

To facilitate convergence, the weight w_{R} is gradually reduced at a rate dictated by the number of steps N (so that $w_{\mathrm{R}} = w^{old}_{\mathrm{R}}(1 - n/N)$ at step n, see Line 4).

Suppose we have converged to the solution $\{\boldsymbol{X}_{(n-1)}, \boldsymbol{w}_{(n-1)}\}$ and we want to determine the solution for the next step n using a Newton–Raphson iterative scheme, modified so as to account for the constraints that the points must remain within the domain, and that the weights should be positive (although this latter constraint will be relaxed, as explained in what follows). The pseudo-code of this modified Newton–Raphson scheme is described in turn in Algorithm 5. The integration residual at iteration $k \leq K_{max}$ is computed in Line 7. This residual admits the following decomposition in terms of unknown and known variables:

$$\boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{u}^T(\boldsymbol{X}_{\mathbf{L}})\boldsymbol{w}_{\mathbf{L}} + \boldsymbol{u}^T(\boldsymbol{X}_{\mathbf{P}})\boldsymbol{w}_{\mathbf{P}} + \boldsymbol{u}^T(\boldsymbol{X}_R)w_R - \boldsymbol{b}. \tag{57}$$

Here, $\mathbf{L} \subset \{1, 2 \dots p\}$ denotes the set of points whose positions and weights are unknown, while $\mathbf{P} \subset \{1, 2 \dots p\}$ is the set in which the positions are fixed, but the weights are unknown. At the first iteration, $\mathbf{P} = \emptyset$ (see Line 2). The unknown weights will be collectively denoted hereafter by $\boldsymbol{w}_{\mathbf{S}} = [\boldsymbol{w}^T_{\mathbf{L}}, \boldsymbol{w}^T_{\mathbf{P}}]^T$, and the vector of unknowns (including positions and weights) by $\boldsymbol{q} := [\boldsymbol{X}^T_{\mathbf{L}}, \boldsymbol{w}^T_{\mathbf{S}}]^T$.

If the Euclidean norm of the residual is not below the prescribed error tolerance (Line 7), we compute, as customary in Newton-Raphson procedures, a correction $\boldsymbol{\Delta q} = [\Delta \boldsymbol{X}^T_{\mathbf{L}}, \Delta \boldsymbol{w}^T_{\mathbf{S}}]^T$ by obtaining *one* solution of the *underdetermined* linear equation

$$\widehat{\boldsymbol{J}}\, \boldsymbol{\Delta q} = -\boldsymbol{r}. \tag{58}$$

Here, $\widehat{\boldsymbol{J}}$ stands for the block matrix of the Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{p \times (d+1)p}$ formed by the rows corresponding to the indexes of the unknown positions $\boldsymbol{X}_{\mathbf{L}}$ and the unknown weights $\boldsymbol{w}_{\mathbf{S}}$, i.e:

$$\widehat{\boldsymbol{J}} := \begin{bmatrix} \boldsymbol{J}_{\boldsymbol{X}_{\mathbf{L}}} & \boldsymbol{J}_{\boldsymbol{w}_{\mathbf{S}}} \end{bmatrix}, \tag{59}$$

Algorithm 5: Modified Newton-Rapshon algorithm for solving the constrained nonlinear equation $\boldsymbol{r} = \boldsymbol{u}^T(\boldsymbol{X})\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$, using as initial guess $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$. The unknowns are the position and weights of the nonzero entries of $\boldsymbol{w}^{old}$, except for $\boldsymbol{w}^{old}_{\mathrm{R}}$, which is given (this function is invoked in Line 5 of Algorithm 4).

1 **Function** $[C, \boldsymbol{X}, \boldsymbol{w}, \mathcal{E}] \leftarrow$ NEWTONRmod$(\boldsymbol{X}^{old}, \boldsymbol{w}^{old}, R, \mathcal{A}, \mathcal{M}, \mathcal{E})$

Data: $\{\boldsymbol{X}^{old}, \boldsymbol{w}^{old}\}$: Cubature rule previous step. R $\in \{1, 2 \dots p\}$: Index controlled point. $\mathcal{A} = \{K_{max}, \epsilon_{NR}, N_{neg}\}$, K_{max}: Maximum number of iterations; ϵ_{NR}: Tolerance convergence residual; N_{neg}: Maximum number of negative weights allowed during iterations; $\mathcal{M}$ and $\mathcal{E}$: data structures containing variables needed to evaluate the residual $\boldsymbol{r}$ at any point $\boldsymbol{x} \in \Omega$

Result: $C = true$: The Newton-based iterative algorithm has converged to a feasible solution $\{\boldsymbol{X}, \boldsymbol{w}\}$ of the equation $\boldsymbol{r} = \boldsymbol{u}(\boldsymbol{X})^T\boldsymbol{w} - \boldsymbol{b} = \boldsymbol{0}$, where $w_{\mathrm{R}} = w^{old}_{\mathrm{R}}$ is given.

2 $\mathbf{P} \leftarrow \emptyset$ // Indexes of points with fixed position but unknown weights

3 $\mathbf{L} \leftarrow$ Indexes nonzero entries of $\boldsymbol{w}^{old}$, excluding R

4 $k \leftarrow 1$; $C \leftarrow false$; $\mathbf{S} \leftarrow \mathbf{L}$; $\epsilon_{svd} \leftarrow 10^{-10}$ // Initializations

5 **while** $k \leq K_{max}$ ***AND*** $C = false$ **do**

6 $[\boldsymbol{u}(\boldsymbol{X}^{old}), \nabla\boldsymbol{u}(\boldsymbol{X}^{old}), \mathcal{E}] \leftarrow$ EVALBASIS $(\boldsymbol{X}^{old}, \mathcal{M}, \mathcal{E})$ // Determine basis functions and gradients at $\boldsymbol{X}^{old}$

7 $\boldsymbol{r} \leftarrow \boldsymbol{u}^T(\boldsymbol{X}^{old})\boldsymbol{w}^{old} - \boldsymbol{b}$ // Integration residual

8 **if** $\|\boldsymbol{r}\|_2 \leq \epsilon_{NR}$ **then**

9 $C \leftarrow true$; $\boldsymbol{X} \leftarrow \boldsymbol{X}^{old}$; $\boldsymbol{w} \leftarrow \boldsymbol{w}^{old}$ // Converged solution

10 **else**

11 $\boldsymbol{J}_S \leftarrow [\boldsymbol{J}_{X_S}, \boldsymbol{J}_{w_S}]$ // Jacobian matrix (indexes S). $\boldsymbol{J}_X$ and $\boldsymbol{J}_w$ are defined in Eqs. (60) and (61)

12 $E_{feas} \leftarrow false$ // $E_{feas} \leftarrow false$ while tentative solution not feasible

13 **while** $k \leq K_{max}$ ***AND*** $E_{feas} = false$ **do**

14 $\hat{\boldsymbol{J}} \leftarrow [\boldsymbol{J}_{X_\mathbf{L}}, \boldsymbol{J}_{w_\mathbf{S}}]$ // $\mathbf{L} \subseteq \mathbf{S}$ (L: indexes unknown weights and positions)

15 $[\boldsymbol{U}_J, \boldsymbol{S}_J, \boldsymbol{G}^T] \leftarrow$ SVD$(\hat{\boldsymbol{J}}, \epsilon_{svd})$ // $\hat{\boldsymbol{J}} \approx \boldsymbol{U}_J\boldsymbol{S}_J\boldsymbol{G}$ with relative error $\epsilon_{svd} = 10^{-10}$

16 $n_{dofs} \leftarrow (d+1)$length$(\boldsymbol{L})$ + length$(\boldsymbol{P})$ // Number of unknowns (d: no of spatial dimensions)

17 **if** $n_{dofs} <$ length$(\boldsymbol{S}_J)$ **then**

18 **break** // Overdetermined system (no solution). Exiting internal loop without convergence

19 **else**

20 $\boldsymbol{c} \leftarrow -\boldsymbol{S}_J^{-1}\boldsymbol{U}_J^T\boldsymbol{r}$ // $\boldsymbol{U}_J\boldsymbol{S}_J\boldsymbol{G}\Delta\boldsymbol{q} = -\boldsymbol{r} \;\Rightarrow\; \boldsymbol{G}\Delta\boldsymbol{q} = -\boldsymbol{S}_J^{-1}\boldsymbol{U}_J^T\boldsymbol{r}$

21 $\Delta\boldsymbol{q} \leftarrow$ Sparse solution of $\boldsymbol{G}\Delta\boldsymbol{q} = \boldsymbol{c}$ obtained via QR pivoting // In Matlab: $\Delta\boldsymbol{q} = \boldsymbol{G}\backslash\boldsymbol{c}$

22 $\boldsymbol{X}_\mathbf{L} \leftarrow \boldsymbol{X}^{old}_\mathbf{L} + \Delta\boldsymbol{q}_x$; $\boldsymbol{w}_\mathbf{S} \leftarrow \boldsymbol{w}^{old}_\mathbf{S} + \Delta\boldsymbol{q}_w$ // $\Delta\boldsymbol{q}_x = \Delta\boldsymbol{q}(1 : (d+1)$length$(\boldsymbol{L}))$, $\Delta\boldsymbol{q}_w$: remaining entries

23 $m_{neg} \leftarrow$ Number of negative entries of $\boldsymbol{w}$

24 **if** $m_{neg} \geq N_{neg}$ **then break**

25 $\mathbf{Y} \leftarrow$ Indexes points outside the domain$(\boldsymbol{X}_{\mathbf{Y}(i)} \notin \Omega)$

26 **if** $\boldsymbol{Y} \neq \emptyset$ **then**

27 $\mathbf{P} \leftarrow \mathbf{P} \cup \mathbf{Y}$; $\mathbf{L} \leftarrow \mathbf{L} \setminus \mathbf{Y}$ $\mathbf{S} \leftarrow \mathbf{L} \cup \mathbf{P}$ // Repeat iteration with $\boldsymbol{X}_\mathbf{P}$ fixed in previous position

28 **else**

29 $\boldsymbol{X}^{old} \leftarrow \boldsymbol{X}$; $\boldsymbol{w}^{old} \leftarrow \boldsymbol{w}$; $E_{feas} = true$ // All points inside, update and exit internal loop

30 **end**

31 $k \leftarrow k + 1$

32 **end**

33 **end**

34 **if** $E_{feas} = false$ **then break** (no feasible solution found, exiting without convergence)

35 **end**

36 **end**

where

$$\boldsymbol{J}_X := \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{X}} = \begin{bmatrix} w_1\nabla^T u_1(\boldsymbol{x}_1) & w_2\nabla^T u_1(\boldsymbol{x}_2) & \dots & w_p\nabla^T u_1(\boldsymbol{x}_p) \\ w_1\nabla^T u_2(\boldsymbol{x}_1) & w_2\nabla^T u_2(\boldsymbol{x}_2) & \dots & w_p\nabla^T u_2(\boldsymbol{x}_p) \\ \vdots & \vdots & \ddots & \vdots \\ w_1\nabla^T u_p(\boldsymbol{x}_1) & w_2\nabla^T u_p(\boldsymbol{x}_2) & \dots & w_p\nabla^T u_p(\boldsymbol{x}_p) \end{bmatrix}_{p \times d\,p}, \tag{60}$$

and

$$\boldsymbol{J}_w := \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{w}} = \boldsymbol{u}^T(\boldsymbol{X}) = \begin{bmatrix} u_1(\boldsymbol{x}_1) & u_1(\boldsymbol{x}_2) & \dots & u_1(\boldsymbol{x}_p) \\ u_2(\boldsymbol{x}_1) & u_2(\boldsymbol{x}_2) & \dots & u_2(\boldsymbol{x}_p) \\ \vdots & \vdots & \ddots & \vdots \\ u_p(\boldsymbol{x}_1) & u_p(\boldsymbol{x}_2) & \dots & u_p(\boldsymbol{x}_p) \end{bmatrix}_{p \times p}. \tag{61}$$

Recall that the gradients of the basis functions can be determined by Eq. (31), if the analytical expressions of the integrand functions are available, or by interpolation via Eq. (34)—using the values of the basis functions at the Gauss points of the element containing the corresponding point. These operations are encapsulated in the function EVALBASIS(), invoked in Line 6

Once we have computed $\boldsymbol{\Delta q}$ from Eq. (58), we update the positions of the points and the weights in Line 22. Since the basis functions are only defined inside the domain Ω (this is one of the constraints appearing in the sparsification problem (52)), it is necessary to first identify (Line 25) and then correct the positions of those points that happen to fall outside the domain. The identification is made by determining which finite elements contain the points in their new positions; for the sake of computational efficiency, the search is limited to a patch of elements centered at the element containing the point at the previous iteration, and located within a radius $\|\Delta \boldsymbol{X}_I\|_2$ ($I \in \mathbf{L}$)—the mesh connectivities, stored in the data structure $\mathcal{M}$, greatly expedites this search task. If it happens that a given point is not inside any element ($\boldsymbol{X}_I \notin \Omega$ for some $I \in \mathbf{L}$), then we set $\boldsymbol{X}_I \leftarrow \boldsymbol{X}_I^{old}$, $\mathbf{P} \leftarrow \mathbf{P} \cup I$, and $\mathbf{L} \leftarrow \mathbf{L} \setminus I$ (see Line 27). Notice that this amounts to "freezing" the position of this critical point at the value of the previous iteration during the remaining iterations of the current step.[7] This operation is to be repeated until all the points lie within the domain—ensuring this is the job of the internal *while* loop starting in Line 13.

The other constraint defining a feasible solution in the sparsification problem (52) is the positiveness of the weights. However, we argued in Section 1.4.1 that, since the volume is exactly integrated, the tendency when one of the weights is reduced is that the remaining weights increase to compensate for the loss of volume. Furthermore, according to the sorting criterion employed in Line 4 of Algorithm 3, negative weights are the first to be zeroed in each local sparsification step, and, thus, tend to disappear as the algorithm progresses. For these reasons, the solution procedure does not incorporate any specific strategy for enforcing positiveness of the weights—rather, we limit ourselves to keep the number of negative weights below a user-prescribed threshold M_{neg} during the iterative procedure (Line 24 in Algorithm 5). Nevertheless, as a precautionary measure, Line 7 in Algorithm 2 prevents negative weights from appearing in the final solution.

5.2. Properties of Jacobian matrix and maximum sparsity

It only remains to address the issue of how to solve the system of linear Eq. (58). Solving this system of equations is worthy of special consideration because of two reasons: firstly, the system is *underdetermined* (more unknowns than equations), and, secondly, the Jacobian matrix $\widehat{\boldsymbol{J}}$ may become *rank-deficient* during the final steps of the sparsification process, specially in 2D and 3D problems.

5.2.1. Rank deficiency

That the Jacobian matrix $\widehat{\boldsymbol{J}}$ may become rank-deficient can be readily demonstrated by analyzing the case of the integration of polynomials in Cartesian domains. A polynomial of order t gives rise to $p = (t+1)^d$ ($d = 1, 2$ or 3) integration conditions (as many as monomials). If one assumes that the Jacobian matrix $\widehat{\boldsymbol{J}}$ remains full rank during the entire sparsification process, then it follows that the number of optimal points one can get under such an assumption, denoted henceforth by m_{eff}, is when $\widehat{\boldsymbol{J}}$ becomes square[8] (or underdetermined with less than d surplus unknowns); this condition yields

$$m_{eff} = \texttt{ceil}\left(\frac{p}{d+1}\right) = \texttt{ceil}\left(\frac{(t+1)^d}{d+1}\right) \tag{62}$$

where ceil() rounds its argument to the nearest integer greater or equal than itself. For 1D polynomials ($d = 1$), it is readily seen that m_{eff} coincides with the number of points of the well-known (optimal) Gauss quadrature rule; for instance, for $t = 3$ (cubic polynomials), the above equation gives $m_{eff} = 4/2 = 2$ integration points. This implies that in this 1D case, the Jacobian matrix *does* remain full rank during the process, as presumed. However, this does not hold in the 2D and 3D cases. For instance, for 3D cubic polynomials ($t = 3$, $d = 3$), the above equation yields $m_{eff} = (1+3)^3/(1+3) = 16$ points, yet it is well known that 8-points tensor product rule ($2 \times 2 \times 2$) can integrate exactly cubic polynomials in any cartesian domain. This means that, in this 3D case, from the rule with 16 nonzero weights, to the cubature rule with 8 nonzero weights, the Jacobian matrix $\widehat{\boldsymbol{J}}$ must remain *necessarily* rank-deficient.

