Encapsulated PGD Algebraic Toolbox Operating with High-Dimensional Data

P. Díez[1,2] · S. Zlotnik[1,2] · A. García-González[1,2] · A. Huerta[1,2]

Abstract

In its original conception, proper generalized decomposition (PGD) provides explicit parametric solutions, denoted as computational vademecums or digital abacuses, to parametric boundary value problems. The PGD approach is extended here to devise a set of algebraic tools enabling to operate with multidimensional tensor data. These tools are designed to store, compress and perform basic operations (in particular divisions) with tensors in separable format. These tools are directly producing the computational vademecums for the resulting high-dimensional tensor data. Thus, the general methodology enables performing nontrivial operations (storage, compression, division, solving linear systems of equations...) for multi-dimensional tensor data. The idea is based on the principle of the PGD separation, that produces a separable least squares approximation of any multidimensional function. The PGD compression is a particular case, extensively used in practice to compact the separable solution without loss of accuracy. Here, this concept is applied to algebraic tensor structures that are also seen as functions in multidimensional Cartesian domains. Moreover, a straightforward extension of this concept is devised to operate with multidimensional objects stored in the separable format. That allows creating a toolbox of PGD arithmetic operators that is publicly released at https://git.lacan.upc.edu/zlotnik/algebraicPGDtools. Numerical tests dem-onstrate the performance and efficiency of the toolbox, both for tensor data handling and operation and also in applications pertaining to the discretized version of boundary value problems.

1 Introduction

Algebraic separation of multidimensional tensors has extensive applications in methods dealing with multidimensional data, and is therefore object of intensive research efforts [12, 13, 16]. Tensor separation is a very efficient strategy for data compression and opens the door to operate in higher dimensions following incremental-iterative approaches [1, 2]. These operations with high-dimensional tensors are a key ingredient in the solution of parametric problems in computational mechanics, in particular for stochastic models [7, 8].

Proper generalized decomposition (PGD) [4, 5] produces separated representations of multidimensional functions. In the discrete format, this is equivalent to obtain separated format of multidimensional tensor data. The initial idea is to obtain a separated representation of the (unknown) solution of some boundary-value problem. Here, the same idea is exploited to obtain separable approximations of multidimensional objects that result of operating with other multidimensional objects, in particular when the original objects are already expressed with a separable representation, as already analyzed in [15].

The aim of this paper is to introduce a general methodology to operate with separable representation of multidimensional data. First, it is natural to recall and provide an algorithmic description of the PGD least squares projection in a functional framework, see [6]. These functions are readily represented by tensorial discretizations (a function with one argument is discretized as a vector, with two arguments in a cartesian domain as a matrix, with three as a tensor of order three...).

1.1 Background and Notation

The multidimensional data is represented by a real-valued function F taking values in $\Omega \subset \mathbb{R}^{n_d}$. The domain Ω is

✉ P. Díez
pedro.diez@upc.edu

[1] Laboratori de Càlcul Numèric, E.T.S. de Ingeniería de Caminos, Universitat Politècnica de Catalunya, Barcelona, Spain

[2] International Centre for Numerical Methods in Engineering, CIMNE, Barcelona, Spain

assumed to be Cartesian, that is Ω is the Cartesian product $\mathtt{n_d}$ *simple* sectional domains Ω_i, for $i = 1, 2, \ldots, \mathtt{n_d}$. Typically, the sectional domains are real intervals, $\Omega_i =]a_i, b_i[$, with $a_i < b_i$, where the variable x_i ranges (the term *sectional* refers to one individual coordinate or dimension).

Definition 1 The function F is said to be separable if for some integer value M, there exists a set of functions f_i^m taking values in Ω_i, for $i = 1, 2, \ldots, \mathtt{n_d}$ and $m = 1, 2, \ldots, M$, such that

$$F(x_1, x_2, \ldots, x_{\mathtt{n_d}}) = \sum_{m=1}^{M} f_1^m(x_1) f_2^m(x_2) \cdots f_{\mathtt{n_d}}^m(x_{\mathtt{n_d}}). \tag{1}$$

Note that this is the form adopted by the PGD solutions.

Remark 1 (*Dealing with vector coordinates*) For the sake of a simpler presentation, each coordinate x_i for $i = 1, 2, \ldots, \mathtt{n_d}$ is taken as ranging in a 1D domain $]a_i, b_i[$. In the general case the coordinates belong to arbitrary domains $\Omega_i \subset \mathbb{R}^{d_i}$ with integer dimensions $d_i \geq 1$. A particularly interesting case is taking the first coordinate x_1 as describing the physical domain (thus, typically in $\mathbb{R}^2$ or $\mathbb{R}^3$) and the rest as scalar parameters. In the remainder, for the sake of a clearer presentation and with no loss of generality, it is assumed that all the coordinates are scalar magnitudes.

The $\mathtt{n_d}$-dimensional domain Ω is discretized with a *Cartesian* mesh resulting of discretizing each of the sectional domains Ω_i with n_i nodes denoted by x_i^k, for $k = 1, 2, \ldots, n_i$. In the particular case of selecting a uniform grid in 1D sectional domains $\Omega_i =]a_i, b_i[$, these nodes read $x_i^k = a_i + (b_i - a_i)(k-1)/(n_i - 1)$ for $i = 1, 2, \ldots, \mathtt{n_d}$ and $k = 1, 2, \ldots, n_i$. Note however that the proposed methodology is not restricted to uniformly distributed nodes. Then, multidimensional function F is readily represented by the tensor of its nodal values $\boldsymbol{F} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{\mathtt{n_d}}}$, such that

$$[\boldsymbol{F}]_{k_1 k_2 \ldots k_{\mathtt{n_d}}} = F(x_1^{k_1}, x_2^{k_2}, \ldots, x_{\mathtt{n_d}}^{k_{\mathtt{n_d}}})$$

The separability of $\boldsymbol{F}$ (the discrete representation of F) is defined as follows.

Definition 2 Tensor $\boldsymbol{F} \in \mathbb{R}^{n_1 \times \cdots \times n_{\mathtt{n_d}}}$ is said to be separable if for some integer value M, there exist a set of vectors $\boldsymbol{f}_i^m \in \mathbb{R}^{n_i}$ for $i = 1, 2, \ldots, \mathtt{n_d}$ and $m = 1, 2, \ldots, M$ such that

$$\boldsymbol{F} = \sum_{m=1}^{M} \boldsymbol{f}_1^m \otimes \boldsymbol{f}_2^m \otimes \cdots \otimes \boldsymbol{f}_{\mathtt{n_d}}^m \tag{2}$$

Note that Definitions 1 and 2 are equivalent in the sense that vectors $\boldsymbol{f}_i^m$ are seen as containing the nodal values of f_i^m in the grid of points x_i^k for $i = 1, 2, \ldots, \mathtt{n_d}$, $m = 1, 2, \ldots, M$ and $k = 1, 2, \ldots, n_i$, that is $[\boldsymbol{f}_i^m]_k = f_i^m(x_i^k)$. The aim of this paper is to present numerical strategies to produce and operate with separable objects, mainly in their tensorial format. The functional format is here equivalent to the tensorial format. It is used in the presentation because the basis of the PGD is formulated in a functional setting as a method to solve parametric PDEs.

1.2 Layout of the Paper

First, in Sect. 2, the PGD compression based on the least-squares higher-order projection introduced in [14] and algorithmically detailed in [6] is reviewed and presented both in the functional and tensorial setup, including here the new feature of using nontrivial least-squares norms, that is the possibility of having different projection criteria (different norms in the least-squares fitting) for each dimension. This is discussed in Sect. 2.3. A synthetic algorithm is also presented in Sect. 2.2, which is accompanied by a downloadable `matlab` implementation.

The PGD compression is generalized in Sect. 3 to define operations between separable approximations. The sum and product are considered as the straightforward operations. Note that operators are a priori very simple. However, in practice, they require a compression to be effective. Section 3.1 describes how to perform a nontrivial operation between two tensors in separable format, the Hadamard division. Also here, a synthetic algorithm is presented to illustrate the implementation available in the open source repository. Section 3.2 presents the strategy to solve linear systems of equations arising from the parametric version of a discretized boundary value problem (using the preferred method, viz. finite elements, finite differences, boundary elements...). In practice, it requires two input tensors (one representing the parametric matrix, one representing the parametric vector). All the parametric sectional dimensions have to be of the same type in both tensors. The first dimension (the so-called space dimension) has to be compatible: a square matrix for the left-hand-side term and a vector of the same length for the right-hand-side.