To account for this potential rank-deficiency, we determine the truncated SVD of $\widehat{\boldsymbol{J}}$ (with error threshold $\epsilon_{SVD} = 10^{-10}$ to avoid near-singular cases) in Line 15 of Algorithm 5: $\widehat{\boldsymbol{J}} \approx \boldsymbol{U}_J \boldsymbol{S}_J \boldsymbol{G}$. Replacing $\widehat{\boldsymbol{J}}$ by this decomposition in Eq. (58), and pre-multiplying both sides of the resulting equation by $\boldsymbol{S}_J^{-1}\boldsymbol{U}_J^T$, we obtain, by virtue of the property $\boldsymbol{U}_J^T\boldsymbol{U}_J = \boldsymbol{I}$:

$$\boldsymbol{G}\boldsymbol{\Delta q} = \boldsymbol{c}, \tag{63}$$

where $\boldsymbol{G}$ denotes the transpose of the orthogonal matrix of right-singular vectors of $\widehat{\boldsymbol{J}}$, while $\boldsymbol{c} = -\boldsymbol{S}_J^{-1}\boldsymbol{U}_J^T\boldsymbol{r}$, $\boldsymbol{S}_J$ being the diagonal matrix of singular values, and $\boldsymbol{U}_J$ the matrix of left singular vectors.

[7] This can be done because system (58) is underdetermined, and therefore, one can constrain some points not to move and still find a solution. It should be noticed that these constrained points are freed at the beginning of each step, see Line 2.

[8] Because if there are less unknowns than equations, there are no solution to the equation $\boldsymbol{r} = \boldsymbol{0}$.

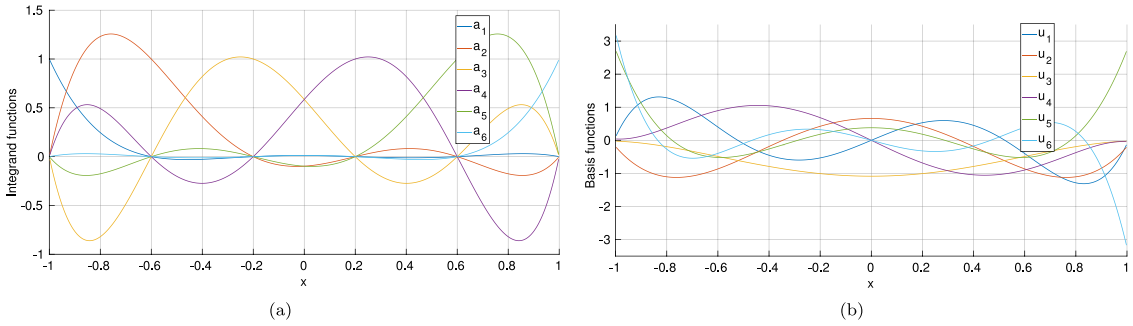


Fig. 3. Integration of univariate Lagrange polynomials of degree $p = 5$. (a) Original integrand functions $a_1, a_2 \dots a_6$. (b) Orthogonal basis functions u_1, u_2 $\dots u_6$ derived from the weighted SVD described in step 5 of Box 5.1. The CECM operates on these basis functions rather than on the original integrand functions in Fig. 3(a).

5.2.2. Underdetermination and sparse solutions

Let us discuss now the issue of underdeterminacy. It is easy to show that the preceding system of equations remains underdetermined during the entire sparsification process, with a degree of underdeterminacy (surplus of unknowns over number of equations) decaying at each sparsification step until the optimum is reached, when $\boldsymbol{G}$ becomes as square as possible. For instance, at the very first step of the process, in a problem with p basis functions, $\widehat{\boldsymbol{J}}$ is by construction[9] full rank (i.e., there are p linearly independent equations), while the number of unknowns is $(p-1)(1+d)$ ($m = p-1$ points with d unknowns coordinates associated to the position of each point and one unknown associated to its weight). Thus, the solution space in this case is of dimension $(d(p-1)-1)$.

To update the weights and the positions, we need to pick up *one* solution from this vast space. The standard approach in Newton's method for underdetermined systems (and also the method favored in the literature on generalized Gaussian quadratures [28,29,31]) is to use the *least ℓ_2-norm solution*, which is simply $\boldsymbol{\Delta q} := \boldsymbol{G}^{+}\boldsymbol{c}$, where $\boldsymbol{G}^{+} = \boldsymbol{G}^{T}(\boldsymbol{G}\boldsymbol{G}^{T})^{-1}$ is the pseudo-inverse of $\boldsymbol{G}$ (notice that in our case[10] $\boldsymbol{G}^{+} = \boldsymbol{G}^{T}$). However, we do not use this approach here because the resulting solution tends to be *dense*, and this implies that the positions of all the cubature points have to updated at all iterations. This is a significant disadvantage in our interpolatory framework, since updating the position of one point entails an interpolation error of greater or lesser extent depending on the functions being interpolated and the distance from the FE Gauss points. Thus, it would be beneficial for the overall accuracy of the method to determine solutions that *minimize the number of positions being updated at each iteration*—incidentally, this would also help to reduce the computational effort associated to the spatial search carried out in Line 25 of Algorithm 5. This requisite natural calls for *solution methods that promote sparsity*. For this reason, we use here (see Line 21 in Algorithm 5) the QR factorization with column pivoting (QRP) proposed in Golub et al. [36] (page 300, Algorithm 5.6.1), which furnishes a solution with as many nonzero entries as equations.[11] An alternative strategy would be to determine the *least ℓ_1-norm solution*, which, as discussed in Section 1.2, also promotes sparsity [9,12]. However, computing this solution would involve addressing a convex, nonquadratic optimization problem at each iteration, and this would require considerably more effort and sophistication than the simple QRP method employed in Line 21.

5.3. Summary

By way of conclusion, we summarize in Box 5.1, all the operations required to determine an optimal cubature rule using as initial data the location of the FE Gauss points, their corresponding weights, and the values at such Gauss points of the functions we wish to efficiently integrate.

6. Numerical assessment

A repository containing both the Continuous Empirical Cubature Method (CECM), as well as the Sequential Randomized SVD (SRSVD), allowing to reproduce the following examples is publicly available at https://github.com/Rbravo555/CECM-continuous-empirical-cubature-method

1. Given the coordinates of the nodes of the finite element mesh, the array of element connectivities, and the position of the Gauss points for each element in the parent domain, determine the location of such points in the physical domain: $\boldsymbol{X}_{FE} = \{\bar{\boldsymbol{x}}_1^1, \bar{\boldsymbol{x}}_2^1, \dots \bar{\boldsymbol{x}}_i^e \dots\}$. Likewise, compute the vector of (positive) finite element weights ($\boldsymbol{W}_{FE} \in \mathbb{R}^M$) for each of these points as the product of the corresponding Gauss weights and the Jacobian of the transformation from the parent domain to the physical domain.
2. Determine the values at all Gauss points $\boldsymbol{X}_{FE}$ of the parameterized function $\boldsymbol{a} : \Omega \times D \to \mathbb{R}^n$ we wish to efficiently integrate for the chosen parameters $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \dots \boldsymbol{\mu}_P\} \subset D$, and store the result in matrix $\boldsymbol{A}_{FE} \in \mathbb{R}^{M \times Pn}$ (see Eq. (4)). In the case of hyperreduced-order models, the analytical expression of the integrand functions is normally not available as an explicit function of the input parameters, and constructing matrix $\boldsymbol{A}_{FE}$ entails solving the corresponding governing equations for the chosen input parameters. If the matrix proves to be too large to fit into main memory, it should be partitioned into column blocks $\boldsymbol{A}_{FE1}$, $\boldsymbol{A}_{FE2}$, … $\boldsymbol{A}_{FEp}$. Such blocks need not be loaded into main memory all at once.
3. Compute the weighted matrix $\bar{\boldsymbol{A}} := \text{diag}\sqrt{\boldsymbol{W}_{FE}}\boldsymbol{A}$ (see Eq (15)) (alternatively, one may directly store in Step 2, rather than $\boldsymbol{A}_{FE}$, $\bar{\boldsymbol{A}}$ itself; this is especially convenient when the matrix is treated in a partitioned fashion, because it avoids loading the submatrices twice).
4. Determine the SVD of $\bar{\boldsymbol{A}}$ ($\bar{\boldsymbol{A}} \approx \bar{\boldsymbol{U}}\boldsymbol{S}\boldsymbol{V}$), with relative truncation tolerance ϵ_{svd} equal to the desired error threshold for the integration (see Eq. (17)). If the matrix is relatively small, one can use directly standard SVD implementations (see function SVDT in Algorithm 7 of Appendix). If the matrix is large but still fits into main memory without compromising the machine performance, the incremental randomized SVD proposed in Appendix (function RSVDinc, described in Algorithm 9) may be used instead. Lastly, if the matrix does not fit into main memory, and is therefore provided in a partitioned format, the Sequential Randomized SVD described also in Appendix (function SRSVD() in Algorithm 6) is to be used.
5. Determine a $\boldsymbol{W}_{FE}$-orthogonal basis matrix for the range of $\boldsymbol{A}_{FE}$ by making $\boldsymbol{U} = \text{diag}(\sqrt{\boldsymbol{W}_{FE}})^{-1}\bar{\boldsymbol{U}}$. Following the guidelines outlined in Section 2.2, augment $\boldsymbol{U}$ with one additional column if necessary so that the column space of $\boldsymbol{U} \in \mathbb{R}^{M \times p}$ contains the constant function.
6. Apply the Discrete Empirical Cubature Method (DECM, see Section 3) to compute a set of indexes $\boldsymbol{z}$ and positive weights $\boldsymbol{w}^*$ such that $\boldsymbol{U}(\boldsymbol{z}, :)^T\boldsymbol{w}^* = \boldsymbol{U}^T\boldsymbol{W}_{FE}$ (see Eq. (41)).
7. Using as initial solution the weights $\boldsymbol{w}^*$ obtained by the DECM, as well as the corresponding positions $\boldsymbol{X}^*$, solve the *sparsification* problem (52) by means of function SPARSIFglo in Algorithm 1:

   $$[\boldsymbol{X}, \boldsymbol{w}, \boldsymbol{\mathcal{E}}] \leftarrow \texttt{SPARSIFglo}(\boldsymbol{X}^*, \boldsymbol{w}^*, N_{steps}, \boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{M}})$$

   The desired cubature rule is given by $\boldsymbol{w}^{cecm} = [w_{g_1}, w_{g_2}, \dots w_{g_m}]$ ($w_{g_i} > 0$) and $\boldsymbol{X}^{cecm} = \{\boldsymbol{x}_{g_1}, \boldsymbol{x}_{g_2}, \dots \boldsymbol{x}_{g_m}\}$, where $g_1, g_2 \dots g_m$ denote the indexes of the nonzero entries of the (sparse) output weight vector $\boldsymbol{w}$.

Box 5.1: Algorithmic steps involved in the proposed *Continuous Empirical Cubature Method* (CECM).

6.1. Univariate polynomials

We begin the assessment of the proposed methodology by examining the example used for motivating the proposal: the integration of univariate polynomials in the domain $\Omega = [-1, 1]$. The employed finite element mesh features $N_{el} = 200$ equally-sized elements, and $r = 4$ Gauss points per element, resulting in a total of $M = 800$ Gauss points. Given a degree $p > 0$, and a set of $P = p + 1$ equally space nodes $x_1, x_2 \dots x_P$, we seek the optimal integration rule for the Lagrange polynomials

$$a_i(x) = \prod_{j=1, i \neq j}^{P} \frac{x - x_j}{x_i - x_j}, \qquad i = 1, 2 \dots P \tag{64}$$

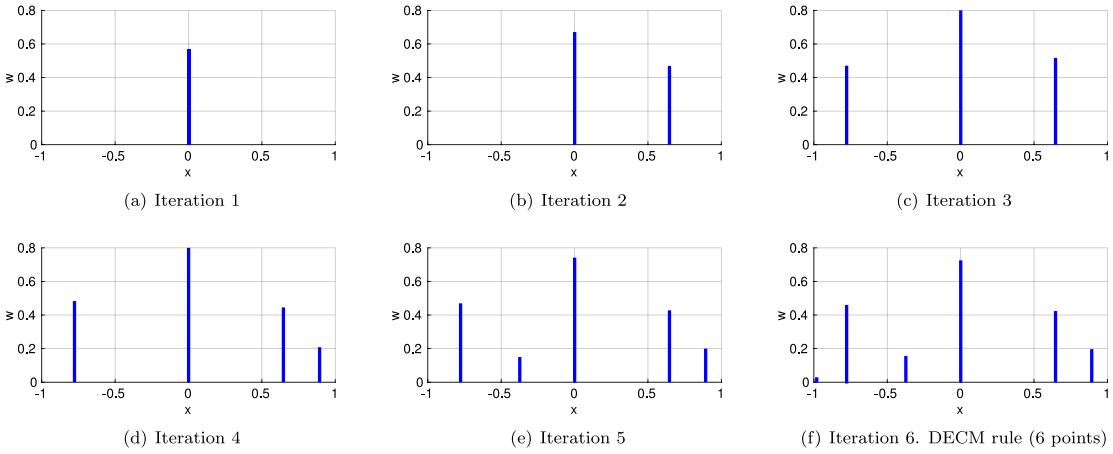
(graphically represented in Fig. 3(a) for degree up to $p = 5$).

The value of these Lagrange polynomials at the M Gauss points are stored in the matrix $\boldsymbol{A}_{FE} \in \mathbb{R}^{M \times P}$, which is then subjected to the weighted SVD (step 5 in Box 5.1) for determining L_2-orthogonal basis functions u_1, u_2 … u_P—plotted in Fig. 3(b) for the case $p = 1, 2 \dots 5$. The truncation tolerance in the SVD of expression (17) is set in this case to $\epsilon_{svd} = 0$, for we seek quadrature rules that *exactly* integrate any polynomial up to the specified degree. The resulting basis matrix $\boldsymbol{U}$ (obtained by Eq. (20) from the left singular vectors of the above mentioned SVD), along with the full-order weight vector $\boldsymbol{W}_{FE} \in \mathbb{R}^M$, are then used to determine an interpolatory quadrature rule by means of the DECM (step 6 in Box 5.1). As commented in Section 3, this algorithm selects one point at each iteration, until arriving at as many points as basis functions. By way of illustration, we show in Fig. 4 the iterative sequence leading to the interpolatory quadrature rule with 6 points (i.e., for polynomials up to degree 5).

[9] On the grounds that $\boldsymbol{u}(\boldsymbol{X}^*)$ is also full rank because otherwise Eq. (42) would not hold.

[10] In this regard, it should be pointed out that Refs. [28,29,31] calculate the pseudo-inverse of the Jacobian matrix as $\hat{\boldsymbol{J}}^+ = \hat{\boldsymbol{J}}^T(\hat{\boldsymbol{J}}\hat{\boldsymbol{J}}^T)^{-1}$, thus ignoring the fact that, as we have argued in the foregoing, $\hat{\boldsymbol{J}}$ might become rank-deficient, and therefore, $\hat{\boldsymbol{J}}\hat{\boldsymbol{J}}^T$ cannot be inverted.

[11] In Matlab, this QRP solution is the one obtained in using the "backslash" operator (or mldivide($\boldsymbol{G}, \boldsymbol{c}$)).