Section 4 presents a set of examples illustrating the capabilities of the presented algorithms for all the operations analyzed.

2 Least-Squares PGD Separation and Compression

2.1 Problem Statement in the Functional Framework

A bilinear form $A(\cdot, \cdot)$ taking values in $\mathcal{L}_2(\Omega) \times \mathcal{L}_2(\Omega)$ is expressed in terms of how it affects rank-one separable

functions, that is functions that can be expressed in the form of (1) with $M = 1$. Namely, for two rank-one functions like $F(x_1, x_2, \dots, x_{\mathtt{n_d}}) = f_1(x_1) f_2(x_2) \dots f_{\mathtt{n_d}}(x_{\mathtt{n_d}})$ and $G(x_1, x_2, \dots, x_{\mathtt{n_d}}) = g_1(x_1) g_2(x_2) \dots g_{\mathtt{n_d}}(x_{\mathtt{n_d}})$, the form reads

$$A(F, G) = \prod_{i=1}^{\mathtt{n_d}} a_i(f_i, g_i) \tag{3}$$

where the sectional symmetric and positive definite bilinear forms $a_i(\cdot, \cdot)$ take values in $\mathcal{L}_2(\Omega_i) \times \mathcal{L}_2(\Omega_i)$ for $i = 1, 2, \dots, \mathtt{n_d}$.

The definition of $A(\cdot, \cdot)$ for general separable functions of arbitrary ranks (not only for rank one, as in (3), is introduced as follows. Let F and G be such that

$$F(x_1, x_2, \dots, x_{\mathtt{n_d}}) = \sum_{m=1}^{M} f_1^m(x_1) f_2^m(x_2) \dots f_{\mathtt{n_d}}^m(x_{\mathtt{n_d}}),$$

$$G(x_1, x_2, \dots, x_{\mathtt{n_d}}) = \sum_{\hat{m}=1}^{\hat{M}} g_1^{\hat{m}}(x_1) g_2^{\hat{m}}(x_2) \dots g_{\mathtt{n_d}}^{\hat{m}}(x_{\mathtt{n_d}})$$

it is assumed that

$$A(F, G) = \sum_{m=1}^{M} \sum_{\hat{m}=1}^{\hat{M}} \prod_{i=1}^{\mathtt{n_d}} a_i(f_i^m, g_i^{\hat{m}}) \tag{4}$$

This definition is such that $A(\cdot, \cdot)$ inherits the bilinear symmetric and positive definiteness properties of the sectional forms $a_i(\cdot, \cdot)$, $i = 1, \dots, \mathtt{n_d}$.

Remark 2 (*Bilinear nature of form* $A(\cdot, \cdot)$) Note that the form $A(\cdot, \cdot)$ defined from (3), (4) and the set of bilinear sectional forms $a_i(\cdot, \cdot)$ is itself bilinear, symmetric and positive definite. The symmetry and the homogeneity for scalars, that is

$$A(F, G) = A(G, F) \quad \text{and} \quad A(\alpha F, G) = \alpha A(F, G)$$

are obvious from (4) and the symmetry of the sectional forms. The linearity of each argument is readily shown by considering a function

$$\tilde{F}(x_1, x_2, \dots, x_{\mathtt{n_d}}) = \sum_{\tilde{m}=1}^{\tilde{M}} \tilde{f}_1^{\tilde{m}}(x_1) \tilde{f}_2^{\tilde{m}}(x_2) \dots \tilde{f}_{\mathtt{n_d}}^{\tilde{m}}(x_{\mathtt{n_d}})$$

and observing that

$$\begin{aligned} A(F + \tilde{F}, G) = & \sum_{m=1}^{M} \sum_{\hat{m}=1}^{\hat{M}} \prod_{i=1}^{\mathtt{n_d}} a_i(f_i^m, g_i^{\hat{m}}) \\ & + \sum_{\tilde{m}=1}^{\tilde{M}} \sum_{\hat{m}=1}^{\hat{M}} \prod_{i=1}^{\mathtt{n_d}} a_i(\tilde{f}_i^{\tilde{m}}, g_i^{\hat{m}}) \\ = & A(F, G) + A(\tilde{F}, G) \end{aligned}$$

The positive definiteness results from the positive definiteness of the sectional forms.

Remark 3 (*Description of bilinear form* $A(\cdot, \cdot)$ *with only one separated term*) The form $A(\cdot, \cdot)$ is defined in Eq. (3) with only one product, and not with a sum of these terms as in, for instance, [17]. In a more general setup, the operator is often characterized by an affine decomposition with n terms, that is

$$A(F, G) = \sum_{\ell=1}^{n} \prod_{i=1}^{\mathtt{n_d}} a_i^{\ell}(f_i, g_i)$$

The form adopted in Eq. (3), has only one term because it is sufficient to properly represent a standard norm in $\mathcal{L}_2(\Omega) \times \mathcal{L}_2(\Omega)$ by introducing the sectional norm in each of the tensorial dimensions. This is the case of the standard $\mathcal{L}_2$ product when it reduces to separable functions. The generalization of the methodologies presented here to the case in which $A(\cdot, \cdot)$ is defined by a general affine decomposition with n terms ($n \geq 2$) is straightforward.

The standard definition of least-squares projection of some function $\boldsymbol{\Phi} \in \mathcal{L}_2(\Omega)$ into some subset $V \subset \mathcal{L}_2(\Omega)$ is the element $F \in V$ such that

$$F = \arg\min_{G \in V} A(G - \boldsymbol{\Phi}, G - \boldsymbol{\Phi}) \tag{5}$$

In the case of V being a linear subspace of $\mathcal{L}_2(\Omega)$ of finite dimension, the problem of obtaining F results in a linear system of algebraic equations (normal equations; the matrix of the system is the representation of $A(\cdot, \cdot)$ in the basis describing the linear subspace).

If subset V has not the structure of a vectorial space with a well identified basis, the least-squares definition in (5) still holds, but obtaining the approximation as a best fit is not as simple.

In PGD, the proper definition of the *approximation space* V is tricky. As it is clear from (1), and its algebraic version (2), even with a single term ($M = 1$, the aforementioned rank-one approximation) the structure of the approximation is not linear.

In any case, solving problem (5) is equivalent to find an stationary point F such that

$$A(F - \boldsymbol{\Phi}, F^\star) = 0 \tag{6}$$

for all functions $F^\star$ in a proper test space.

The PGD least-squares algorithm is based on two ideas: a greedy algorithm combined with an alternating directions nonlinear solver. The greedy part consists in computing sequentially the terms in the sum for $m = 1, \dots, M$. The alternating directions iterative solver is applied to

the nonlinear least-squares problem of finding the best fit of one-term separable approximation (the best rank-one approximation).

2.2 Rank-One Approximation

Let V be the subset of $\mathcal{L}_2(\Omega)$ containing the functions F that can be expressed as

$$F(x_1, x_2, \dots, x_{\mathtt{n_d}}) = f_1(x_1) f_2(x_2) \dots f_{\mathtt{n_d}}(x_{\mathtt{n_d}}). \tag{7}$$

for some functions $f_i \in V_i \subset \mathcal{L}_2(\Omega_i)$, $i = 1, 2, \dots, \mathtt{n_d}$. These functions are separable with only one-term for the sectional spaces $V_i \subset \mathcal{L}_2(\Omega_i)$, and are denoted as rank-one approximations. Even if the sectional spaces V_i are linear, the subset V of rank-one functions is not a linear subspace of $\mathcal{L}_2(\Omega)$.

Thus, the problem under consideration reads: for some given function $\boldsymbol{\Phi}(x_1, x_2, \dots, x_{\mathtt{n_d}}) \in \mathcal{L}_2(\Omega)$, find $F \in V$ according to the least-squares criterion given in (5).