(a) Iteration 1 (b) Iteration 2 (c) Iteration 3

(d) Iteration 4 (e) Iteration 5 (f) Iteration 6. DECM rule (6 points)

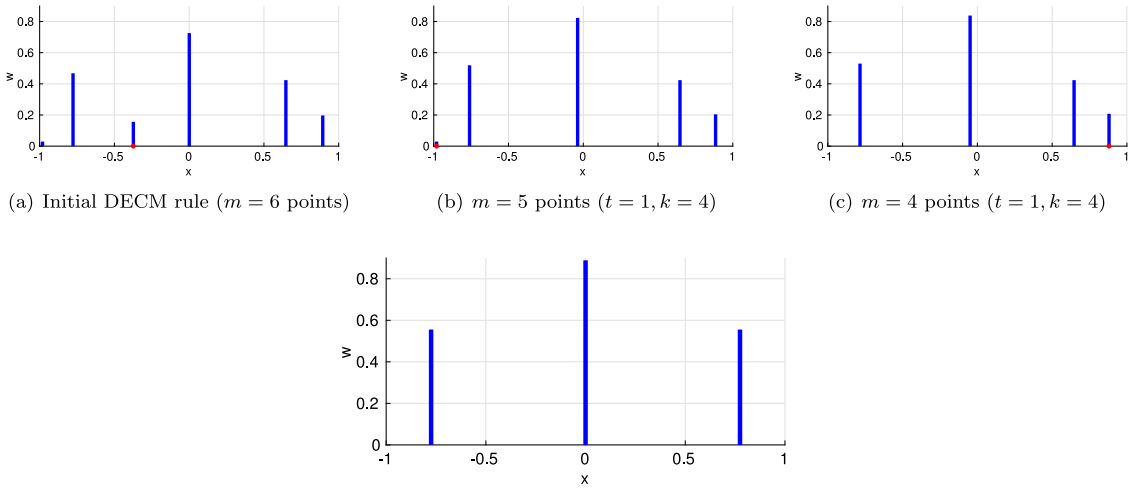
Fig. 4. Location (x) and corresponding weights (w) selected by the DECM (see Box 5.1, step 6), at each iteration, for the case of polynomials of degree $p = 5$. The final integration rule (iteration 6, graph 4(f)) is the starting point for the subsequent sparsification problem (illustrated, in turn, in Fig. 5).

Table 1
Quadrature rules computed by the CECM for univariate polynomials of degree up to $p = 12$. The rightmost column represents the relative deviations with respect to the optimal Gaussian rules (for polynomials of odd degree), calculated as $e^2 = (\|\boldsymbol{X}_{cecm} - \boldsymbol{X}_{gauss}\|_2^2 + \|\boldsymbol{w}_{cecm} - \boldsymbol{w}_{gauss}\|_2^2)/(\|\boldsymbol{X}_{gauss}\|_2^2 + \|\boldsymbol{w}_{gauss}\|_2^2)$.

Degree	Positions	Weights	Error (Gs.)	Degree	Positions	Weights	Error (Gs.)
1	−7.31662253127291E−17	2	2.2504e−16	9	−0.906179845938664	0.236926885056189	4.8426e−16
2	−0.718298153787432	0.784973499347808			−0.538469310105683	0.478628670499367	
	0.464059849765363	1.21502650065219			3.18949244021535E−16	0.568888888888889	
3	−0.577350269189626	1	1.6653e−16		0.538469310105683	0.478628670499366	
	0.577350269189626	1			0.906179845938664	0.236926885056189	
4	−0.966877924131567	0.25396505608136		10	−0.940907514603222	0.151215583523548	
	−0.266817254357718	1.00671682129073			−0.694289509608625	0.336602757683989	
	0.695455188587597	0.739318122627913			−0.287589748592848	0.462395955495934	
5	−0.774596669241483	0.555555555555556	8.4549e−16		0.194969778013914	0.482737964706169	
	5.19179955929866E−17	0.88888888888889			0.637318996155301	0.382659779209725	
	0.774596669241483	0.555555555555556			0.927198931106149	0.184387959380636	
6	−0.837102793435635	0.405516777593141		11	−0.932469514203152	0.17132449237917	1.0484e−15
	−0.245834821655188	0.717131675886906			−0.661209386466264	0.360761573048139	
	0.45930568155797	0.62534095948649			−0.238619186083197	0.467913934572691	
	0.906836939036126	0.252010587033462			0.238619186083197	0.46791393457269	
7	−0.861136311594053	0.347854845137454	5.8993e−16		0.661209386466265	0.360761573048139	
	−0.339981043584856	0.652145154862546			0.932469514203152	0.17132449237917	
	0.339981043584856	0.652145154862545		12	−0.967242104087041	0.088484427217067	
	0.861136311594053	0.347854845137454			−0.803941159117123	0.240851330885801	
8	−0.998731268512389	0.081227685280535			−0.493300094782076	0.37168382888409	
	−0.719380135473419	0.445922523698483			−0.083660679344391	0.434164516926175	
	−0.166434516034323	0.623254258303059			0.346567257597574	0.411907795624614	
	0.446668026062019	0.562352205951177			0.712911972240284	0.308489844409304	
	0.885864638161693	0.287243326766745			0.943166559093504	0.144418256052949	

The final step in the process is the sparsification of the vector of DECM weights to produce the final CECM rule (step 7 in Box 5.1). Table 1 shows the location and weights obtained in this sparsification for polynomials up to degree $p = 12$. The parameters used in this process are: K_{max}= 40, $\epsilon_{NR} = 10^{-8}$, $N_{neg} = 5$ and $N_{steps} = 20$ (the definition of these parameters is given in Algorithm 5), and we use analytical evaluation of the integrand and their derivatives through formulas (30) and (31), respectively.

It can be inferred from the information displayed in Table 1 that, for polynomials of even degree, the CECM provides rules whose number of points is equal to $(p+2)/2$, whereas for polynomials of odd order, the number of points is equal to $(p+1)/2$ in all cases. Thus, for instance, both CECM rules derived from polynomials of degree 4 and 5 possess 3 points; notice that the rule for the



(a) Initial DECM rule ($m = 6$ points) (b) $m = 5$ points ($t = 1, k = 4$) (c) $m = 4$ points ($t = 1, k = 4$)

(d) Final CECM rule, $m = 3$ points ($t = 1, k = 6$)

Fig. 5. Location and weights of the quadrature rules generated during the sparsification process for the case of univariate polynomials of degree $p = 5$ in $\Omega = [-1, 1]$. Variable t represents the number of passes over the loop that selects the weights to be zeroed in Algorithm 3 (see line 7), whereas k indicates the total number of iterations of the modified Newton–Raphson scheme in Algorithm 5. The initial integration rule, displayed in Fig. 5(a) is the one corresponding to the last iteration of the DECM, displayed previously in Fig. 4(f). The red point in each graph indicates the point whose weight is to be zeroed in the following step. The final quadrature rule is the one shown in graph 5(d). The values of the coordinates and weights are given in Table 1, in which one can see that this quadrature rule is indeed the optimal 3-points Gauss rule. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

polynomials of degree 4 is asymmetric, whereas the one for polynomials of degree 5 is symmetrical. Furthermore, comparison of this symmetrical rule with the corresponding Gauss rule with the same number of points[12] reveals that they are identical (relative error below 10^{-15}). The same trend is observed for the remaining CECM rules for polynomials of odd degree. To further corroborate this finding, we extended the study to cover the cases of polynomials from $p = 13$ to $p = 25$, and the result was invariably the same. Thus, we can assert that, at least for univariate polynomials, *the proposed CECM is able to arrive at the optimal cubature rule*, that is, the rule with the *minimal number of points* . To gain further insight into the performance of the method, we present in Fig. 5 the sequence of rules produced during the sparsification process (from the 6-points DECM rule (Fig. 5(a)) to the optimal 3-points (Gauss) rule of Fig. 5(d).

6.2. Multivariate polynomials

Let us now extend the preceding assessment to the integration of *multivariate Lagrange polynomials* in 2D and 3D cartesian domains—for which it is known that the optimal rules are tensor product of univariate Gauss rules [27]. More specifically, we shall focus here on bivariate and trivariate Lagrange polynomials on biunit squares ($\Omega = [-1, 1] \times [-1, 1]$) and cubes ($\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$), respectively.

Given a degree p, and a set of $p + 1$ equally spaced nodes for each direction, let us define the monomials:

$$\Gamma_i(x) = \prod_{h=1, i\neq h}^{P} \frac{x - x_h}{x_i - x_h}, \qquad i = 1, 2 \dots (p + 1) \tag{65}$$

$$\Gamma_j(y) = \prod_{h=1, j\neq h}^{P} \frac{y - y_h}{y_j - y_h}, \qquad j = 1, 2 \dots (p + 1) \tag{66}$$

$$\Gamma_k(z) = \prod_{h=1, k\neq h}^{P} \frac{z - z_h}{z_k - z_h}, \qquad k = 1, 2 \dots (p + 1). \tag{67}$$

The expression for the $P = (p + 1)^2$ integrand functions for the case of bivariate polynomials is given by

$$a_l(x, y) = \Gamma_i(x)\Gamma_j(y), \qquad l = (j - 1)(p + 1) + i, \quad i, j = 1, 2 \dots (p + 1), \tag{68}$$

whereas for trivariate polynomials, the $P = (p + 1)^3$ integrand functions adopt the expression

$$a_l(x, y, z) = \Gamma_i(x)\Gamma_j(y)\Gamma_k(z), \qquad l = (k - 1)(p + 1)^2 + (j - 1)(p + 1) + i, \quad i, j, k = 1, 2 \dots (p + 1). \tag{69}$$

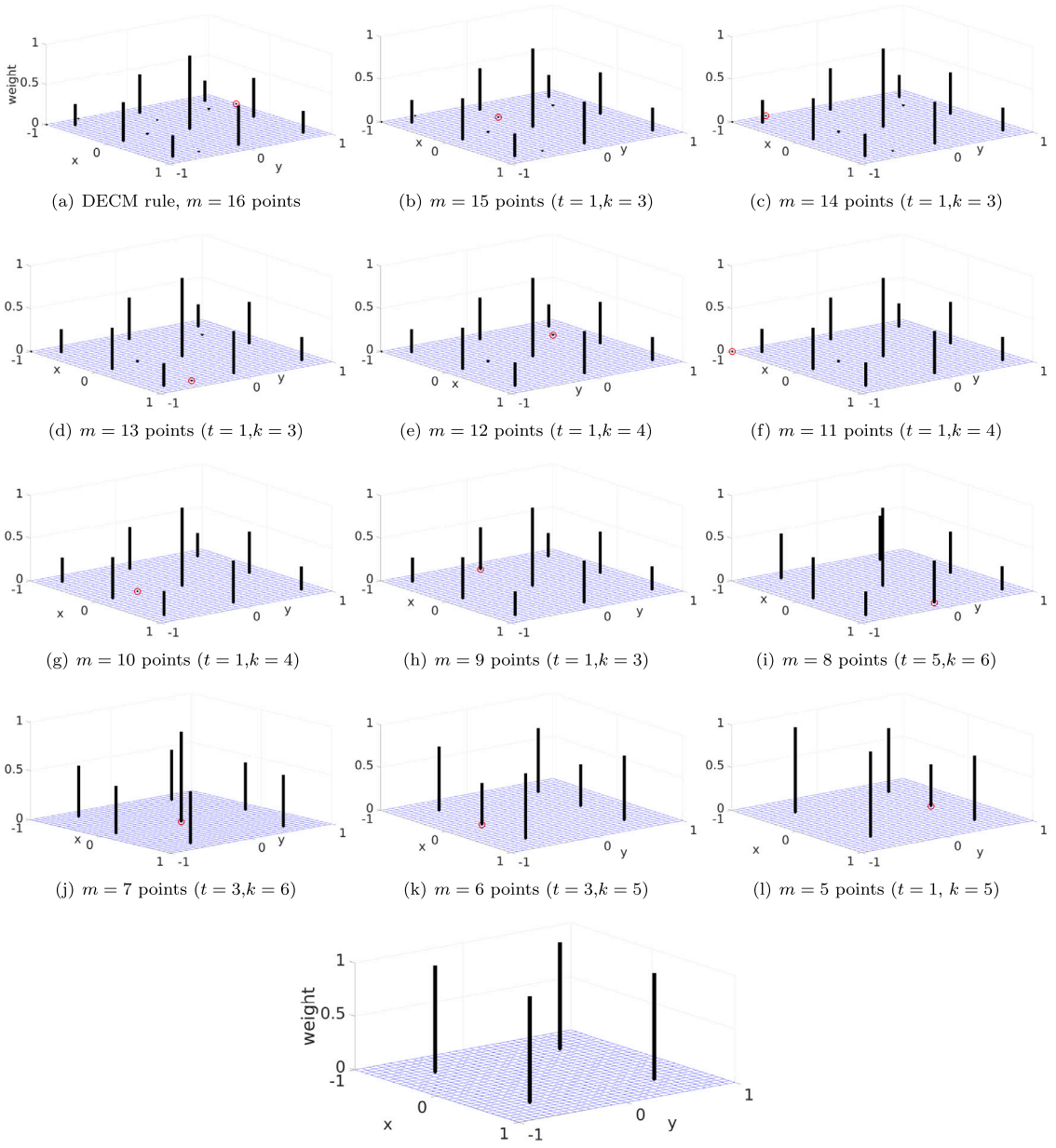
[12] A procedure for determining Gauss rules of arbitrary number of points is given in Ref. [27], page 86.

Table 2
Cubature rules computed by the CECM for bivariate polynomials of degree up to $p = 7$ (for both variables) in $\Omega = [-1,1] \times [-1,1]$. The rightmost column represents the relative deviations with respect to the optimal Gauss product rules (for polynomials of odd degree).

d	Positions		Weights	Error (Gs.)	d	Positions		Weights	Error (Gs.)
	x	y				x	y		
1	0.000000000	−0.000000000	4.000000000	1.1104e−15	6	0.835410647	0.241939775	0.295169258	
2	0.706474840	0.648819610	0.707815861			0.238387911	0.241939775	0.521300430	
	0.505956954	−0.513753481	1.262661689			0.839282581	−0.465525643	0.250410863	
	−0.471826192	0.648819610	1.059826914			0.255295202	−0.465525643	0.443797308	
	−0.658817575	−0.513753481	0.969695536			0.463842461	0.836215301	0.255076972	
3	0.577350269	0.577350269	1.000000000	2.0914e−15		0.909197293	0.836215301	0.100975680	
	0.577350269	−0.577350269	1.000000000			0.528704543	−0.910089745	0.157360912	
	−0.577350269	0.577350269	1.000000000			0.952103717	−0.910089745	0.044168747	
	−0.577350269	−0.577350269	1.000000000			−0.471295457	0.241939775	0.451263578	
4	0.235019870	0.887041325	0.346272588			−0.242986893	0.836215301	0.293371414	
	0.928559333	0.872658207	0.111090783			−0.444666035	−0.465525643	0.391121725	
	0.235019870	0.192355601	0.936139040			−0.205611449	−0.910089745	0.185331125	
	0.928559333	0.175016914	0.279848771			−0.913220650	0.241939775	0.173212013	
	−0.703200858	0.889364290	0.251544803			−0.899639238	−0.465525643	0.166229135	
	−0.703200858	0.195018638	0.687833785			−0.836453352	0.836215301	0.165982881	
	0.235019870	−0.713942525	0.682449449			−0.828149253	−0.910089745	0.105227960	
	0.928559333	−0.718476493	0.203099967		7	0.861136312	0.861136312	0.121002993	5.7779e−16
	−0.703200858	−0.713255984	0.501720814			0.339981044	0.861136312	0.226851852	
5	0.774596669	0.774596669	0.308641975	5.9957e−16		0.861136312	0.339981044	0.226851852	
	−0.000000000	0.774596669	0.493827160			0.339981044	0.339981044	0.425293303	
	0.774596669	−0.000000000	0.493827160			−0.339981044	0.861136312	0.226851852	
	0.000000000	−0.000000000	0.790123457			0.861136312	−0.339981044	0.226851852	
	−0.774596669	0.774596669	0.308641975			−0.339981044	0.339981044	0.425293303	
	0.774596669	−0.774596669	0.308641975			0.339981044	−0.339981044	0.425293303	
	−0.774596669	0.000000000	0.493827160			−0.861136312	0.861136312	0.121002993	
	−0.000000000	−0.774596669	0.493827160			−0.339981044	−0.339981044	0.425293303	
	−0.774596669	−0.774596669	0.308641975			0.861136312	−0.861136312	0.121002993	
						−0.861136312	0.339981044	0.226851852	
						0.339981044	−0.861136312	0.226851852	
						−0.861136312	−0.339981044	0.226851852	
						−0.339981044	−0.861136312	0.226851852	
						−0.861136312	−0.861136312	0.121002993	

We use structured meshes of 20×20 quadrilateral elements for the square, and $20 \times 20 \times 20$ hexahedra elements for the cube, with Gauss integration rules for each element of 2×2 and $2 \times 2 \times 2$ points, respectively. The parameters governing the performance of the CECM are the same employed in the univariate case. We examined the rules computed by the CECM for degrees up to $p = 12$ for both 2D and 3D cases (the same degree for all variables). For reasons of space limitation, we only display the coordinates and weights up to $p = 7$ for the 2D case (see Table 2), and in Table 3 up to $p = 4$ for the 3D case. The comparison with the product Gauss rules contained in both tables reveals the very same pattern observed in the case of univariate polynomials: for even degrees, the CECM produces asymmetrical rules, whereas for odd degrees, the CECM produces symmetrical rules identical to the corresponding product Gauss rules (featuring in both cases $(p+1)^d/2^d$ points, where $d = 2, 3$). Although not shown here, the same trend was observed for the remaining polynomial degrees.

These results provide further confirmation of the ability of the proposed sparsification algorithm to arrive at the *integration rules with minimal number of points*. Figs. 6 and 7 depict the sparsification process for the case $p = 3$ in 2D and 3D, respectively. As done previously in Fig. 5 for the univariate case, we show in each graph the number of trials required to find the point whose weight is to be zeroed (i.e., the number of times the method passes over the loop in line 7 of Algorithm 3), as well as the number of iterations required for zeroing the chosen weight (in the modified Newton–Raphson scheme of Algorithm 5). Whereas in the case of univariate polynomials, displayed previously in Fig. 5, the method succesfully determines the weights to be zeroed on the first trial, in the multivariate case several trials are necessary in some cases, especially when the algorithm approaches the optimum. For instance, to produce the rule with 8 points in the bivariate case, see Fig. 6(i), the algorithm tries $t = 5$ different points until finding



(a) DECM rule, $m = 16$ points (b) $m = 15$ points $(t = 1, k = 3)$ (c) $m = 14$ points $(t = 1, k = 3)$

(d) $m = 13$ points $(t = 1, k = 3)$ (e) $m = 12$ points $(t = 1, k = 4)$ (f) $m = 11$ points $(t = 1, k = 4)$

(g) $m = 10$ points $(t = 1, k = 4)$ (h) $m = 9$ points $(t = 1, k = 3)$ (i) $m = 8$ points $(t = 5, k = 6)$

(j) $m = 7$ points $(t = 3, k = 6)$ (k) $m = 6$ points $(t = 3, k = 5)$ (l) $m = 5$ points $(t = 1,\ k = 5)$

(m) Final CECM rule, $m = 4$ points $(t = 1, k = 5)$

Fig. 6. Locations and weights of the cubature rules generated during the sparsification process for the case of bivariate polynomials of degree $p = 3$ in $\Omega = [-1, 1] \times [-1, 1]$. The red circle in each graph indicates the point whose weight is to be zeroed in the following step. Variables t and k, on the other hand, have the same interpretation as in Fig. 5. The initial DECM rule has $(p + 1)^2 = 16$ points (see Fig. 6(a)), while the final rule features $(p + 1)^2/2^2 = 4$ points, see graph 6(l). The exact values of the coordinates and weights are given in Table 2 (it can be seen that the CECM rule coincides with the standard 2×2 product Gauss rule). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the appropriate combination. Closer examination of the causes for this increase of iterative effort indicates that the most common cause is the violation of the constraint that the points must remain within the domain.

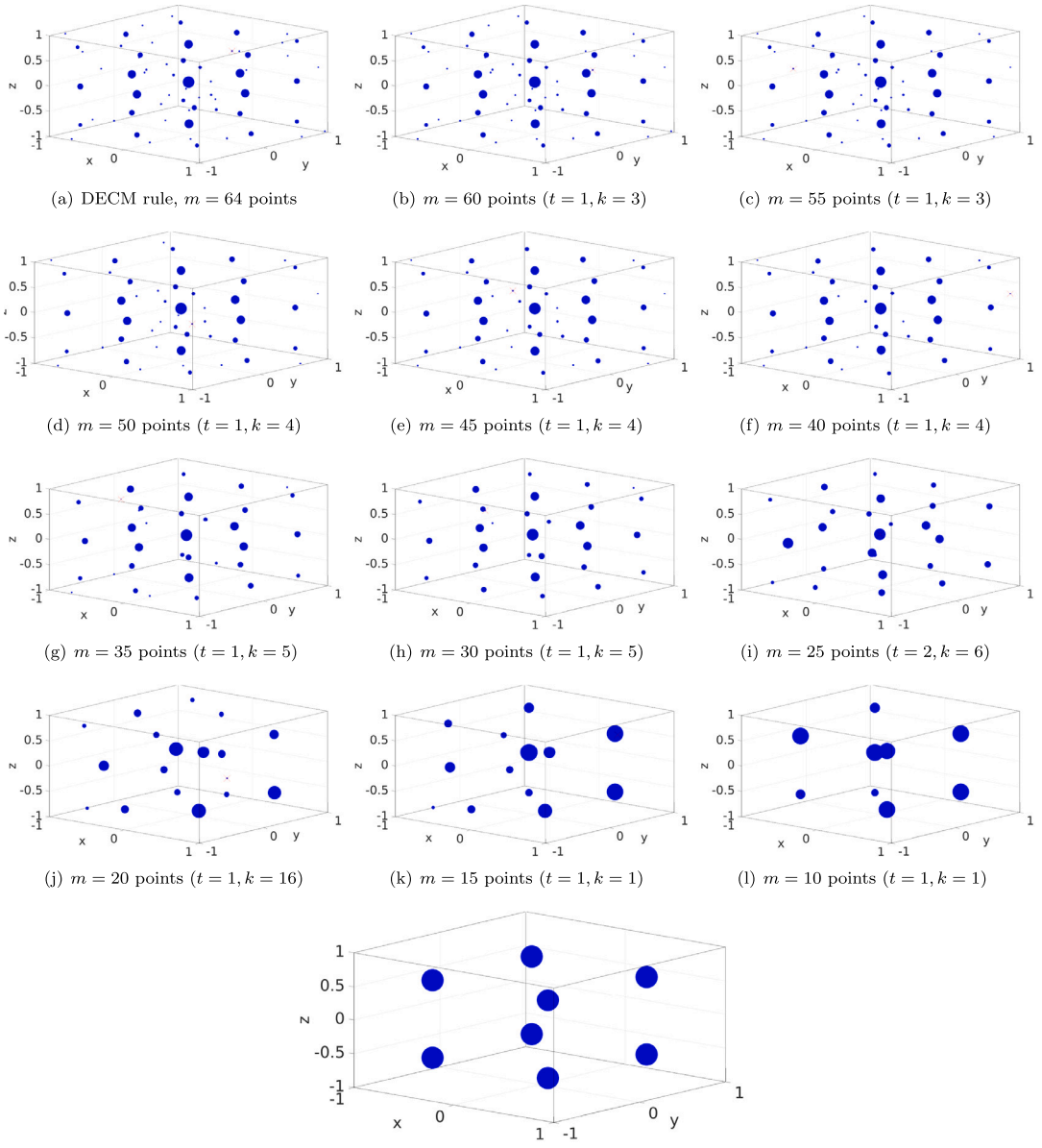
6.3. Exponential–sinusoidal function

We next study the derivation of a cubature rule for the following parameterized, vector-valued function:

$$\begin{aligned} &\boldsymbol{a} : \Omega \times \mathcal{D} \to \mathbb{R}^6, \\ &\quad (\boldsymbol{x}, \boldsymbol{\mu}) \mapsto (a_1, a_2, \ldots, a_6) \end{aligned} \tag{70}$$

Table 3
Cubature rules computed by the CECM for trivariate polynomials of degree up to $p = 4$ in $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$. The rightmost column represents the relative deviations with respect to the optimal Gauss product rules (for polynomials of odd degree).

d	Positions			Weights	Error (Gs.)
	x	y	z		
1	8.38519099255208E−17	−1.37013188830199E−16	2.54794197642291E−16	8.00000000000022	2.7534e−14
2	0.743706246140823	0.588904961511756	0.484154086744496	0.86563029280071	
	0.743706246140821	−0.566022287327392	0.831399263412485	0.499063612121599	
	−0.44820563907193	0.741835057873505	0.830045419614278	0.613961492138332	
	0.743706246140824	0.588904961511757	−0.688486047024208	0.608724673042735	
	0.743706246140825	−0.56602228732739	−0.400930513175059	1.03489533941434	
	−0.44820563907193	−0.449336183017346	0.830045419614395	1.01362448933892	
	−0.448205639071929	0.739388192080586	−0.401584450027124	1.274239461537	
	−0.448205639071928	−0.450823176382297	−0.401584450027067	2.08986063960634	
3	0.577350269189625	0.577350269189626	0.577350269189626	0.999999999999999	4.3425e−16
	0.577350269189626	0.577350269189626	−0.577350269189626	0.999999999999999	
	−0.577350269189626	0.577350269189626	0.577350269189626	1	
	−0.577350269189626	0.577350269189626	−0.577350269189626	1	
	0.577350269189626	−0.577350269189626	0.577350269189626	1	
	0.577350269189626	−0.577350269189626	−0.577350269189626	1	
	−0.577350269189626	−0.577350269189626	0.577350269189626	1	
	−0.577350269189626	−0.577350269189626	−0.577350269189626	1	
4	0.691575606960029	0.160521574170478	−0.261954268935127	0.701290700192769	
	−0.282918324688049	−0.217978724118607	−0.27678081615893	1.00337289978755	
	0.691575606960029	−0.282053603748311	0.696630178723833	0.561750852289478	
	−0.28291832468805	−0.211998475048984	0.69305243630094	0.733763793522678	
	−0.282918324688049	0.707430491792361	−0.27678081615893	0.734672939267592	
	0.691575606960029	−0.722360355870564	−0.261954268935126	0.50651426374409	
	0.691575606960028	0.691783591238766	0.696630178723832	0.412586162346216	
	0.691575606960028	0.861535244017794	−0.261954268935126	0.294025738847668	
	−0.28291832468805	0.708932471043466	0.693052436300939	0.536831048244394	
	0.691575606960028	0.067249317677481	−0.960515811309942	0.175016341839065	
	−0.282918324688048	−0.910801772049135	−0.27678081615893	0.330495727456156	
	−0.282918324688049	0.26535993911279	−0.980548984419782	0.246322371421189	
	−0.28291832468805	−0.904960687303036	0.693052436300938	0.248651857974508	
	0.691575606960028	−0.988158764971828	0.696630178723831	0.128498060521655	
	−0.989432805330447	−0.053487256448397	−0.23226227865251	0.203195059703459	
	0.691575606960029	0.804707428230073	−0.960515811309941	0.096280486097567	
	0.691575606960028	−0.750168356265182	−0.960515811309941	0.118969763496464	
	−0.989432805330449	−0.053487256448396	0.703880237838895	0.148998882881412	
	−0.28291832468805	−0.695806658208985	−0.980548984419782	0.180892299291507	
	−0.989432805330447	0.754804289030939	0.072783264075214	0.124784457736787	
	−0.282918324688049	0.96494897614471	−0.980548984419783	0.062699664209236	
	−0.989432805330446	−0.797964254050424	−0.226012314072442	0.11467159716885	
	−0.989432805330448	0.754804289030939	0.807516321196852	0.067775700765236	
	−0.989432805330447	−0.053487256448394	−0.925565909634688	0.06232167903138	
	−0.989432805330448	−0.797964254050423	0.705427653813099	0.084038633103628	
	−0.989432805330448	0.754804289030939	−0.748349786829047	0.085270559050749	
	−0.989432805330448	−0.797964254050424	−0.918958907578276	0.036308460008693	



(a) DECM rule, $m = 64$ points (b) $m = 60$ points $(t = 1, k = 3)$ (c) $m = 55$ points $(t = 1, k = 3)$

(d) $m = 50$ points $(t = 1, k = 4)$ (e) $m = 45$ points $(t = 1, k = 4)$ (f) $m = 40$ points $(t = 1, k = 4)$

(g) $m = 35$ points $(t = 1, k = 5)$ (h) $m = 30$ points $(t = 1, k = 5)$ (i) $m = 25$ points $(t = 2, k = 6)$

(j) $m = 20$ points $(t = 1, k = 16)$ (k) $m = 15$ points $(t = 1, k = 1)$ (l) $m = 10$ points $(t = 1, k = 1)$

(m) Final CECM rule: $m = 8$ points $(t = 2, k = 13)$

Fig. 7. Locations and weights of the cubature rules generated during the sparsification process for the case of trivariate polynomials of degree $p = 3$ in $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$. Variables t and k have the same interpretation as in Figs. 5 and 6. The initial DECM rule has $(p+1)^3 = 64$ points (see Fig. 7(a)), and the final rule $(p+1)^3/2^3 = 8$ points, see graph 7(m). The exact values of the coordinates and weights are given in Table 3, wherein it can be seen that the computed rule does coincide with the standard $2 \times 2 \times 2$ product Gauss rule.

where

$$\Omega = [-1, 1] \times [-1, 1] \times [-1, 1], \qquad \text{(spatial domain)} \tag{71}$$

$$\mathcal{D} = [1, \pi] \times [1, \pi], \qquad \text{(parametric domain)} \tag{72}$$

$$\begin{aligned} a_1 &= B(x_1)C(x_1, \mu_1)E(x_1, \mu_1) + 1 \\ a_2 &= B(x_2)C(x_2, \mu_1)E(x_2, \mu_1) + 1 \\ a_3 &= B(x_1)C(x_1, \mu_1)E(x_2, \mu_1) + 1 \\ a_4 &= B(x_2)C(x_2, \mu_1)E(x_1, \mu_1) + 1 \\ a_5 &= B(x_1)C(x_1, \mu_1)E(x_3, \mu_2) + 1 \\ a_6 &= B(x_3)C(x_3, \mu_2)E(x_2, \mu_1) + 1 \end{aligned} \tag{73}$$

Table 4
Comparison of the performance of the proposed Sequential Randomized SVD (see Algorithm 6 in Appendix) with respect to the standard SVD in determining an approximate orthogonal basis matrix (truncation tolerance $\epsilon_{svd} = 10^{-4}$) for the column space of the integrand matrix of the vector-valued function (6 components) defined in Eq. (70). The number of spatial integration points is $M = (30 \cdot 3)^3 = 729000$, and the function is sampled at the points of uniform grids in parameter space of varying size ($n_{samp} \times n_{samp}$, see first column). The number of columns of the integrand matrix is therefore $n_{col} = 6n_{samp}^2$, and its size (in gigabytes) equal to $8 \cdot 10^{-9} n_{col} M$. For the matrix of 33.63 GB (last row) there is no information on either the computing time nor the rank (number of basis vectors) for the standard truncated SVD (using the builtin Matlab function `svd`, see Algorithm 7 in Appendix), because the computation exhausted the memory capabilities of the employed 64 GB RAM computer. N_{part} denotes the number of partitions of the integrand matrix in the case of the SRSVD, and N_{iter}^{avg} the average number of iterations employed by the incremental randomized orthogonalization (see Algorithm 10 in Appendix) for all the partitions. The rightmost column represents the relative difference between the singular values computed by both methods ($\|\boldsymbol{S}_{svd} - \boldsymbol{S}_{srsvd}\|_2 / \|\boldsymbol{S}_{svd}\|_2$).

n_{samp}	n_{col}	Size (GB)	SVD		SRSVD				ERROR SING. VAL.
			Time (s)	Rank	N_{part}	N_{iter}^{avg}	Time (s)	Rank	
4	96	0.56	2.1	36	1	3	2.5	36	5.05E−15
6	216	1.26	5.4	53	1	3	5.9	53	5.26E−15
8	384	2.24	11.4	70	1	2	6.6	70	4.86E−15
11	726	4.23	28.2	90	3	1.67	11.9	90	5.93E−14
16	1536	8.96	84.7	122	4	1.67	21.3	122	3.66E−14
22	2904	16.94	234.0	131	9	1.22	35.3	131	2.62E−13
31	5766	33.63	*	*	16	1.06	65.2	133	*

and

$$B(r) = 1 - r, \qquad C(r,s) = \cos 3\pi s(r+1), \qquad E(r,s) = e^{(-1+r)s}. \tag{74}$$

We use a structured spatial mesh of 30 × 30 × 30 hexahedra elements, each element being equipped with a product Gauss rule of 3 × 3 × 3 points. Unlike the case of polynomials discussed in the foregoing, where we knew beforehand which was the space of functions to be integrated – the SVD only played a secondary, orthogonalizing role therein –, in this problem we have to delineate first the space in which the integrand lives. This task naturally confronts us with the question of how dense should be the sampling of the parametric space so that the column space of the corresponding integrand matrix $\boldsymbol{A}_{FE}$ becomes representative of this linear space. We address here this question by gradually increasing the number of sampled points in parametric space, applying the SVD with a fixed user-prescribed truncation tolerance to the corresponding integrand matrix (here we use $\epsilon_{SVD} = 10^{-4}$), and then examining when the rank of the approximation (number of retained singular values) appears to converge to a maximum value. Since there are only two parameters here, it is computationally affordable[13] to conduct this exploration by uniformly sampling the parametric space.