Note that it is of special interest the particular case when $\boldsymbol{\Phi}$ is already in separated format, that is, for some M_ϕ

$$\boldsymbol{\Phi}(x_1, x_2, \dots, x_{\mathtt{n_d}}) = \sum_{m=1}^{M_\phi} \phi_1^m(x_1) \phi_2^m(x_2) \dots \phi_{\mathtt{n_d}}^m(x_{\mathtt{n_d}}). \tag{8}$$

The methodology applied to this case is denoted as *compression* (and not *separation*) because the resulting approximation is expected to have less terms ($M \ll M_\phi$).

Thus, the scalar form to be minimized by the solution F reads

$$A(F - \boldsymbol{\Phi}, F - \boldsymbol{\Phi}) = A(F, F) - 2A(F, \boldsymbol{\Phi}) - A(\boldsymbol{\Phi}, \boldsymbol{\Phi})$$

The two terms that depend on the unknown F are

$$A(F, F) = \prod_{i=1}^{\mathtt{n_d}} a_i(f_i, f_i) \tag{9}$$

and, for the general case of $\boldsymbol{\Phi}$ non separable,

$$A(F, \boldsymbol{\Phi}) = A\left(\prod_{i=1}^{\mathtt{n_d}} f_i, \boldsymbol{\Phi} \right). \tag{10}$$

Whereas for the particular case of separable $\boldsymbol{\Phi}$, the same term reads

$$A(F, \boldsymbol{\Phi}) = \sum_{m=1}^{M_\phi} \prod_{i=1}^{\mathtt{n_d}} a_i(f_i, \phi_i^m). \tag{11}$$

The proposed methodology to find the rank-one approximation is based on the alternating directions approach. A succession of approximations to $f_\gamma, \gamma = 1, 2, \dots, \mathtt{n_d}$, is iteratively produced, expecting to converge to the actual value of f_γ. The iterations are obtained in an alternating directions fashion, that is in order to compute f_γ, the previously computed approximations to the rest of the unknowns f_j, for $j \neq \gamma$ are assumed to be known and therefore not modified in the current iteration.

Thus, at the stage of computing component γ in the current iteration, the unknown reads

$$F = \left[\prod_{j \neq \gamma} f_j \right] f_\gamma,$$

where the part in brackets is known and computable, and the term $A(F, F)$ results

$$A(F, F) = \underbrace{\left[\prod_{j \neq \gamma} a_j(f_j, f_j) \right]}_{=: \alpha(\text{computable})} a_\gamma(f_\gamma, f_\gamma). \tag{12}$$

The term $A(F, \boldsymbol{\Phi})$ reads for the general case described in (10)

$$A(F, \boldsymbol{\Phi}) = A\left(\left[\prod_{j \neq \gamma} f_j \right] f_\gamma, \boldsymbol{\Phi} \right) := \ell_\gamma(f_\gamma) \tag{13}$$

where the linear form $\ell_\gamma(\cdot)$ depends on known functions f_j (for $j \neq \gamma$) and $\boldsymbol{\Phi}$. Characterizing $\ell_\gamma(\cdot)$ requires integrating $\boldsymbol{\Phi}$ along all dimensions but the i-th. In any case, the linear form $\ell_\gamma(\cdot)$ may be described by its Riesz representation with respect to $a_\gamma(\cdot, \cdot)$, g such that, for all $f^\star \in V_i$

$$\ell_\gamma(f^\star) = a_\gamma(g, f^\star). \tag{14}$$

Remark 4 (*Computing the Riesz representation g*) The Riesz representation of $\ell_\gamma(\cdot)$ with respect to $a_\gamma(\cdot, \cdot)$, g, is introduced for convenience in the following developments. In practice, computing function $g \in \mathcal{L}_2(\Omega_\gamma)$ (or a discrete approximation) requires solving the linear problem (14) for all $f^\star$. This is exactly the strategy adopted in [14] to approximate a multidimensional continuous function, and requires solving a linear system of equations with the matrix that represents the sectional bilinear form $a_\gamma(\cdot, \cdot)$. In the following, the focus is made in approximating and operating with discrete objects expressed in terms of tensors and it is convenient to identify the right-hand-side of the problems to be solved with function g (or vector $\boldsymbol{g}$).

In the particular case of separable $\boldsymbol{\Phi}$, see (11), the expression (13) becomes

$$A(F, \boldsymbol{\Phi}) = \sum_{m=1}^{M_\phi} \underbrace{\left[\prod_{j \neq \gamma} a_j(f_j, \phi_j^m) \right]}_{=: \beta_m(\text{computable})} a_\gamma(f_\gamma, \phi_\gamma^m). \tag{15}$$

Therefore, the expression for $\ell_\gamma(\cdot)$ is

$$\ell_\gamma(f_\gamma) = \sum_{m=1}^{M_\phi} \beta_m a_\gamma(f_\gamma, \phi_\gamma^m) = a_\gamma\left(f_\gamma, \sum_{m=1}^{M_\phi} \beta_m \phi_\gamma^m\right) \tag{16}$$

and therefore the Riesz representation g of $\ell_\gamma(\cdot)$, see (14), is in the case of the separable input $\boldsymbol{\Phi}$ given in (8),

$$g = \sum_{m=1}^{M_\phi} \beta_m \phi_\gamma^m. \tag{17}$$

Thus, the functional to be minimized reads $\mathcal{J}(f_\gamma) = \alpha\, a_\gamma(f_\gamma,f_\gamma) - 2\ell_\gamma(f_\gamma)$, and the solution f_γ of this sectional problem is such that for all $f^\star \in V_\gamma$

$$\alpha\, a_\gamma(f_\gamma,f^\star) = \ell_\gamma(f^\star) = a_\gamma(g,f^\star) \tag{18}$$

Problem (18) is linear and sectional, and consists in identifying the first arguments of the bilinear form, that is it reduces to

$$f_\gamma = \frac{1}{\alpha} g. \tag{19}$$

2.3 PGD Greedy Modal Updating

The PGD strategy consists in successively computing rank-one approximations, aiming at each step to better fit the unresolved part of the function to be approximated, $\boldsymbol{\Phi}$. Thus, once a rank-M approximation F is obtained, see (1) or its algebraic counterpart (2), the next step is to obtain $\delta F = \prod_{i=1}^{\mathtt{n_d}} \delta f_i$ as the rank-one approximation of $\boldsymbol{\Phi} - F$. Then, the updated function $F + \delta F$ is taken as the rank-$M+1$ approximation. This is equivalent to taking $f_i^{M+1} = \delta f_i$, for $i = 1, \dots, \mathtt{n_d}$.

The computation of δF as the rank-one approximation of $\boldsymbol{\Phi} - F$ is performed using the methodology presented in Sect. 2.2. This strategy of computing each term finding the optimal at each stage is classifying the PGD approach as a greedy algorithm. It consists in repeatedly finding rank-one approximations of the remaining residual.

At this stage, an important point in the algorithm is how to select a proper stopping criterion to decide the number of terms that provide a fair enough approximation and how to implement it. The simplest strategy consists in controlling the *amplitude* of each term and to truncate the sum when the amplitude of the current mode is significantly lower than the first one.

This requires introducing the normalized version of each sectional mode, that is replacing f_i by $f_i^m / \|f_i^m\|$, for $i = 1, \dots, \mathtt{n_d}$, $m = 1, \dots, M$, being $\|f_i^m\| = \sqrt{a_i(f_i^m, f_i^m)}$. The norms dividing each sectional mode are accumulated in the amplitude of each term, $\sigma_m = \prod_{i=1}^{\mathtt{n_d}} \|f_i^m\|$.

Thus, the expression for the separated approximation (1) is rewritten in a normalized version as

$$F(x_1, x_2, \dots, x_{\mathtt{n_d}}) = \sum_{m=1}^{M} \sigma_m \prod_{i=1}^{\mathtt{n_d}} f_i^m(x_i). \tag{20}$$

This allows deciding when to truncate the PGD expansion in the basis of the evolution of the amplitudes. For example, a common strategy is to keep computing terms while $\sigma_m \geq \eta^\star \sigma_1$, for some tolerance $\eta^\star$ setting the expected accuracy in the PGD truncation: the lower is $\eta^\star$, the larger the number of PGD terms, M, is expected.