We show in Table 4 the result of this convergence study using both the standard SVD and the proposed Sequential Randomized SVD (described in Appendix). The study has been devised so that the size of the integrand matrix doubles at each refinement step. Likewise, the block partition of the integrand matrix in the case of the SRSVD has been taken so that the size of each block matrix is approximately 2 GB. This convergence study reveals that the dimension of the linear space in which the integrand lies (for the prescribed tolerance) is around 130. The study also serves to highlight the advantages of the proposed SRSVD in terms of both computing time and memory requirements: for the matrix of size 16.96 GB, the SRSVD is almost 7 times faster than the SVD, and for the largest matrix of 33.63 GB, the standard SVD cannot handle the operation because it exhausts the memory capabilities of the employed computer (which has 64 GB RAM[14]); the SRSVD, by contrast, returns the result in approximately 1 min. The reasons of this clear outperformance of the SRSVD over the SVD are further discussed in Appendix A.4 of Appendix.

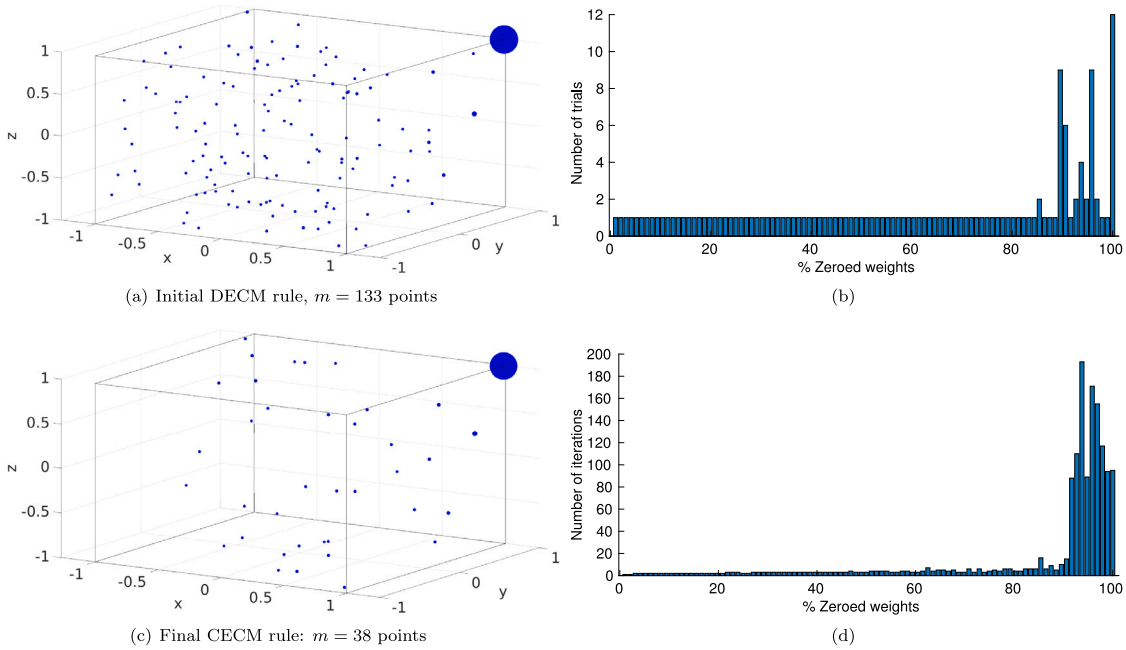
Fig. 8(a) shows the DECM rule determined using the 133 left singular vectors provided by the SRSVD for the parametric grid of 31 × 31 points (see last row of Table 4), while Fig. 8(c) displays the final 38-points CECM rule obtained after the sparsification process—the reduction factor is approximately 3.4. The variables controlling the sparsification are the same employed in the polynomial case. Further information about the sparsification process are displayed in Figs. 8(b) and 8(d). The number of trials taken by the algorithm to find the weight to be zeroed (versus the percentage of zeroed weights) is shown in Fig. 8(b), whereas Fig. 8(d) represents the total number of accumulated nonlinear iterations (also versus the percentage of zeroed weights). It can be seen in Fig. 8(b) that approximately 90% of the weights are zeroed on the first trial; it is only in the last 10% that the number of trials increases considerably. The same behavior is observed in terms of accumulated nonlinear iterations in Fig. 8(d): while the first 90% of the points are zeroed in 5 iterations on average, for the last 10% of points, the number of iterations required for this very task raises sharply (close to 200 iterations in some cases). This is not only due to the increase of the number of attempts to zeroed the weights reflected in Fig. 8(b), but also because, at this juncture of the sparsification process, the weights of the points are relatively large and, to ensure converge, the problem of driving the integration residual to zero is solved in more than one step[15] (we take here $N_{steps} = 20$).

Obviously, this uneven distribution of iterations during the sparsification process translated into an equally uneven computing time distribution: zeroing the first 90% of weights took less than 1 min, whilst the remaining 10% required about 8 min.

[13] Higher parameter dimensions may require more sophisticated sampling strategies, such as the greedy adaptive procedure advocated in Ref. [38] for reduced-order modeling purposes.

[14] The code is implemented in Matlab, and executed in an Intel(R) Core(TM) i7-8700 CPU, 3.20 GHz with 64 Gb RAM (Linux platform).

[15] The sparsification process enters the second stage in line 5 of Algorithm 1.



(a) Initial DECM rule, $m = 133$ points (b)

(c) Final CECM rule: $m = 38$ points (d)

Fig. 8. Cubature of the parameterized function defined in Eq. (70) in $\Omega = [-1, 1] \times [-1, 1] \times [-1, 1]$. The orthogonal basis vectors are determined by the SRSVD using a parametric grid of 31 × 31 points (see last row of Table 3). (a) Initial interpolatory DECM rule. (b) Number of passes over the loop that selects the weights to be zeroed in Algorithm 3 (see line 7) as a function of the percentage of zeroed weights. (c) Final CECM rule. (d) Total number of iterations of the modified Newton–Raphson scheme in Algorithm 5 as a function of the percentage of zeroed weights.

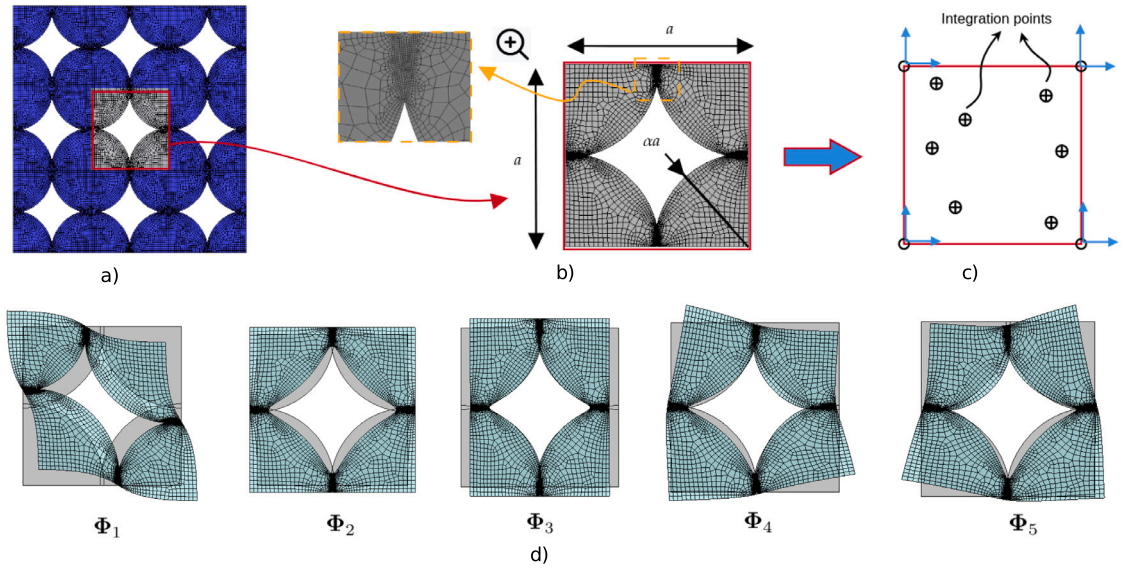


Fig. 9. Assessment of the performance of the CECM in the hyperreduction of multiscale finite elements. (a) Periodic structure under study. (b) Fine-scale mesh of the unit cell for which the coarse-scale representation is required ($a = 0.195$ m, $\alpha = 0.5135$). The total number of Gauss points is $M = 42471$. (c) Coarse-scale representation of the unit cell, possessing 8 degrees of freedom and a number of integration points to be determined by the CECM. (d) Deformational modes of the unit cell. The integral to be tackled by the CECM is the projection of the fine-scale nodal internal forces onto the span of these modes (see Eq. (75)).

6.4. Hyperreduction of multiscale finite element models

We conclude the validation of the proposed CECM by illustrating its use in the hyperreduction of finite element models. More specifically, attention is concentrated in the derivation of low-dimensional surrogate models in the context of multiscale finite element methods. The employed multiscale methodology is the *Empirical Interscale FE* (EIFE) method developed by the first author

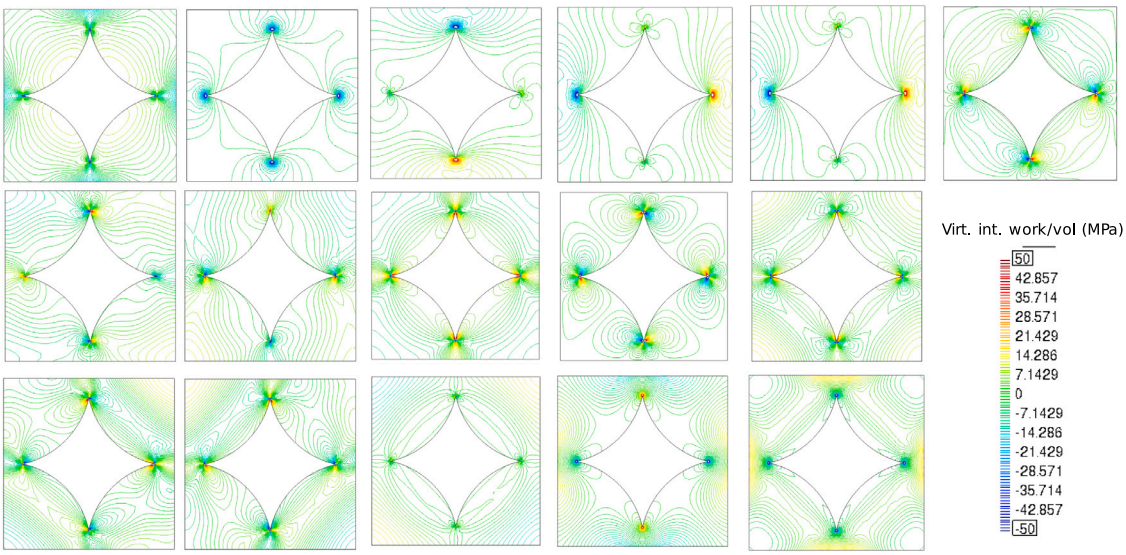


Fig. 10. Contour lines corresponding to the 16 basis functions for the integrand in Eq. (75), which represents the virtual work per unit volume (for the case in which the reduced basis $\boldsymbol{\Phi}$ is formed by the $n = 5$ modes shown previously in Fig. 9.d).

and co-worker in Refs. [39] (for beam elements) and Ref. [40] (for solid elements). The chosen example is the modeling of the periodic structure displayed in Fig. 9.a, made of tiling copies of the unit cell displayed in turn in Fig. 9.b. The FE mesh of this unit cell, also depicted in Fig. 9.b, is formed by 19,356 nodes, and $N_{el} = 4719$ quadratic quadrilateral elements, with $r = 9$ Gauss points per element (hence $M = N_{el}r = 42471$). The areas in which stress concentration are likely to appear are densely meshed to avoid both highly distorted elements and pronounced interelemental jumps—both issues are detrimental, not only to the accuracy of the FE analysis per se, but also to the accuracy of the final cubature rule. The goal in the EIFE method is to replace the fine-scale representation of the unit cell in Fig. 9.b by a surrogate coarse-scale element such as the one shown in Fig. 9.c. The coarsening process involves two sequential stages; the first stage involves the reduction of number of degrees of freedom (DOFs), and the second stage involves an additional reduction in complexity in terms of number of integration points for each term of the governing equations (this is why the second stage is called "hyperreduction"). The particular details on how the first reduction of number of DOFs is carried out are of no concern here—the reader interested in such details is referred to Ref. [40]. It suffices to say that the process involves running firstly FE analyses in domains comprising several cells (such as the one in Fig. 9.a) under appropriate prescribed displacements, and then extracting characteristic deformational patterns of the unit cell in the center via the SVD. In the case studied here, in which the coarse model has 8 DOFs and the FE analyses are conducted, for simplicity of exposition,[16] in the linear elastic regime (Young's Modulus $E = 70000$ MPa and Poisson's ratio $\nu = 0.3$, in plane strain), the resulting deformational patterns are the $n = 5$ modes whose deformed shapes are displayed in Fig. 9.d.

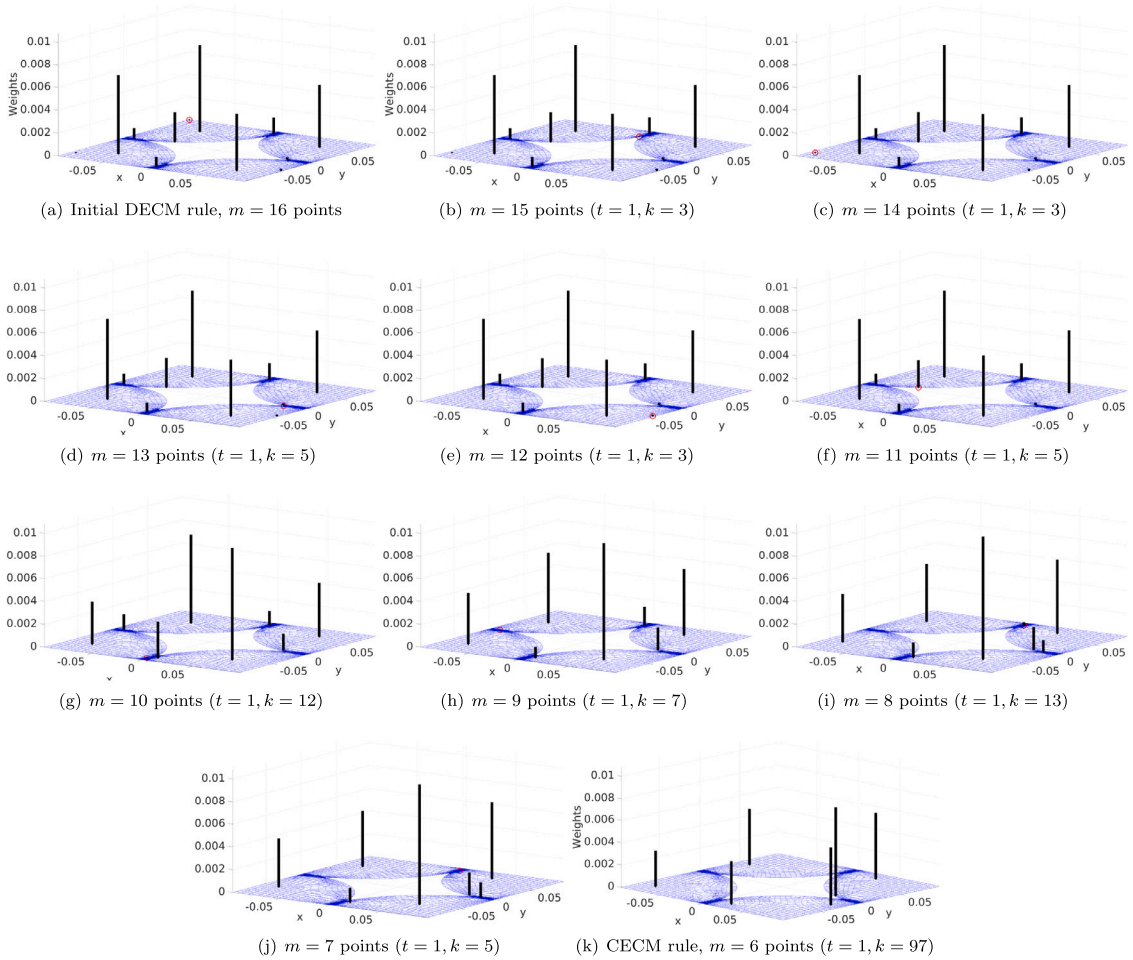
Our interest lies in the hyperreduction stage, in which one has to devise efficient cubature rules for each one of the integrals appearing in the (reduced) governing equations. We focus here in the internal force term, but a similar procedure can be followed for the integrals associated to body forces and surface traction. The reduced internal forces are given by the projection onto the space spanned by the deformational modes $\boldsymbol{\Phi}$ of the FE nodal internal forces $\boldsymbol{F}_{int}$:

$$\boldsymbol{\Phi}^T \boldsymbol{F}_{int} = \int_\Omega \boldsymbol{\Phi}^T \boldsymbol{B}_{FE}^T \boldsymbol{\sigma} \, d\Omega. \tag{75}$$

Here, $\boldsymbol{B}_{FE}$ denotes the standard strain–displacement FE matrix (in its globally supported format), whereas $\boldsymbol{\sigma}$ is the stress vector. Our parametric integrand is therefore equal to $\boldsymbol{a} = (\boldsymbol{B}_{FE}\boldsymbol{\Phi})^T\boldsymbol{\sigma}$, the work per unit volume done over virtual strains of the form $\boldsymbol{B}_{FE}\boldsymbol{\Phi}_i$ $(i = 1, 2 \dots n)$ by the stresses $\boldsymbol{\sigma}$ caused in turn by strains also of the form $\boldsymbol{\epsilon} = (\boldsymbol{B}_{FE}\boldsymbol{\Phi})\boldsymbol{q}$ (Galerkin projection). It follows then that $\boldsymbol{\mu} = \boldsymbol{q}$, that is, in this problem, the amplitude of the deformational modes in the expression for the stresses plays the role of input parameters. Since we are assuming linear elastic behavior, the number of possible stress states is equal to $n = 5$ as well. This implies that the integrand matrix $\boldsymbol{A}_{FE}$ is formed by $n^2 = 25$ columns, which are all the possible combinations of stress modes times virtual strain modes. The SVD (step 4 in Box 5.1) of this matrix (using a fairly low truncation tolerance, $\epsilon_{svd} = 10^{-10}$, to eliminate numerical errors) reveals that there are redundant work modes: out of the $n^2 = 25$ work modes, only 15 are linearly independent.[17]

[16] The procedure can be applied to any constitutive stress–strain law.

[17] It can be readily seen that the existence of these 10 redundant work patterns is nothing but the consequence of the symmetry of the elastic problem—Betti's reciprocity theorem [41]. In general, if there are n deformational modes, the number of independent work modes will be equal to $(n+1)n/2$.



(a) Initial DECM rule, $m = 16$ points (b) $m = 15$ points $(t = 1, k = 3)$ (c) $m = 14$ points $(t = 1, k = 3)$

(d) $m = 13$ points $(t = 1, k = 5)$ (e) $m = 12$ points $(t = 1, k = 3)$ (f) $m = 11$ points $(t = 1, k = 5)$

(g) $m = 10$ points $(t = 1, k = 12)$ (h) $m = 9$ points $(t = 1, k = 7)$ (i) $m = 8$ points $(t = 1, k = 13)$

(j) $m = 7$ points $(t = 1, k = 5)$ (k) CECM rule, $m = 6$ points $(t = 1, k = 97)$

Fig. 11. Locations and weights of the cubature rules generated during the sparsification process for the case of the virtual internal work per unit volume in Eq. (75) (integrand functions shown previously in Fig. 10). The red circle in each graph indicates the point whose weight is to be zeroed in the following step. Variables t and k, on the other hand, have the same interpretation as in Fig. 5. The initial DECM rule has 16 points (see Fig. 11(a)), while the final rule features $m = 6$ points, see Fig. 11(k). The total reduction in number of integration points is $M/m = 42471/6 = 7078.5$. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

By augmenting the integrand basis with an additional vector for accounting for the integration of the volume (step 5 in Box 5.1), we end up with 16 basis functions for the integrand; the contour lines of these 16 functions are depicted in Fig. 10.

The 16-points interpolatory DECM rule corresponding to these 16 functions is displayed in turn in Fig. 11.a. The fact that the weights of 8 of the 16 points are comparatively small gives an indication that the number of points can be further reduced by the proposed sparsification process. The result of this sparsification process is displayed in Figs. 11.b to 11.k. The number of trials and accumulated nonlinear iterations are also shown in the captions of these Figures. As observed with the 3D analytical function in Section 6.3, the number of iterations required for the last 10% of the weights (which in this case is just the last step) requires significantly more iterations than the previous 90%. Nevertheless, the total computing time is not significant—less than 6 s for the entire procedure, including the computation of the DECM rule. To assess the final integration error, we compare the reduced stiffness matrix

$$\boldsymbol{\Phi}^T \boldsymbol{K} \boldsymbol{\Phi} = \int_{\Omega} \boldsymbol{\Phi}^T (\boldsymbol{B}_{FE}^T \boldsymbol{C} \boldsymbol{B}_{FE}) \boldsymbol{\Phi}^T \, d\Omega. \tag{76}$$

(here $\boldsymbol{K}$ is the nodal stiffness matrix of the unit cell, and $\boldsymbol{C}$ the corresponding elasticity matrix) computed by both the original 42471-points Gauss element rule and the 6-points CECM rule. The difference turns out to be below 0.005%. This result demonstrates that, one the one hand, that the computed 6-points CECM effectively encodes the physics of the coarsened unit cell, and on the other hand, that the proposed cubature algorithm is able to deal with complex domains in which the integrand function is only defined at some points of such domain—the FE Gauss points. In fact, it should be remarked that, since the tolerance employed in the SVD is negligible, this small error is to be exclusively attributed to the fitting procedure outlined in Section 2.3.2 for constructing approximations of the integrand at element level .

The assessment of the CECM-based hyperreduction has been carried out in the linear regime for simplicity. However, it is important to note that the true potential of hyperreduction becomes apparent when the integrand functions exhibits a nonlinear relationships with the unknowns. The reader interested in the application of the proposed CECM-hyperreduction to nonlinear scenarios is referred to the recent work by the first author [40], in which the methodology is applied to predict plastic yielding in multiscale problems.

7. Conclusions

In this paper we presented the Continuous Empirical Cubature Method (CECM), a novel algorithm designed to enhance the efficiency of numerical integration rules. The CECM is a two-stage algorithm whose first stage consists on the application of a point selection strategy for obtaining an interpolatory rule—featuring as many points as functions to integrate. We have used for this purpose the Discrete Empirical Cubature Method (DECM) [2,16]. Then, for the second stage, we have applied a sparsification strategy whose aim is rendering to zero the associated weight of as many points as possible. To this end, the locations of the initially selected points were changed following a modified Newton–Raphson algorithm whose details have been outlined within the text (The code has also been made available at https://github.com/Rbravo555/CECM-continuous-empirical-cubature-method).

The versatility of the method was highlighted in the numerical assessment section, showcasing its effectiveness across a diverse range of problems. For the case of univariate and multivariate Lagrange polynomials, the CECM was able to recover the optimal Gaussian rule whenever the number of function to integrate was odd. For even number of functions to integrate, the number of points still coincides with the Gauss rule, although their locations differ. Reductions in required points were observed, with 1D domains requiring half, 2D domains requiring one fourth, and 3D domains requiring one eighth of the initial interpolatory rule.

Section 6.3 showcases an exponential-sinusoidal function in a 3D domain. For this case, the CECM reduced the number of points from 133 of the original DECM rule, to 38. This example also showcased an scenario where the size of the matrices involved would render infeasible the computation of the SVD on a regular desktop computer. The SVD was computed using the sequential randomized SVD (SRSVD) algorithm presented also in this paper. The SRSVD allowed to efficiently compute the required orthogonal basis for matrices of up to 33 GB in size (in a computer with a 64 GB RAM, in about 1 min).

Finally, the CECM was applied to an empirical interscale finite element (EIFE) example, for which the number of points was reduced from the original fine scale 42,471 points to a 6-point rule, while incurring a negligible integration error of 0.005%.

Overall, the CECM algorithm presents an innovative approach to generating optimal integration rules. Its incorporation into frameworks that allow input of quadrature points' positions leads to substantial improvements in integral evaluation performance compared to standard interpolatory rules.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is sponsored in part by the Spanish Ministry of Economy and Competitiveness, through the *Severo Ochoa Programme for Centres of Excellence in* R&D (CEX2018-000797-S)". The authors acknowledge the support of the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955558 (the JU receives, in turn, support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway), as well as the R&D project PCI2021-121944, financed by MCIN/AEI/10.13039/501100011033 and by the "European Union NextGenerationEU/PRTR". J.A. Hernández expresses gratitude by the support of, on the one hand, the "MCIN/AEI/10.13039/501100011033/y *por FEDER una manera de hacer Europa*" (PID2021-122518OB-I00), and, on the other hand, the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 952966 (project FIBREGY). Lastly, both J.R. Bravo and S. Ares de Parga acknowledge the *Departament de Recerca i Universitats de la Generalitat de Catalunya* for the financial support through doctoral grants FI-SDUR 2020 and FI SDUR-2021, respectively.

Appendix. Sequential randomized SVD

A.1. Overview

In this Appendix we explain and provide the implementational details (see Algorithm 6) of the proposed procedure for computing the SVD of a partitioned matrix $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2 \dots \boldsymbol{A}_p]$ ($\boldsymbol{A}_i \in \mathbb{R}^{n\times m_i}$, with $m = \sum_{i=1}^{p} m_i$), alluded to in Remark 2.1. The method is based on the same idea behind other *randomized* algorithms [42], namely, that the SVD of a matrix $\boldsymbol{A}$ ($\boldsymbol{A} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$) can be alternatively computed from the matrices of the SVD of $\boldsymbol{L} = \boldsymbol{Q}^T\boldsymbol{A}$, $\boldsymbol{Q} \in \mathbb{R}^{n\times r}$ being an arbitrary orthogonal basis matrix for the column space of $\boldsymbol{A}$ (here $r \leq \min(n, m)$ denotes the rank of the matrix). More specifically, given $\boldsymbol{L} = \bar{\boldsymbol{U}}\bar{\boldsymbol{S}}\bar{\boldsymbol{V}}^T$, then $\boldsymbol{U} = \boldsymbol{Q}\bar{\boldsymbol{U}}$, $\boldsymbol{S} = \bar{\boldsymbol{S}}$ and $\bar{\boldsymbol{V}} = \boldsymbol{V}$. The proof of this property follows from expressing $\boldsymbol{A}$ as $\boldsymbol{A} = \boldsymbol{Q}(\boldsymbol{Q}^T\boldsymbol{A})$ and replacing $\boldsymbol{Q}^T\boldsymbol{A}$ by its SVD:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{Q}^T = (\boldsymbol{Q}\bar{\boldsymbol{U}})\bar{\boldsymbol{S}}\bar{\boldsymbol{V}}^T. \tag{77}$$

Both $\bar{\boldsymbol{S}}$ and $\bar{\boldsymbol{V}}$ arise from an SVD, and therefore, are diagonal with positive entries and orthogonal, respectively—and $\bar{\boldsymbol{V}}$ is a basis matrix for the row space of $\boldsymbol{L} = \boldsymbol{Q}^T\boldsymbol{A}$, which is the same as the row space of $\boldsymbol{A}$. Furthermore, $\boldsymbol{U}$ is also an orthogonal matrix:

$$\boldsymbol{U}^T\boldsymbol{U} = (\boldsymbol{Q}\bar{\boldsymbol{U}})^T(\boldsymbol{Q}\bar{\boldsymbol{U}}) = \bar{\boldsymbol{U}}^T(\boldsymbol{Q}^T\boldsymbol{Q})\bar{\boldsymbol{U}} = \boldsymbol{I}. \tag{78}$$

Therefore, it follows from the uniqueness of the SVD (up to the signs of the left- and right-singular vectors) that the factorization $\boldsymbol{U}\bar{\boldsymbol{S}}\bar{\boldsymbol{V}}^T$ is the SVD of $\boldsymbol{A}$, as asserted.

It can be readily shown that this property also holds for the case in which *truncation* is introduced. Besides, since $\boldsymbol{S} = \bar{\boldsymbol{S}}$, the truncation threshold for the SVD of $\boldsymbol{L}$ is the same as the input truncation threshold,[18] ϵ, that is,

$$\|\boldsymbol{L} - \bar{\boldsymbol{U}}\bar{\boldsymbol{S}}\bar{\boldsymbol{V}}^T\|_F \leq \epsilon\|\boldsymbol{L}\|_F = \epsilon\|\boldsymbol{A}\|_F. \tag{79}$$

This general strategy for computing the SVD of a matrix proves advantageous only when the following two conditions are met. Firstly, the rank of the matrix r should be significantly smaller[19] than the number of columns and rows of the matrix—because otherwise the SVD of $\boldsymbol{L} = \boldsymbol{Q}^T\boldsymbol{A}$ could become as costly as the original SVD. In the context of reduced-order models, this property is expected to hold—and if it does not, it means that the parameterized boundary value problem we intend to solve might not be amenable to dimensionality reduction. The other condition is that the computation of the orthogonal basis matrix $\boldsymbol{Q}$ should be efficient, in the sense that it should be carried out by an algorithm in which the asymptotic count of floating point operations (flops) is less than that required by the standard SVD itself.

Algorithm 6: Sequential Randomized Singular Value Decomposition (SRSVD) of a partitioned matrix $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_p]$.

1 **Function** $[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] \leftarrow \mathrm{SRSVD}([\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_p],\ \epsilon)$
  Data: $[\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_q]$: Partitioned matrix (with $\boldsymbol{A}_i \in \mathbb{R}^{n\times m_i}$, and $m = \sum_{i=1}^{p} m_i$). $\epsilon \in [0, 1]$: relative error threshold
  Result: Truncated Singular Value Decomposition (with relative truncation threshold ϵ) of $\boldsymbol{A} \in \mathbb{R}^{n\times m}$, i.e.: $\boldsymbol{A} \approx \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{n\times k}$ is the orthogonal matrix of left singular vectors, $\boldsymbol{S} \in \mathbb{R}^{k\times k}$ is the diagonal matrix of positive singular values, and $\boldsymbol{V} \in \mathbb{R}^{m\times k}$ is the orthogonal matrix of right singular vectors. Here $k \leq \min(n, m)$ denotes the number of retained singular vectors upon truncation, which is, by definition of SVD, the lowest number of vectors such that $\|\boldsymbol{A} - \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\|_F \leq \epsilon\|\boldsymbol{A}\|_F$.
2 $[\boldsymbol{Q}, \boldsymbol{L}] \leftarrow \mathrm{SRORTH}([\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_p])$ // Factorization $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{L}$, where $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$ (see Algorithm 8)
3 $\epsilon_L \leftarrow \epsilon\|\boldsymbol{L}\|_F$; $\mu_{mach} = \max(n, m)\,\mathrm{eps}(\|\boldsymbol{L}\|_F)$ // Truncation thresholds (see definition of eps in Algorithm 7).
4 $[\tilde{\boldsymbol{U}}, \boldsymbol{S}, \boldsymbol{V}] \leftarrow \mathrm{SVDT}(\boldsymbol{L}, \epsilon_L, \mu_{mach})$ // Truncated SVD (see Algorithm 7). .
5 $\boldsymbol{U} \leftarrow \boldsymbol{Q}\tilde{\boldsymbol{U}}$ // Matrix of left singular vectors

A.2. Sequential randomized orthogonalization

A.2.1. Infinite-precision arithmetic

To meet this latter condition, we propose to determine $\boldsymbol{Q}$ by the *Sequential Randomized Orthogonalization* (SRORTH) invoked in Line 2 of Algorithm 6, and with pseudo-code outlined in Algorithm 8. The qualifier *sequential* refers to the fact that the method only processes *one block matrix at a time*, thus alleviating potential memory bottlenecks. On the other hand, we call it *randomized* because one of the factorizations employed by the algorithm is carried out by a modified version of the *incremental randomized SVD* proposed in Ref. [34].

[18] This is not exactly true in the limiting case $\epsilon \to 0$, when the truncation criterion is established in terms of a machine-dependent precision parameter, which also depends on the size of the matrix being factorized (see Algorithm 6 Line 3) .