2.4 Tensorial Version of Least-Squares PGD

As stated in Sect. 1.1, the functional and tensorial format for multidimensional data are equivalent once the functional space is discretized. Thus, the algorithm devised in the previous section is readily adapted to deal with multidimensional data in tensorial format. In this case, the input data is a tensor $\boldsymbol{\Phi} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_{\mathtt{n_d}}}$ to be approximated by a tensor $\boldsymbol{F}$ in the same space and expressed in a separable format, as indicated in Eq. (2).

The sectional bilinear forms $a_i(\cdot,\cdot)$, $i = 1, \dots, \mathtt{n_d}$ are represented in the tensorial format by their discrete versions in the interpolation spaces, that is by matrices $\boldsymbol{A}_i \in \mathbb{R}^{n_i \times n_i}$. Thus, the application of the bilinear form to functions is identified by the scalar product of vectors, namely

$$a_i(f_i,f_i^\star) = \boldsymbol{f}_i^{\mathsf{T}} \boldsymbol{A}_i \boldsymbol{f}_i^\star$$

being $\boldsymbol{f}_i$ and $\boldsymbol{f}_i^\star$ the vectors in $\mathbb{R}^{n_i}$ representing f_i and $f_i^\star$.

The conceptual steps of the methodology are identical. In the following, the main points of the procedure devised in Sects. 2.2 and 2.3 are revisited in their tensorial forms.

Note that the aim of the algorithm is producing a separable tensor $\boldsymbol{F}$ fairly approximating $\boldsymbol{\Phi}$, as in (2). The normalized form, analogous to (20), is preferred to control the importance of the subsequent terms and also to compare the successive iterations in the alternating directions loop. This form reads

$$\begin{aligned} \boldsymbol{F} &= \sum_{m=1}^{M} \sigma_m \boldsymbol{f}_1^m \otimes \boldsymbol{f}_2^m \otimes \cdots \otimes \boldsymbol{f}_{\mathtt{n_d}}^m \\ &= \sum_{m=1}^{M} \sigma_m \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{f}_j^m \end{aligned} \tag{21}$$

where $\boldsymbol{f}_i$, $i = 1, \dots, \mathtt{n_d}$, are unit vectors and the multi-tensorial product is introduced to shorten the notation.

The scalar α introduced in (12) is in the tensorial format defined as

$$\alpha := \prod_{j\neq\gamma}^{\mathtt{n_d}} \boldsymbol{f}_j^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j. \tag{22}$$

Let us denote by $\boldsymbol{g} \in \mathbb{R}^{n_\gamma}$ the discrete representation of the linear form $\ell_\gamma(\cdot)$. Equation (13) is rewritten as

$$A(\boldsymbol{F}, \boldsymbol{\Phi}) = A\left(\left[\bigotimes_{j\neq\gamma} \boldsymbol{f}_j\right] \otimes \boldsymbol{f}_\gamma, \boldsymbol{\Phi}\right) = \boldsymbol{g}^{\mathsf{T}} \boldsymbol{A}_\gamma \boldsymbol{f}_\gamma, \tag{23}$$

where vector $\boldsymbol{g}$ is the discrete representation of the linear form $\ell_\gamma(\cdot)$ in (13). Recall that in the iteration loop, the unknown is $\boldsymbol{f}_\gamma$ and all the rest ($\boldsymbol{f}_j$ for $j \neq \gamma$) are considered to be known. Thus, vector $\boldsymbol{g}$ is the result of contracting all the indices of tensor $\boldsymbol{\Phi}$ but the γ-th one with tensor $\bigotimes_{j\neq\gamma} \boldsymbol{f}_j$ and using the sectional matrices $\boldsymbol{A}_j$ in each dimension. This results in computing $\boldsymbol{g}$ such that

$$\boldsymbol{g} = \boldsymbol{\Phi} \;\vdots\; \bigotimes_{j\neq\gamma} \left[\boldsymbol{A}_j \boldsymbol{f}_j\right] \tag{24}$$

this is written in a compact form using symbol $\vdots$, that indicates tensor contraction of all possible indices. In this case, provided that $\boldsymbol{\Phi}$ is a tensor of $\mathtt{n_d}$ dimensions and $\bigotimes_{j\neq\gamma}^{\mathtt{n_d}} \boldsymbol{f}_j$ is a tensor of $\mathtt{n_d} - 1$ dimensions, this means summing up in all indices i_j for $j = 1, \ldots, \mathtt{n_d}$ with $j \neq \gamma$. The expansion in terms of all the indices of the expression (24) is such that each component $[\boldsymbol{g}]_{i_\gamma}$, for $i_\gamma = 1, \ldots, n_\gamma$ of $\boldsymbol{g}$ reads

$$\begin{aligned}[\boldsymbol{g}]_{i_\gamma} = &\sum_{i_1=1}^{n_1} \cdots \sum_{i_{\gamma-1}=1}^{n_{\gamma-1}} \sum_{i_{\gamma+1}=1}^{n_{\gamma+1}} \\ &\cdots \sum_{i_{\mathtt{n_d}}=1}^{\mathtt{n_d}} [\boldsymbol{\Phi}]_{i_1 \ldots i_{\gamma-1} i_\gamma i_{\gamma+1} \ldots i_{\mathtt{n_d}}} \prod_{j\neq\gamma}^{\mathtt{n_d}} \left[\boldsymbol{A}_j \boldsymbol{f}_j\right]_{i_j}\end{aligned} \tag{25}$$

Then, the solution analogous to (19) is

$$\boldsymbol{f}_\gamma = \frac{1}{\alpha} \boldsymbol{g} \tag{26}$$

As noted in (15), in the case $\boldsymbol{\Phi}$ is already available in a separable format, the expressions are simpler. Indeed, the separable version of $\boldsymbol{\Phi}$ analogous to (8) reads

$$\boldsymbol{\Phi} = \sum_{m=1}^{M_\phi} \boldsymbol{\phi}_1^m \otimes \boldsymbol{\phi}_2^m \otimes \cdots \otimes \boldsymbol{\phi}_{\mathtt{n_d}}^m \tag{27}$$

And the coefficients β_m defined in (15) are defined now by

$$\beta_m = \prod_{j\neq\gamma} \boldsymbol{f}_j^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{\phi}_j^m, \quad \text{for } m = 1, \ldots, M_\phi, \quad \text{and for some } \gamma \tag{28}$$

In this particular situation (for a separable $\boldsymbol{\Phi}$ that has to be compressed), the computation of $\boldsymbol{f}_\gamma$, analogous to (17) is straightforwardly given by taking

$$\boldsymbol{g} = \sum_{m=1}^{M_\phi} \beta_m \boldsymbol{\phi}_\gamma^m. \tag{29}$$

As indicated in Sect. 2.3, in the greedy loop (loop in the number of PGD terms, m) the function to be approximated is not $\boldsymbol{\Phi}$ but the remaining residual, that is

$$\boldsymbol{\Phi} - \sum_{\tilde{m}=1}^{m-1} \sigma_{\tilde{m}} \boldsymbol{f}_j^{\tilde{m}}.$$

In the case of the PGD separation, when $\boldsymbol{\Phi}$ is a general tensor, the computation of $\boldsymbol{g}$ has to be modified and, instead of (24), one has

$$\begin{aligned}\boldsymbol{g} &= \left(\boldsymbol{\Phi} - \sum_{\tilde{m}=1}^{m-1} \sigma_{\tilde{m}} \boldsymbol{f}_j^{\tilde{m}}\right) \;\vdots\; \bigotimes_{j\neq\gamma} \left[\boldsymbol{A}_j \boldsymbol{f}_j\right] \\ &= \boldsymbol{\Phi} \;\vdots\; \bigotimes_{j\neq\gamma} \left[\boldsymbol{A}_j \boldsymbol{f}_j\right] - \sum_{\tilde{m}=1}^{m-1} \sigma_{\tilde{m}} \left(\prod_{j\neq\gamma} \boldsymbol{f}_j^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\tilde{m}}\right) \boldsymbol{f}_\gamma^{\tilde{m}}\end{aligned} \tag{30}$$

For a separated $\boldsymbol{\Phi}$ (PGD compression), taking this into the account consists in adding more terms to the description of the function to be approximated. Thus, in the iteration for $\boldsymbol{f}_\gamma$, (29)$\boldsymbol{g}$ has to be modified to accounting for previously computed terms, that is

$$\boldsymbol{g} = \sum_{m=1}^{M_\phi} \beta_m \boldsymbol{\phi}_\gamma^m - \sum_{\tilde{m}=1}^{m-1} \sigma_{\tilde{m}} \left(\prod_{j\neq\gamma} \boldsymbol{f}_j^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\tilde{m}}\right) \boldsymbol{f}_\gamma^{\tilde{m}} \tag{31}$$

Algorithm 1 describes the full procedure for the case of PGD compression (applied to a separable $\boldsymbol{\Phi}$).