[19] This first condition may be relaxed by making $\boldsymbol{A} \approx \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}$, that is, by determining an approximated basis matrix for the column space of $\boldsymbol{A}$. However, it should be noticed that, in this case, the right-singular vector matrix $\boldsymbol{V}$ ceases to be a basis matrix for the row space of $\boldsymbol{A}$, and therefore, Eq. (30) would not hold.

Algorithm 7: Truncated Singular Value Decomposition (via standard, deterministic methods)

Function $[\boldsymbol{U},\boldsymbol{S},\boldsymbol{V}] \leftarrow \mathrm{SVDT}(\boldsymbol{C},\epsilon,\mu_{mach})$:
  Data: $\boldsymbol{C}\in\mathbb{R}^{m\times n}$. Absolute tolerance threshold $\epsilon>0$. Machine-precision parameter $\mu_{mach}>0$ (optional input, default value $\mu_{mach}=\max(n,m)\,\mathrm{eps}(\|\boldsymbol{C}\|_F)$, where $\mathrm{eps}(x)$ denotes the machine-dependent floating-point relative precision associated to x.
  Result: Truncated SVD such that $\|\boldsymbol{C}-\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T\|_F\le\epsilon$ (if $\epsilon\le\mu_{mach}$, the truncation criterion changes to the one specified in Line 4).
2 $[\boldsymbol{U},\boldsymbol{S},\boldsymbol{V}]\leftarrow \mathrm{svd}(\boldsymbol{C})$ // Thin SVD of $\boldsymbol{C}$ (In matlab's built-in function svd($\boldsymbol{C}$,'econ')
3 **if** $\epsilon\le\mu_{mach}$ **then**
4 $k\leftarrow$ Smallest singular value such that $\boldsymbol{S}(k,k)\ge\mu_{mach}$ // Numerical rank
5 **else**
6 $k\leftarrow$ Find smallest k such that $\sqrt{\sum_{i=k+1}^{n}\boldsymbol{S}(i,i)^2}\le\epsilon$ // Truncation level
7 **end**
8 $\boldsymbol{U}\leftarrow\boldsymbol{U}(:,1:k)$, $\boldsymbol{V}\leftarrow\boldsymbol{V}(:,1:k)$, $\boldsymbol{S}\leftarrow\boldsymbol{S}(1:k,1:k)$

Let us describe first the overall structure of this orthogonalization procedure, without delving into the randomized part of the algorithm, which will be treated later on, in Appendix A.3. In essence, the procedure is a Gram–Schmidt orthogonalization which operates, rather than on single vectors, on block matrices. Accordingly, $\boldsymbol{Q}$ is constructed as the concatenation of basis matrices (one for each block matrix $\boldsymbol{A}_i$):

$$\boldsymbol{Q}=\begin{bmatrix}\boldsymbol{\Delta Q}_1 & \boldsymbol{\Delta Q}_2 & \cdots & \boldsymbol{\Delta Q}_p\end{bmatrix} \tag{80}$$

where $\boldsymbol{\Delta Q}_i^T\boldsymbol{\Delta Q}_j=\boldsymbol{0}$ if $i\neq j$, and $\boldsymbol{\Delta Q}_i^T\boldsymbol{\Delta Q}_i=\boldsymbol{I}$ $(i=1,2\ldots p)$. In turn, these orthogonal submatrices are computed by the recursion

$$\begin{array}{lll}
\boldsymbol{\Delta Q}_1=\mathrm{ORTH}(\boldsymbol{A}_1); & \boldsymbol{Q}^{(1)}=\boldsymbol{\Delta Q}_1; & \boldsymbol{P}_1=\boldsymbol{Q}^{(1)^T}\boldsymbol{A}_1\\
\boldsymbol{\Delta Q}_2=\mathrm{ORTH}(\boldsymbol{A}_2-\boldsymbol{\Delta Q}_1\boldsymbol{\Delta Q}_1{}^T\boldsymbol{A}_2); & \boldsymbol{Q}^{(2)}=[\boldsymbol{Q}^{(1)},\boldsymbol{\Delta Q}_2]; & \boldsymbol{P}_2=\boldsymbol{Q}^{(2)^T}\boldsymbol{A}_2\\
 & \vdots\qquad\vdots & \\
\boldsymbol{\Delta Q}_i=\mathrm{ORTH}(\boldsymbol{A}_i-\boldsymbol{Q}^{(i-1)}\boldsymbol{Q}^{(i-1)^T}\boldsymbol{A}_i); & \boldsymbol{Q}^{(i)}=[\boldsymbol{Q}^{(i-1)},\boldsymbol{\Delta Q}_i]; & \boldsymbol{P}_i=\boldsymbol{Q}^{(i)^T}\boldsymbol{A}_i\\
 & \vdots\qquad\vdots & \\
\boldsymbol{\Delta Q}_p=\mathrm{ORTH}(\boldsymbol{A}_p-\boldsymbol{Q}^{(p-1)}\boldsymbol{Q}^{(p-1)^T}\boldsymbol{A}_p); & \boldsymbol{Q}=[\boldsymbol{Q},\boldsymbol{\Delta Q}_p]; & \boldsymbol{P}_p=\boldsymbol{Q}^T\boldsymbol{A}_p
\end{array} \tag{81}$$

(here $\mathrm{ORTH}(\bullet)$ symbolizes the function that determines an orthogonal basis matrix for the column space of its input). Notice that this procedure for determining $\boldsymbol{Q}$ need not store in the computer's main memory the entire matrix, but just one block matrix at a time, as asserted earlier. The other matrix in the factorization, $\boldsymbol{L}=\boldsymbol{Q}^T\boldsymbol{A}$, can be also constructed incrementally as the algorithm progresses.[20] Indeed, by exploiting that $\boldsymbol{\Delta Q}_j^T\boldsymbol{A}_i=\boldsymbol{0}$ if $j>i$, we have that

$$\boldsymbol{L}=\begin{bmatrix}\boldsymbol{\Delta Q}_1^T\boldsymbol{A}_1 & \boldsymbol{\Delta Q}_1^T\boldsymbol{A}_2 & \cdots & \boldsymbol{\Delta Q}_1^T\boldsymbol{A}_p\\ \boldsymbol{\Delta Q}_2^T\boldsymbol{A}_1 & \boldsymbol{\Delta Q}_2^T\boldsymbol{A}_2 & \cdots & \boldsymbol{\Delta Q}_2^T\boldsymbol{A}_p\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{\Delta Q}_p^T\boldsymbol{A}_1 & \boldsymbol{\Delta Q}_p^T\boldsymbol{A}_1 & \cdots & \boldsymbol{\Delta Q}_p^T\boldsymbol{A}_p\end{bmatrix}=\begin{bmatrix}\boldsymbol{\Delta Q}_1^T\boldsymbol{A}_1 & \boldsymbol{\Delta Q}_1^T\boldsymbol{A}_2 & \cdots & \boldsymbol{\Delta Q}_1^T\boldsymbol{A}_p\\ \boldsymbol{0} & \boldsymbol{\Delta Q}_2^T\boldsymbol{A}_2 & \cdots & \boldsymbol{\Delta Q}_2^T\boldsymbol{A}_p\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{\Delta Q}_p^T\boldsymbol{A}_p\end{bmatrix}. \tag{82}$$

Inspection of the nonzero entries in each column of the right-most matrix of the above equation shows that these entries are expressible in terms of the matrices $\boldsymbol{P}_i$ $(i=1,2\ldots p)$ appearing in the recursion formulas (81) as follows:

$$\boldsymbol{\Delta Q}_1^T\boldsymbol{A}_1=\boldsymbol{P}_1;\qquad\begin{bmatrix}\boldsymbol{\Delta Q}_1^T\\ \boldsymbol{\Delta Q}_2^T\end{bmatrix}\boldsymbol{A}_2=\boldsymbol{P}_2;\qquad\cdots\qquad\begin{bmatrix}\boldsymbol{\Delta Q}_1^T\\ \boldsymbol{\Delta Q}_2^T\\ \vdots\\ \boldsymbol{\Delta Q}_i^T\end{bmatrix}\boldsymbol{A}_i=\boldsymbol{P}_i\qquad\cdots\qquad\begin{bmatrix}\boldsymbol{\Delta Q}_1^T\\ \boldsymbol{\Delta Q}_2^T\\ \vdots\\ \boldsymbol{\Delta Q}_p^T\end{bmatrix}\boldsymbol{A}_p=\boldsymbol{P}_p, \tag{83}$$

A.2.2. Finite-precision arithmetic

The preceding recursive scheme would in principle work seamlessly in an ideal infinite-precision arithmetic scenario. Yet the devil is in the details, and when moving to the real case of finite-precision arithmetic, its performance is seriously afflicted by sensitivity to rounding errors and loss of orthogonality over multiple steps—as it occurs with the classical Gram–Schmidt orthogonalization [36].

[20] Note that this statement would cease to be true in the case of approximated basis matrices for the range of $\boldsymbol{A}$ ($\boldsymbol{A}\approx\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}$).

The computational implementation described in Algorithm 8 incorporates ingredients that mitigate these deleterious effects. Re-orthogonalization is carried out in Line 8 by determining the component of $\boldsymbol{\Delta Q}$ which is orthogonal to the current basis $\boldsymbol{Q}$, and then applying the SVD again, setting the corrected $\boldsymbol{\Delta Q}$ equal to the matrix of left-singular vectors of such a decomposition. The effect of rounding errors, on the other hand, is treated by computing the orthogonal basis matrix $\boldsymbol{\Delta Q}$ as the left-singular vectors of the truncated SVD of $\boldsymbol{\Delta A} = \boldsymbol{A}_i - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}_i$ (see Line 12), but using as truncation tolerance a machine-dependent precision parameter based on the norm of $\boldsymbol{A}_i$:

$$\mu_{mach} = \max(n, m_i)\,\mathrm{eps}(\|\boldsymbol{A}_i\|_F) \tag{84}$$

(see Line 10) rather than the default option, which would be in terms of the norm of the input matrix $\boldsymbol{\Delta A}$ (the definition of $\mathrm{eps}(x)$ is given in the description of Algorithm 7).

Algorithm 8: Sequential Randomized Orthogonalization of a partitioned matrix $\boldsymbol{A} = [\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_p]$ (employed in Line 2 of Algorithm 6).

1 **Function** $[\boldsymbol{Q},\ \boldsymbol{L}] \leftarrow \mathtt{SRORTH}([\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_p])$
  Data: $[\boldsymbol{A}_1, \boldsymbol{A}_2, \dots \boldsymbol{A}_p]$: Partitioned matrix (with $\boldsymbol{A}_i \in \mathbb{R}^{n\times m_i}$), and $m = \sum_{i=1}^{p} m_i$)
  Result: Factorization $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{L}$, where $\boldsymbol{Q} \in \mathbb{R}^{n\times r}$ is an orthogonal basis matrix for the column space of $\boldsymbol{A}$ ($r \le \min(n, m)$ is the numerical rank of $\boldsymbol{A}$), while $\boldsymbol{L} = \boldsymbol{Q}^T\boldsymbol{A}$.
2 $\boldsymbol{Q} \leftarrow \emptyset$, $\boldsymbol{P}_i \leftarrow \emptyset$, $i = 1, 2 \dots p$ // Initializations
3 **for** $i = 1$ *to* p **do**
4 $\boldsymbol{\Delta Q} \leftarrow \emptyset$ // Incremental basis matrix
5 **if** $i = 1$ **then**
6 $\boldsymbol{\Delta A} \leftarrow \boldsymbol{A}_i$; $r \leftarrow \mathtt{ceil}(0.01\min(n, m_i))$ // r: Estimation rank $\boldsymbol{A}_i$
7 **else**
8 $\boldsymbol{\Delta A} \leftarrow \boldsymbol{A}_i - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{A}_i$ // Component of $\boldsymbol{A}_i$ orthogonal to the column space of $\boldsymbol{Q}$
9 **end**
10 $\mu_{mach} \leftarrow \max(n, m_i)\,\mathrm{eps}(\|\boldsymbol{A}_i\|_F)$ // Machine-dependent precision parameter
11 **if** $\|\boldsymbol{\Delta A}\|_F > \mu_{mach}$ **then**
12 $[\boldsymbol{\Delta Q}, \bullet, \bullet] \leftarrow \mathtt{RSVDinc}(\boldsymbol{\Delta A}, 0, \mu_{mach}, r)$ // Incremental randomized SVD, see Algorithm 9
13 **if** $i > 1$ **then** $[\boldsymbol{\Delta Q}, \bullet, \bullet] \leftarrow \mathtt{SVDT}((\boldsymbol{\Delta Q} - \boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{\Delta Q}), 0)$ // Re-orthogonalization
14
15 $\boldsymbol{Q} \leftarrow [\boldsymbol{Q}, \boldsymbol{\Delta Q}]$ // Basis matrix augmented with the columns of $\boldsymbol{\Delta Q}$
16 **end**
17 $\boldsymbol{P}_i \leftarrow \boldsymbol{Q}^T\boldsymbol{A}_i$
18 $r \leftarrow \mathtt{ncol}(\boldsymbol{\Delta Q})$ // Number of columns of $\boldsymbol{\Delta Q}$ (estimation for the rank of the next submatrix in Line 12)
19 **end**
20 $\boldsymbol{L} \leftarrow$ Use $\boldsymbol{P}_1, \boldsymbol{P}_2 \dots \boldsymbol{P}_p$ to construct $\boldsymbol{L}$ according to expressions (82) and (83)

A.3. Incremental randomized SVD

Let us focus now on the randomized ingredient of the method, which is the *Incremental Randomized SVD* (RSVDinc) invoked in Line 12 of Algorithm 8 for determining the basis matrix for $\boldsymbol{\Delta A}$. As commented previously, this randomized SVD is partially based on the adaptive randomized algorithm proposed in Ref. [34], and it is the actual ingredient that renders the proposed scheme faster than the standard "deterministic" SVD when the rank of the input matrix is significantly smaller than the minor dimension of the matrix. The reason is that, as argued in Refs. [34] (see also [34,43]), the asymptotic cost of this type of randomization algorithms is $\mathcal{O}(nmr)$, r being the rank of the matrix—as opposed to the standard SVD, whose cost is independent of the rank and scales quadratically with its minor dimension.[21]

Algorithm 9: Incremental Randomized SVD

1 **Function** $[\boldsymbol{\Delta Q}, \boldsymbol{S}, \boldsymbol{V}] \leftarrow \mathtt{RSVDinc}(\boldsymbol{\Delta A}, \epsilon,\ \mu, r)$:
  Data: $\boldsymbol{\Delta A} \in \mathbb{R}^{n\times m_i}$, tolerance threshold $\epsilon \ge 0$. OPTIONAL ARG.: $\mu > 0$: machine precision parameter; r : estimation of rank $\boldsymbol{\Delta A}$ (default $r = \mathrm{ceil}(0.05\max(n, m_i))$)
  Result: Truncated SVD such that $\|\boldsymbol{\Delta A} - \boldsymbol{\Delta Q}\boldsymbol{S}\boldsymbol{V}^T\|_F \le \epsilon$
2 $[\boldsymbol{H}, \boldsymbol{B}] \leftarrow \mathtt{RORTHinc}(\boldsymbol{\Delta A}, \mu, r)$ // $\boldsymbol{H}$: Basis matrix for the column space of $\boldsymbol{\Delta A}$, see Algorithm 10
3 $\epsilon_B \leftarrow \epsilon\,\|\boldsymbol{B}\|_F$; $\mu_{mach} = \max(n, m_i)\,\mathrm{eps}(\|\boldsymbol{B}\|_F)$
4 $[\boldsymbol{\Delta Q}, \boldsymbol{S}, \boldsymbol{V}] \leftarrow \mathtt{SVDT}(\boldsymbol{B}, \epsilon_B, \mu_{mach})$ // Truncated SVD of $\boldsymbol{B} = \boldsymbol{H}^T\boldsymbol{\Delta A}$, see Algorithm 7
5 $\boldsymbol{\Delta Q} \leftarrow \boldsymbol{H}\,\boldsymbol{\Delta Q}$;

[21] For a detailed account of the asymptotic costs of classical SVD, the reader is referred to Ref. [44], Lecture 31.

The pseudo-code of this randomized SVD is described in Algorithm 9. The basic steps are identical to the ones employed in the SRSVD of Algorithm 6, namely, (1) determination of an orthogonal basis matrix $\boldsymbol{H}$ for the column space (range) of the input matrix $\boldsymbol{\Delta A}$ in Line 2; and (2) Truncated SVD of $\boldsymbol{B} = \boldsymbol{H}^T \boldsymbol{\Delta A}$ in Line 4. The computation of the basis matrix in the first step also shares common features with the one employed in the SRSVD of Algorithm 6. Indeed, as can be inferred from the pseudo-code of Algorithm 10 of the function devoted to this task (RORTHinc), the desired orthogonal basis matrix $\boldsymbol{\Delta H}$ is built iteratively in a Gram–Schmid orthogonalization fashion:

$$\boldsymbol{\Delta H} = \begin{bmatrix} \boldsymbol{\Delta H}_1 & \boldsymbol{\Delta H}_2 & \cdots & \boldsymbol{\Delta H}_s \end{bmatrix}, \quad \text{where} \quad \boldsymbol{\Delta H}_i^T \boldsymbol{\Delta H}_j = \boldsymbol{0}\ (i \neq j), \qquad \boldsymbol{\Delta H}_i^T \boldsymbol{\Delta H}_i = \boldsymbol{I}. \tag{85}$$

The actual difference with SRORTH in Algorithm 8 is that, at a given iteration i, the corresponding orthogonal matrix $\boldsymbol{\Delta H}_i$ is not determined from a column partition of the input matrix, but rather as an orthogonal basis matrix for the range of the matrix defined by

$$\boldsymbol{Y}_i = \frac{1}{\sqrt{n\Delta R_i}} \boldsymbol{\Omega}_i \boldsymbol{C}_i, \qquad i = 1, 2 \ldots s \tag{86}$$

(see Lines 4 and 5 in Algorithm 10). Here, $\boldsymbol{C}_i \in \mathbb{R}^{n \times m}$ is the residual matrix at iteration u, whereas $\boldsymbol{\Omega}_i$ is a $n \times \Delta R_i$ standard Gaussian test matrix (a *random* matrix whose entries are independent standard normal variables). The distinguishing feature of our algorithm with respect to the original scheme put forward in Ref. [34] is that, in our case, the number of columns ΔR_i of this random matrix changes during the iterations. In the first iteration ($i = 1$), we set $\Delta R_1 = R$, R being the number of columns of the incremental basis matrix of the previous block matrix (see Line 18 in Algorithm 8). As argued in Ref. [42], if this initial guess is well above the rank of input matrix $\boldsymbol{\Delta A}$, it is highly probable that the basis matrix for the range of $\boldsymbol{Y}_1$ in Eq. (86) is the required orthogonal matrix $\boldsymbol{\Delta H}$. Numerical experience shows that when the submatrices $\boldsymbol{A}_{k-1}$ and $\boldsymbol{A}_k$ ($k = 2, 3 \ldots$ p) correspond to input parameters that are close in parameter space, then this estimation is normally a reliable upper bound, and therefore, only one iteration is required.

If this first iteration is not sufficient to reduce the norm of the residual matrix $\boldsymbol{C}_i$ below the prescribed error threshold μ (see Line 3 in Algorithm 10), then it is necessary to calculate a guess for the number of columns of the random matrix in the next iteration. Our proposal in this respect is to use the logarithmic estimation displayed in Line 12 of Algorithm 10. This estimation is based on the observation that, in most physical problems amenable to dimensionality reduction, the singular values of the integrand matrix decay in an exponential manner. Nevertheless, to avoid situation in which the estimated increments ΔR_i are either too large or too small, the minimum and maximum sizes of the increment, ΔR_m and ΔR_M respectively, can be also specified as optional arguments.

Lastly, it should be noted that this randomized factorization is also subject to the vagaries of finite precision arithmetics. To address this, Algorithm 10 includes a re-orthogonalization step in Line 8.

Algorithm 10: Incremental Randomized orthogonalization, with steps of varying size

1 **Function** $[\boldsymbol{H}, \boldsymbol{B}, c] \leftarrow$ RORTHinc$(\boldsymbol{C}, \mu, R, \Delta R_m, \Delta R_M)$:
  Data: $\boldsymbol{C} \in \mathbb{R}^{n \times m}$, tolerance $\mu > 0$; rank estimate $R < q$ (where $q = \min(n, m)$). Optional arguments ΔR_m, ΔR_M bounds for rank increment ΔR (default ceil$(0.01q)$ and ceil$(0.25q)$)
  Result: Orthogonal matrix $\boldsymbol{H}$ such that $\|\boldsymbol{C} - \boldsymbol{H}\boldsymbol{B}\|_F \leq \mu$, where $\boldsymbol{B} = \boldsymbol{H}^T \boldsymbol{C}$
2 $\boldsymbol{H} \leftarrow \emptyset$; $\boldsymbol{B} \leftarrow \emptyset$; $\Delta R \leftarrow R$; $c \leftarrow \mu + 1$; $i \leftarrow 1$; $q \leftarrow \min(n, m)$
3 **while** $c \geq \mu$ **do**
4 $\Omega \leftarrow$ randn$(n, \Delta R)/\sqrt{n\Delta R}$ // Draw a random $n \times \Delta R$ matrix
5 $[\boldsymbol{\Delta H}, \cdot, \cdot] \leftarrow$ SVDT$(\boldsymbol{C}\Omega, 0)$ // Orthogonal basis matrix for range $(\boldsymbol{C}\,\Omega)$
6 **if** $\boldsymbol{\Delta H} = \emptyset$ **then** break // Exiting the loop
7
8 **if** $i > 1$ **then** $[\boldsymbol{\Delta H}, \cdot, \cdot] \leftarrow$ SVDT$(\boldsymbol{\Delta H} - \boldsymbol{H}(\boldsymbol{H}^T \boldsymbol{\Delta H}), 0)$ // Re-orthogonalization
9
10 $\boldsymbol{\Delta B} \leftarrow \boldsymbol{\Delta H}^T \boldsymbol{C}$; $\boldsymbol{C} \leftarrow \boldsymbol{C} - \boldsymbol{\Delta H}\boldsymbol{\Delta B}$; $c \leftarrow \|\boldsymbol{C}\|_F$ // Residual update
11 $\boldsymbol{H} \leftarrow \begin{bmatrix} \boldsymbol{H} & \boldsymbol{\Delta H} \end{bmatrix}$; $\boldsymbol{B} \leftarrow \begin{bmatrix} \boldsymbol{B}^T & \boldsymbol{\Delta B}^T \end{bmatrix}^T$; // Basis matrix update
12 $\bar{R} \leftarrow R_i + \dfrac{R - R_i}{logc - logc_i}(log\mu - logc_i)$; $\bar{R} \leftarrow$ min$(q, \bar{R})$ // Logarithmic estimation rank
13 $\Delta R \leftarrow$ min$(\Delta R_M$, ceil$(\bar{R} - R))$; $\Delta R \leftarrow$ max$(\Delta R_m, \Delta R)$; $\bar{R} \leftarrow R + \Delta R$
14 $i \leftarrow i + 1$; $c_i \leftarrow c$; $R_i \leftarrow R$; $R \leftarrow \bar{R}$
15 **end**

A.4. Numerical study

To compare the performance of the standard SVD and the proposed SRSVD, we use the convergence study presented in Table 4 of Section 6.3 for determining an orthogonal basis functions for the parameterized, vector-valued function of Eq. (70). It can be gleaned from this table that the proposed SRSVD clearly outperforms the standard SVD, both in terms of computing time and memory requirements. For instance, for the 16 GB matrix, the SRSVD turns out to be almost 7 times faster than the standard SVD, and for the largest matrix of 33 GB, the standard SVD simply cannot handle the operation because it exhausts the memory capabilities of the employed 64 GB RAM computer. Furthermore, we can see that, in passing from the matrix of 16 GB to the matrix of 33 GB, the computing time of the SRSVD increases by a factor slightly below 2, a fact that indicates that the cost scales

approximately linearly with the number of columns—as opposed to the standard SVD, whose asymptotic costs scales quadratically with its minor dimension [44]. It is noteworthy also that, as the number of partitions increases, the number of iterations required by the incremental randomized orthogonalization of Algorithm 10 tends to one. This indicates that, as conjectured in Appendix A.3 of Appendix, the rank of a given block matrix is a reliable upper bound for the rank of the orthogonal complement of the next block matrix in the sequence. Incidentally, this may explain in part why the asymptotic costs of the SQRSV appears to scale linearly with its minor dimension, for the standard randomized SVD itself exhibits this desirable feature [34]. Last but not least, we show in Table 4 the relative difference between the singular values computed by both approaches. The results reveal that the difference is negligible, a fact that supports the theoretical claim made at the outset of this Appendix, according to which *the proposed SRSVD is not an approximate method* for computing the truncated SVD of a matrix, but rather an *alternative method to compute the exact factorization*—one that exploits linear correlations existing between blocks of the matrix.

References

[1] C. Farhat, P. Avery, T. Chapman, J. Cortial, Dimensional reduction of nonlinear finite element dynamic models with finite rotations and energy-based mesh sampling and weighting for computational efficiency, Internat. J. Numer. Methods Engrg. 98 (9) (2014) 625–662.
[2] J.A. Hernández, M.A. Caicedo, A. Ferrer, Dimensional hyper-reduction of nonlinear finite element models via empirical cubature, Comput. Methods Appl. Mech. Engrg. 313 (2017) 687–722.
[3] S. An, T. Kim, D. James, Optimizing cubature for efficient integration of subspace deformations, ACM Trans. Graph. 27 (5) (2009) 165.
[4] T. Kim, J. Delaney, Subspace fluid re-simulation, ACM Trans. Graph. 32 (4) (2013) 62.
[5] C.L. Lawson, R.J. Hanson, Solving Least Squares Problems, Vol. 161, SIAM, 1974.
[6] C. von Tycowicz, C. Schulz, H.P. Seidel, K. Hildebrandt, An efficient construction of reduced deformable objects, ACM Trans. Graph. 32 (6) (2013) 213.
[7] Z. Pan, H. Bao, J. Huang, Subspace dynamic simulation using rotation-strain coordinates, ACM Trans. Graph. 34 (6) (2015) 242.
[8] D.L. Donoho, For most large underdetermined systems of linear equations the minimal 1-norm solution is also the sparsest solution, Commun. Pure Appl. Math. J. Issued Courant Inst. Math. Sci. 59 (6) (2006) 797–829.
[9] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge Univ Pr, 2004.
[10] T. Blumensath, M.E. Davies, Normalized iterative hard thresholding: Guaranteed stability and performance, IEEE J. Sel. Top. Signal Process. 4 (2) (2010) 298–309.
[11] J. Yang, Y. Zhang, Alternating direction algorithms for \ell_1-problems in compressive sensing, SIAM J. Sci. Comput. 33 (1) (2011) 250–278.
[12] S.S. Chen, D.L. Donoho, M.A. Saunders, Atomic decomposition by basis pursuit, SIAM Rev. 43 (1) (2001) 129–159.
[13] T. Chapman, P. Avery, P. Collins, C. Farhat, Accelerated mesh sampling for the hyper reduction of nonlinear computational models, Internat. J. Numer. Methods Engrg. (2016).
[14] J.A. Hernández, J. Oliver, A.E. Huespe, M.A. Caicedo, J. Cante, High-performance model reduction techniques in computational multiscale homogenization, Comput. Methods Appl. Mech. Engrg. 276 (2014) 149–189.
[15] J. Oliver, M. Caicedo, A. Huespe, J.A. Hernández, E. Roubin, Reduced order modeling strategies for computational multiscale fracture, Comput. Methods Appl. Mech. Engrg. 313 (2017) 560–595.
[16] J.A. Hernández, A multiscale method for periodic structures using domain decomposition and ECM-hyperreduction, Comput. Methods Appl. Mech. Engrg. 368 (2020) 113192.
[17] A.T. Patera, M. Yano, An LP empirical quadrature procedure for parametrized functions, C. R. Math. 355 (11) (2017) 1161–1167.
[18] M. Yano, A.T. Patera, An LP empirical quadrature procedure for reduced basis treatment of parametrized nonlinear PDEs, Comput. Methods Appl. Mech. Engrg. 344 (2019) 1104–1123.
[19] M. Yano, Discontinuous Galerkin reduced basis empirical quadrature procedure for model reduction of parametrized nonlinear conservation laws, Adv. Comput. Math. (2019) 1–34.
[20] S. Chaturantabut, D.C. Sorensen, Nonlinear model reduction via discrete empirical interpolation, SIAM J. Sci. Comput. 32 (5) (2010) 2737–2764.
[21] K. Carlberg, C. Farhat, J. Cortial, D. Amsallem, The GNAT method for nonlinear model reduction: effective implementation and application to computational fluid dynamics and turbulent flows, J. Comput. Phys. (2013).
[22] P. Tiso, R. Dedden, D. Rixen, A modified discrete empirical interpolation method for reducing non-linear structural finite element models, in: International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Vol. 55973, American Society of Mechanical Engineers, 2013, V07BT10A043.
[23] R. Everson, L. Sirovich, Karhunen–Loeve procedure for gappy data, J. Opt. Soc. Amer. A 12 (8) (1995) 1657–1664.
[24] J. Rutzmoser, Model Order Reduction for Nonlinear Structural Dynamics (Ph.D. thesis), Technische Universität München, 2018.
[25] C. Farhat, T. Chapman, P. Avery, Structure-preserving, stability, and accuracy properties of the energy-conserving sampling and weighting method for the hyper reduction of nonlinear finite element dynamic models, Internat. J. Numer. Methods Engrg. 102 (5) (2015) 1077–1110.
[26] D. Kim, J.P. Haldar, Greedy algorithms for nonnegativity-constrained simultaneous sparse recovery, Signal Process. 125 (2016) 274–289.
[27] G.H. Golub, G. Meurant, Matrices, Moments and Quadrature with Applications, Princeton University Press, 2009.
[28] J. Bremer, Z. Gimbutas, V. Rokhlin, A nonlinear optimization procedure for generalized Gaussian quadratures, SIAM J. Sci. Comput. 32 (4) (2010) 1761–1788.
[29] H. Xiao, Z. Gimbutas, A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions, Comput. Math. Appl. 59 (2) (2010) 663–676.
[30] J. Ma, V. Rokhlin, S. Wandzura, Generalized Gaussian quadrature rules for systems of arbitrary functions, SIAM J. Numer. Anal. 33 (3) (1996) 971–996.
[31] S. Mousavi, H. Xiao, N. Sukumar, Generalized Gaussian quadrature rules on arbitrary polygons, Internat. J. Numer. Methods Engrg. 82 (1) (2010) 99–113.
[32] S. Mousavi, N. Sukumar, Numerical integration of polynomials and discontinuous functions on irregular convex polygons and polyhedrons, Comput. Mech. 47 (5) (2011) 535–554.
[33] Z. Wang, B. McBee, T. Iliescu, Approximate partitioned method of snapshots for pod, J. Comput. Appl. Math. 307 (2016) 374–384.
[34] P.G. Martinsson, S. Voronin, A randomized blocked algorithm for efficiently computing rank-revealing factorizations of matrices, 2015, arXiv preprint arXiv:1503.07157.
[35] G. Liu, S. Quek, the Finite Element Method: A Practical Course, Butterworth-Heinemann, 2003.
[36] G.H. Golub, C.F. Van Loan, Matrix Computations, Vol. 3, JHU Press, 2012.
[37] C. Davis, Theory of positive linear dependence, Amer. J. Math. (1954) 733–746.
[38] T. Bui-Thanh, K. Willcox, O. Ghattas, B. van Bloemen Waanders, Goal-oriented, model-constrained optimization for reduction of large-scale systems, J. Comput. Phys. 224 (2) (2007) 880–896.
[39] A. Giuliodori, J.A. Hernández, E. Soudah, Multiscale modeling of prismatic heterogeneous structures based on a localized hyperreduced-order method, Comput. Methods Appl. Mech. Engrg. 407 (2023) 115913.

[40] J.A. Hernández, A. Giuliodori, E. Soudah, Empirical interscale finite element method (EIFEM) for modeling heterogeneous structures via localized hyperreduction, Comput. Methods Appl. Mech. Engrg. 418 (2024) 116492.
[41] J.N. Reddy, Energy Principles and Variational Methods in Applied Mechanics, John Wiley & Sons, 2017.
[42] N. Halko, P.G. Martinsson, J.A. Tropp, Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, SIAM Rev. 53 (2) (2011) 217–288.
[43] P.G. Martinsson, J.A. Tropp, Randomized numerical linear algebra: Foundations and algorithms, Acta Numer. 29 (2020) 403–572.
[44] L.N. Trefethen, D. Bau, Numerical Linear Algebra, Vol. 50, Siam, 1997.