Note that Algorithm 1 includes two different tolerances, η and $\eta^\star$. These tolerances are used for different purposes. The value of $\eta^\star$ is used to set the stopping criterion of the greedy algorithm, that is to decide when to stop the m-loop, as indicated in Sect. 2.3. Instead, η is used as the criterion to stop the γ-iterations (alternating-directions scheme). This assesses both the difference between to successive amplitudes and the norm of the difference of the normalized sectional modes.

Remark 5 (*Differences with tensor separation, with nonseparable $\boldsymbol{\Phi}$*) The algorithm for PGD separation is very similar to Algorithm 1 for PGD compression. The only difference is to compute $\boldsymbol{g}$ using (30) instead of (31).

Remark 6 (*Selecting different least-squares criteria for each dimension*) Most commonly, the sectorial bilinear forms $a_i(\cdot,\cdot)$ are taken as standard $\mathcal{L}_2$ products in each sectorial space Ω_i. This translates in the discrete format in taking the sectional matrices $\boldsymbol{A}_i$ equal to the standard mass matrices associated with the discretization. Note that, for a 1D sectional space $\Omega_i =]a_i, b_i[$, this is a simple tridiagonal matrix. In the case of being $\boldsymbol{\Phi}$ a genuinely algebraic tensor (and not the

discretization of a multidimensional function), the Euclidean norm is obtained by taking $\boldsymbol{A}_i$ equal to the identity matrix.

In some cases, prior information of the function $\boldsymbol{\Phi}$ to be approximated may suggest alternative choices for the projection criteria. Thus, different sectional Gram matrices $\boldsymbol{A}_i$ could be adopted to preserve some features of the approximation.

Algorithm 1: PGD compression. Least-squares PGD approximation for multidimensional tensor $\boldsymbol{\Phi}$ expressed in separable form.

Data: Tensor of order $\mathtt{n_d}$ to be approximated: $\boldsymbol{\Phi}$ in separable form $\boldsymbol{\Phi} = \sum_{\hat{m}=1}^{M_\phi} \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\phi}_j^{\hat{m}}$
Tolerances $0 < \eta \ll 1$ for alternating directions and $0 < \eta^\star \ll 1$ for greedy algorithm
Sectional Gram matrices $\boldsymbol{A}_j$, for $j = 1, 2, \ldots, \mathtt{n_d}$
Result: PGD approximation:
$\boldsymbol{F}_{\text{PGD}} = \sum_{m=1}^{M} \sigma_m \boldsymbol{f}_1^m \otimes \boldsymbol{f}_2^m \otimes \cdots \otimes \boldsymbol{f}_{\mathtt{n_d}}^m \approx \boldsymbol{\Phi}$
Initialize counter of PGD terms $m = 1$ and starting value of amplitude $\sigma_1 = 1$
while $\sigma_m > \eta^\star \sigma_1$ **do**
  Initialize: assign values to $\boldsymbol{f}_j^{\mathtt{old}}$, for $j = 1, 2, \ldots, \mathtt{n_d}$;
  Compute $\sigma^{\mathtt{old}} = \prod_{j=1}^{\mathtt{n_d}} \left[\boldsymbol{f}_j^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}\right]^{1/2}$;
  Normalize: $\boldsymbol{f}_j^{\mathtt{old}} \leftarrow \left[\boldsymbol{f}_j^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}\right]^{-1/2} \boldsymbol{f}_j^{\mathtt{old}}$, for $j = 1, 2, \ldots, \mathtt{n_d}$;
  while $\varepsilon > \eta$ or $\varepsilon_\sigma > \eta$ **do**
    Initialize: $\varepsilon = 1$; $\varepsilon_\sigma = 1$; $\sigma^{\mathtt{new}} = 1$;
    for $\gamma = 1 \ldots \mathtt{n_d}$ **do**
      Compute $\alpha = \prod_{j \neq \gamma} \boldsymbol{f}_j^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}$
      Compute $\beta_{\hat{m}} = \prod_{j \neq \gamma} \boldsymbol{f}_j^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{\phi}_j^{\hat{m}}$ for $\hat{m} = 1, \ldots, M_\phi$
      Compute
      $\boldsymbol{g} = \sum_{\hat{m}=1}^{M_\phi} \beta_{\hat{m}} \boldsymbol{\phi}_\gamma^{\hat{m}} - \sum_{\tilde{m}=1}^{m-1} \sigma_{\tilde{m}} \left(\prod_{j \neq \gamma} \boldsymbol{f}_j^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\tilde{m}} \right) \boldsymbol{f}_\gamma^{\tilde{m}}$
      and $\boldsymbol{f}_\gamma^{\mathtt{new}} = \frac{1}{\alpha} \boldsymbol{g}$
      Compute norm $\sigma_\gamma^{\mathtt{new}} = \left[\boldsymbol{f}_\gamma^{\mathtt{new}\,\mathsf{T}} \boldsymbol{A}_\gamma \boldsymbol{f}_\gamma^{\mathtt{new}}\right]^{1/2}$;
      Update and Normalize: $\sigma^{\mathtt{new}} \leftarrow \sigma^{\mathtt{new}} \sigma_\gamma^{\mathtt{new}}$;
      $\boldsymbol{f}_\gamma^{\mathtt{new}} \leftarrow \frac{1}{\sigma_\gamma^{\mathtt{new}}} \boldsymbol{f}_\gamma^{\mathtt{new}}$
      Update modal error
      $\varepsilon \leftarrow \varepsilon \left[(\boldsymbol{f}_\gamma^{\mathtt{new}} - \boldsymbol{f}_\gamma^{\mathtt{old}})^{\mathsf{T}} \boldsymbol{A}_\gamma (\boldsymbol{f}_\gamma^{\mathtt{new}} - \boldsymbol{f}_\gamma^{\mathtt{old}})\right]^{1/2}$;
      Update $\boldsymbol{f}_\gamma^{\mathtt{old}} \leftarrow \boldsymbol{f}_\gamma^{\mathtt{new}}$
    end for
    Compute amplitude error $\varepsilon_\sigma = \left|\sigma^{\mathtt{new}} - \sigma^{\mathtt{old}}\right| / \left|\sigma^{\mathtt{new}}\right|$;
    Update $\sigma^{\mathtt{old}} \leftarrow \sigma^{\mathtt{new}}$;
  end while
  Store: $\sigma_m \leftarrow \sigma^{\mathtt{new}}$; $\boldsymbol{f}_j^m \leftarrow \boldsymbol{f}_j^{\mathtt{new}}$, for $i = 1, 2, \ldots, \mathtt{n_d}$;
  Update $m \leftarrow m + 1$;
end while
Store: $M \leftarrow m$;

3 Operating with Separated Representations

The methodology presented in Sect. 2.4 and illustrated in Algorithm 1 provides separable approximations of multidimensional arrays. This is aiming at separating a multidimensional tensor $\boldsymbol{\Phi}$ in general format or compressing (into a separated expression with less terms) a tensor already expressed in a separated format.

The current section is devoted to devise strategies to operate with this type of separable objects. The sum or the product of two separated tensors are straightforward. Let us assume that the input data are tensors $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$, both in $\mathbb{R}^{n_1 \times \cdots \times n_{\mathtt{n_d}}}$, given by

$$\boldsymbol{\Phi} = \sum_{\hat{m}=1}^{M_\phi} \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\phi}_j^{\hat{m}} \quad \text{and} \quad \boldsymbol{\Psi} = \sum_{\tilde{m}=1}^{M_\psi} \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\psi}_j^{\tilde{m}}. \tag{32}$$

The sum $\boldsymbol{\Phi} + \boldsymbol{\Psi}$ and the Hadamard product $\boldsymbol{\Phi} \cdot \boldsymbol{\Psi}$ (component by component product, corresponding to the command "`.*`" in `matlab`) result in separated tensors with $M_\phi + M_\psi$ and $M_\phi M_\psi$ terms, respectively. The different terms in the resulting expressions, in particular with the product, may include some redundant information. Therefore, the PGD compression strategy devised above is to be eventually applied to reduce the number of terms.

Thus, we select as illustrative examples of nontrivial operations, the division term by term of two tensors expressed in their separable formats (Hadamard division) and the solution of linear systems of equations corresponding to parametrized problems.

3.1 Hadamard Division

The Hadamard division is a simple operation between two multidimensional tensors, $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ as defined in (32). It is denoted by $\boldsymbol{\Phi} \oslash \boldsymbol{\Psi}$ and consists in a component by component division. This operation is performed in `matlab` by command "`./`".

The operation is extremely simple if the full tensor is available in its multidimensional format. The aim of this section is to devise an algorithm to perform this operation between to tensors expressed in the separable format, without reconstructing the full tensor. Thus, the goal is to obtain a separable approximation of

$$\boldsymbol{F} = \boldsymbol{\Phi} \oslash \boldsymbol{\Psi} \tag{33}$$

in a PGD fashion.

The rationale of the derivation of the algorithm is similar to the PGD separation and compression described in Sect. 2. First, it is noted that $\boldsymbol{F}$ is such that

$$\boldsymbol{F} \odot \boldsymbol{\Psi} = \boldsymbol{\Phi} \tag{34}$$

and a rank-one approximation with the form $\boldsymbol{F} = \boldsymbol{f}_1^m \otimes \boldsymbol{f}_2^m \otimes \cdots \otimes \boldsymbol{f}_{\mathtt{n_d}}^m$ is sought.

The rank-one problem is solved with alternating directions iterative scheme, that is computing $\boldsymbol{f}_\gamma$, for $\gamma = 1, \ldots, \mathtt{n_d}$, while assuming that the rest of terms ($\boldsymbol{f}_j$ for $j \neq \gamma$) are known. The equation for $\boldsymbol{f}_\gamma$ is found multiplying the two sides of (34) by a *test* tensor $\boldsymbol{F}^\star = \left[\bigotimes_{j\neq\gamma} \boldsymbol{f}_j\right] \otimes \boldsymbol{f}_\gamma^\star$ and enforcing equation

$$A(\boldsymbol{F} \odot \boldsymbol{\Psi} - \boldsymbol{\Phi}, \boldsymbol{F}^\star) = 0 \tag{35}$$

for any possible value of $\boldsymbol{f}_\gamma^\star$. Equation (35) results in computing $\boldsymbol{f}_\gamma$ with the expression

$$\boldsymbol{f}_\gamma = \left[\sum_{\hat{m}=1}^{M_\phi} \beta_{\hat{m}} \boldsymbol{\phi}_\gamma^{\hat{m}}\right] \oslash \left[\sum_{\tilde{m}=1}^{M_\psi} \alpha_{\tilde{m}} \boldsymbol{\psi}_\gamma^{\tilde{m}}\right] \tag{36}$$

where the coefficients $\alpha_{\tilde{m}}$, for $\tilde{m} = 1, \ldots, M_\psi$ and $\beta_{\hat{m}}$, for $\hat{m} = 1, \ldots, M_\phi$ are given by

$$\alpha_{\tilde{m}} = \prod_{j\neq\gamma} \left[\left(\boldsymbol{\psi}_j^{\tilde{m}} \odot \boldsymbol{f}_j\right)^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j\right], \quad \text{and}$$

$$\beta_{\hat{m}} = \prod_{j\neq\gamma} \left[\boldsymbol{\phi}_j^{\hat{m}\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j\right]$$

Analogously to Sect. 2.4, this rank-one solver is embedded in a greedy loop (for $m = 1, 2, \ldots$). The idea is that the approximation with $m-1$ terms $\boldsymbol{F}^{m-1}$ is available and this is to be updated by $\delta\boldsymbol{F}$ such that $\boldsymbol{F}^m = \boldsymbol{F}^{m-1} + \delta\boldsymbol{F}$. The remaining residual unknown $\delta\boldsymbol{F}$ is the solution of

$$\delta\boldsymbol{F} \odot \boldsymbol{\Psi} = \boldsymbol{\Phi} - \boldsymbol{F}^{m-1} \odot \boldsymbol{\Psi} \tag{37}$$

And a rank-one approximation of the solution of (37) in the form $\delta\boldsymbol{F} = \bigotimes_{j=1}^{\mathtt{n_d}} \delta\boldsymbol{f}_j$ is sought. Note that this requires just modifying the right-hand-side of (34), replacing $\boldsymbol{\Phi}$ by $\boldsymbol{\Phi} - \boldsymbol{F}^{m-1} \odot \boldsymbol{\Psi}$.

Algorithm 2 describes how to obtain the Hadamard division of tensors $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$. A first version of this algorithm was included in [3, 10, 11] as part of a PGD solver for parametric nonlinear systems of equations.

Algorithm 2: Hadamard division. Least-squares PGD approximation of multidimensional tensor $\boldsymbol{\Phi} \oslash \boldsymbol{\Psi}$.

Data: Tensors $\boldsymbol{\Phi} = \sum_{\hat{m}=1}^{M_\phi} \bigotimes_{j=1}^{\mathtt{n_d}} \phi_j^{\hat{m}}$ and $\boldsymbol{\Psi} = \sum_{\tilde{m}=1}^{M_\psi} \bigotimes_{j=1}^{\mathtt{n_d}} \psi_j^{\tilde{m}}$
Tolerances $0 < \eta \ll 1$ for alternating directions and $0 < \eta^\star \ll 1$ for greedy algorithm
Sectional Gram matrices $\boldsymbol{A}_j$, for $j = 1, 2, \ldots, \mathtt{n_d}$
Result: PGD approximation:
$\boldsymbol{F}_{\text{PGD}} = \sum_{m=1}^{M} \sigma_m \boldsymbol{f}_1^m \otimes \boldsymbol{f}_2^m \otimes \cdots \otimes \boldsymbol{f}_{\mathtt{n_d}}^m \approx \boldsymbol{\Phi} \oslash \boldsymbol{\Psi}$
Initialize counter of PGD terms $m = 1$ and starting value of amplitude $\sigma_1 = 1$
while $\sigma_m > \eta^\star \sigma_1$ **do**
  Initialize: assign values to $\boldsymbol{f}_j^{\mathtt{old}}$, for $j = 1, 2, \ldots, \mathtt{n_d}$;
  Compute $\sigma^{\mathtt{old}} = \prod_{j=1}^{\mathtt{n_d}} \left[\boldsymbol{f}_j^{\mathtt{old}\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}\right]^{1/2}$;
  Normalize: $\boldsymbol{f}_j^{\mathtt{old}} \leftarrow \left[\boldsymbol{f}_j^{\mathtt{old}\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}\right]^{-1/2} \boldsymbol{f}_j^{\mathtt{old}}$, for $j = 1, 2, \ldots, \mathtt{n_d}$;
  while $\varepsilon > \eta$ or $\varepsilon_\sigma > \eta$ **do**
    Initialize $\varepsilon = 1$; $\varepsilon_\sigma = 1$; $\sigma^{\mathtt{new}} = 1$;
    for $\gamma = 1 \ldots \mathtt{n_d}$ **do**
      Compute $\alpha_{\tilde{m}} = \prod_{j\neq\gamma} \left[\left(\psi_j^{\tilde{m}} \odot \boldsymbol{f}_j^{\mathtt{old}}\right)^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}\right]$
      Compute $\beta_{\hat{m}} = \prod_{j\neq\gamma} \left[\phi_j^{\hat{m}\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}}\right]$ for $\hat{m} = 1, \ldots, M_\phi$
      Compute $\boldsymbol{g} = \sum_{\hat{m}=1}^{M_\phi} \beta_{\hat{m}} \phi_\gamma^{\hat{m}} - \sum_{\tilde{m}=1}^{M_\psi} \sum_{k=1}^{m-1} \sigma_{\tilde{m}} \left[\prod_{j\neq\gamma} \boldsymbol{f}_j^{\mathtt{old}\mathsf{T}} \boldsymbol{A}_j \left(\psi_j^{\tilde{m}} \odot \boldsymbol{f}_j^{\mathtt{old}}\right)\right] \boldsymbol{f}_\gamma^k$
      Compute $\boldsymbol{a} = \sum_{\tilde{m}=1}^{M_\psi} \alpha_{\tilde{m}} \psi_\gamma^{\tilde{m}}$
      Compute $\boldsymbol{f}_\gamma^{\mathtt{new}} = \boldsymbol{g} \oslash \boldsymbol{a}$
      Compute norm $\sigma_\gamma^{\mathtt{new}} = \left[\boldsymbol{f}_\gamma^{\mathtt{new}\mathsf{T}} \boldsymbol{A}_\gamma \boldsymbol{f}_\gamma^{\mathtt{new}}\right]^{1/2}$;
      Update and Normalize: $\sigma^{\mathtt{new}} \leftarrow \sigma^{\mathtt{new}} \sigma_\gamma^{\mathtt{new}}$;
      $\boldsymbol{f}_\gamma^{\mathtt{new}} \leftarrow \frac{1}{\sigma_\gamma^{\mathtt{new}}} \boldsymbol{f}_\gamma^{\mathtt{new}}$;
      Update modal error
      $\varepsilon \leftarrow \varepsilon \left[(\boldsymbol{f}_\gamma^{\mathtt{new}} - \boldsymbol{f}_\gamma^{\mathtt{old}})^{\mathsf{T}} \boldsymbol{A}_\gamma (\boldsymbol{f}_\gamma^{\mathtt{new}} - \boldsymbol{f}_\gamma^{\mathtt{old}})\right]^{1/2}$;
      Update $\boldsymbol{f}_\gamma^{\mathtt{old}} \leftarrow \boldsymbol{f}_\gamma^{\mathtt{new}}$
    end for
    Compute amplitude error $\varepsilon_\sigma = \left|\sigma^{\mathtt{new}} - \sigma^{\mathtt{old}}\right| / \left|\sigma^{\mathtt{new}}\right|$;
    Update $\sigma^{\mathtt{old}} \leftarrow \sigma^{\mathtt{new}}$;
  end while
  Store: $\sigma_m \leftarrow \sigma^{\mathtt{new}}$; $\boldsymbol{f}_j^m \leftarrow \boldsymbol{f}_j^{\mathtt{new}}$, for $i = 1, 2, \ldots, \mathtt{n_d}$;
  Update $m \leftarrow m + 1$;
end while
Store: $M \leftarrow m$;

3.2 Parametric Linear System of Equations

The Hadamard division analyzed in the previous section is just an example of a complex operation that can be performed with two separable input tensors. As mentioned above, this is used as key point of PGD for a nonlinear solver. Another complex operation, extremely useful in the context of obtaining Computational Vademecums with PGD, is the solution of algebraic linear systems of equations with parametric dependence. A particular version of this algorithm is devised in [18] to solve parametric lattice structures in linear regime.

In a continuous setting (that is in a continuous description of the parametric dependence), the problem reads: find vector $\boldsymbol{u}(\boldsymbol{\mu})$ such that

$$\boldsymbol{K}(\boldsymbol{\mu})\boldsymbol{u}(\boldsymbol{\mu}) = \boldsymbol{b}(\boldsymbol{\mu}) \tag{38}$$

that is, being the solution of a $n \times n$ linear system of algebraic equations depending on some parameter $\boldsymbol{\mu}$.

It is assumed that input matrix $\boldsymbol{K}(\boldsymbol{\mu})$ and input vector $\boldsymbol{b}(\boldsymbol{\mu})$ are expressed in a separable fashion. Thus, the PGD solver for linear systems of equations is devised to obtain a separable expression of the parametric dependence of the unknown $\boldsymbol{u}(\boldsymbol{\mu})$. Obviously, this is to be carried out without reconstructing the full multidimensional parametric dependence, and avoiding the *curse of dimensionality*.

Following the rationale of the previous section, we present the methodology in a discrete format. Thus, input matrix $\boldsymbol{K}(\boldsymbol{\mu})$ is given as a tensor $\mathbb{K} \in \mathbb{R}^{n\times n\times n_1\times\cdots\times n_{\mathtt{n_d}}}$

$$\mathbb{K} = \sum_{\hat{m}=1}^{M_\phi} \boldsymbol{K}^{\hat{m}} \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\phi}_j^{\hat{m}} \tag{39}$$

where (non parametric) matrices $\boldsymbol{K}^{\hat{m}} \in \mathbb{R}^{n\times n}$, for $\hat{m} = 1, \dots, M_\phi$ are complemented to a full parametric tensor dependency by the set of vectors $\boldsymbol{\phi}_j^{\hat{m}} \in \mathbb{R}^{n_j}$, $j = 1, \dots, \mathtt{n_d}$. Vectors $\boldsymbol{\phi}_j^{\hat{m}}$ are the discrete representations of sectional modes in the j-th sectional dimension, that is some functions $\phi_j^{\hat{m}}(\mu_j)$.

Similarly, input vector $\boldsymbol{b}(\boldsymbol{\mu})$ is given as a tensor $\boldsymbol{B} \in \mathbb{R}^{n\times n_1\times\cdots\times n_{\mathtt{n_d}}}$

$$\boldsymbol{B} = \sum_{\tilde{m}=1}^{M_\psi} \boldsymbol{b}^{\tilde{m}} \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\psi}_j^{\tilde{m}} \tag{40}$$

where (non parametric) vector $\boldsymbol{b}^{\tilde{m}} \in \mathbb{R}^n$, for $\tilde{m} = 1, \dots, M_\psi$ are complemented by vectors $\boldsymbol{\psi}_j^{\tilde{m}} \in \mathbb{R}^{n_j}$, $j = 1, \dots, \mathtt{n_d}$.

Then, the tensorial equation to solve is

$$\mathbb{K} \cdot \boldsymbol{U} = \boldsymbol{B} \tag{41}$$

and the solution sought in separated format is written as

$$\boldsymbol{U} = \sum_{m=1}^{M} \boldsymbol{u}^m \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{f}_j^m. \tag{42}$$

Note that the product between $\mathbb{K}$ and $\boldsymbol{U}$ in (41) must be understood in the sense that it provides a tensor representation of the parametric equation (38). This corresponds to the following definition of $\mathbb{K} \cdot \boldsymbol{U}$, taking the expressions for $\mathbb{K}$ and $\boldsymbol{U}$ given in (39) and (42):

$$\mathbb{K} \cdot \boldsymbol{U} = \sum_{m=1}^{M} \sum_{\hat{m}=1}^{M_\phi} \left[\boldsymbol{K}^{\hat{m}} \cdot \boldsymbol{u}^m\right] \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \left[\boldsymbol{\phi}_j^{\hat{m}} \odot \boldsymbol{f}_j^m\right]. \tag{43}$$

The product in parametric sectional dimensions is of Hadamard type because it represents the product of sectional modes. In other words, $\boldsymbol{\phi}_j^{\hat{m}} \odot \boldsymbol{f}_j^m$ represents $\phi_j^{\hat{m}}(\mu_j) f_j^m(\mu_j)$. The behavior in the *algebraic* dimension, is different because it actually represents the standard matrix-vector product.

The first rank-one approximation, that is taking

$$\boldsymbol{U} \approx \boldsymbol{u} \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{f}_j \tag{44}$$

that better approximates $\mathbb{K}^{-1} \cdot \boldsymbol{B}$ is considered in detail to illustrate Algorithm 3 and its main aspects.

As in the previous example, the rank-one problem is solved with alternating directions iterative scheme. In this case, we have two types of iterations.

3.2.1 Parametric Rank-One Iterations

First, the standard parametric sectional problem consists in computing $\boldsymbol{f}_\gamma$ in (44), for $\gamma = 1, \dots, \mathtt{n_d}$, while assuming that the rest of terms ($\boldsymbol{u}$ and $\boldsymbol{f}_j$ for $j \neq \gamma$) are known. The equation for $\boldsymbol{f}_\gamma$ is found multiplying the two sides of (41) by a *test* tensor $\boldsymbol{U}^\star = \boldsymbol{u} \otimes \left[\bigotimes_{j\neq\gamma} \boldsymbol{f}_j\right] \otimes \boldsymbol{f}_\gamma^\star$ and enforcing equation

$$A(\mathbb{K} \cdot \boldsymbol{U} - \boldsymbol{B}, \boldsymbol{U}^\star) = 0 \tag{45}$$

for all possible $\boldsymbol{f}_\gamma^\star$.

Note that for the rank-one solution in (44)

$$\mathbb{K}\cdot\boldsymbol{U}=\sum_{\hat{m}=1}^{M_\phi}\left[\boldsymbol{K}^{\hat{m}}\cdot\boldsymbol{u}\right]\otimes\bigotimes_{j=1}^{\mathtt{n_d}}\left[\boldsymbol{\phi}_j^{\hat{m}}\odot\boldsymbol{f}_j\right] \tag{46}$$

therefore

$$\begin{aligned}A(\mathbb{K}\cdot\boldsymbol{U},\boldsymbol{U}^\star)&\\ =\sum_{\hat{m}=1}^{M_\phi}&\underbrace{\left[\boldsymbol{u}^\mathsf{T}\boldsymbol{K}^{\hat{m}\,\mathsf{T}}\boldsymbol{A}_0\boldsymbol{u}\right]\prod_{j\neq\gamma}\left[\left(\boldsymbol{\phi}_j^{\hat{m}}\odot\boldsymbol{f}_j\right)^\mathsf{T}\boldsymbol{A}_j\boldsymbol{f}_j\right]}_{\alpha_{\hat{m}}}\\ &\cdots\left[\left(\boldsymbol{\phi}_\gamma^{\hat{m}}\odot\boldsymbol{f}_\gamma\right)^\mathsf{T}\boldsymbol{A}_\gamma\boldsymbol{f}_\gamma^\star\right]\end{aligned} \tag{47}$$

where $\boldsymbol{A}_0$ stands for the matrix describing the *space* sectional dimension of $A(\cdot,\cdot)$ (corresponding to $\boldsymbol{u}$). Similarly

$$\begin{aligned}A(\boldsymbol{B},\boldsymbol{U}^\star)=\sum_{\tilde{m}=1}^{M_\psi}&\underbrace{\left[\boldsymbol{b}^{\tilde{m}\,\mathsf{T}}\boldsymbol{A}_0\boldsymbol{u}\right]\prod_{j\neq\gamma}\left[\boldsymbol{\psi}_j^{\tilde{m}\,\mathsf{T}}\boldsymbol{A}_j\boldsymbol{f}_j\right]}_{\beta_{\tilde{m}}}\\ &\cdots\left[\boldsymbol{\psi}_\gamma^{\tilde{m}\,\mathsf{T}}\boldsymbol{A}_\gamma\boldsymbol{f}_\gamma^\star\right]\end{aligned} \tag{48}$$

Thus, the solution $\boldsymbol{f}_\gamma$ fulfilling (45) for every $\boldsymbol{f}_\gamma^\star$ is

$$\boldsymbol{f}_\gamma=\underbrace{\left[\sum_{\tilde{m}=1}^{M_\psi}\beta_{\tilde{m}}\boldsymbol{\psi}_\gamma^{\tilde{m}}\right]}_{=:\boldsymbol{g}}\oslash\left[\sum_{\hat{m}=1}^{M_\phi}\alpha_{\hat{m}}\boldsymbol{\phi}_\gamma^{\hat{m}}\right]. \tag{49}$$

3.2.2 Space Rank-One Iteration

The iteration for the *space* sectional dimension (the nonparametric sectional dimension) consists in computing $\boldsymbol{u}$ in (44), while assuming that the rest of terms ($\boldsymbol{f}_j$ for $j=1,\dots,\mathtt{n_d}$) are known. In this case the *test* tensor $\boldsymbol{U}^\star=\boldsymbol{u}^\star\otimes\left[\bigotimes_{j=1}^{\mathtt{n_d}}\boldsymbol{f}_j\right]$ is used to enforce Eq. (45), namely,

$$\begin{aligned}&A(\mathbb{K}\cdot\boldsymbol{U},\boldsymbol{U}^\star)\\ &=\sum_{\hat{m}=1}^{M_\phi}\left[\boldsymbol{u}^\mathsf{T}\boldsymbol{K}^{\hat{m}\,\mathsf{T}}\boldsymbol{A}_0\boldsymbol{u}^\star\right]\underbrace{\prod_{j=1}^{\mathtt{n_d}}\left[\left(\boldsymbol{\phi}_j^{\hat{m}}\odot\boldsymbol{f}_j\right)^\mathsf{T}\boldsymbol{A}_j\boldsymbol{f}_j\right]}_{\alpha_{\hat{m}}}\end{aligned} \tag{50}$$

and

$$A(\boldsymbol{B},\boldsymbol{U}^\star)=\sum_{\tilde{m}=1}^{M_\psi}\left[\boldsymbol{b}^{\tilde{m}\,\mathsf{T}}\boldsymbol{A}_0\boldsymbol{u}^\star\right]\underbrace{\prod_{j=1}^{\mathtt{n_d}}\left[\boldsymbol{\psi}_j^{\tilde{m}\,\mathsf{T}}\boldsymbol{A}_j\boldsymbol{f}_j\right]}_{\beta_{\tilde{m}}}. \tag{51}$$

Thus, $\boldsymbol{u}$ fulfilling (45) for every $\boldsymbol{u}^\star$ is the solution of the following $n\times n$ linear system of algebraic equations

$$\left[\sum_{\hat{m}=1}^{M_\phi}\alpha_{\hat{m}}\boldsymbol{K}^{\hat{m}}\right]\boldsymbol{u}=\sum_{\tilde{m}=1}^{M_\psi}\beta_{\tilde{m}}\boldsymbol{b}^{\tilde{m}}=:\boldsymbol{g} \tag{52}$$

3.2.3 Modal Updating

The computation of the sectional iterations proposed in Eqs. (49) and (52) stand for the first term of the PGD expansion in (42) (for $M=1$). For the subsequent terms ($M>1$), one has to account for the terms already computed. In practice, this requires to replace $\boldsymbol{B}$ in (40) by

$$\boldsymbol{B}=\sum_{\tilde{m}=1}^{M_\psi}\boldsymbol{b}^{\tilde{m}}\otimes\bigotimes_{j=1}^{\mathtt{n_d}}\boldsymbol{\psi}_j^{\tilde{m}}-\sum_{m=1}^{M-1}\boldsymbol{u}^m\otimes\bigotimes_{j=1}^{\mathtt{n_d}}\boldsymbol{f}_j^m. \tag{53}$$

This is equivalent to properly modify the numerator in the right-hand-side of (49) and the right-hand-side vector of the linear system (52), both denoted by $\boldsymbol{g}$. The resulting operations are indicated in Algorithm 3.

Algorithm 3: Linear system of equations. PGD approximation of multidimensional tensor $\boldsymbol{U} = \mathbb{K}^{-1} \cdot \boldsymbol{B}$.

Data: Tensors $\mathbb{K} = \sum_{\hat{m}=1}^{M_\phi} \boldsymbol{K}^{\hat{m}} \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\phi}_j^{\hat{m}}$ and $\boldsymbol{B} = \sum_{\tilde{m}=1}^{M_\psi} \boldsymbol{b}^{\tilde{m}} \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{\psi}_j^{\tilde{m}}$
Tolerances and Sectional Gram matrices $\boldsymbol{A}_j$, for $j = 0, 1, \ldots, \mathtt{n_d}$
Result: PGD approximation: $\boldsymbol{U} = \sum_{m=1}^{M} \sigma_m \boldsymbol{u}^m \otimes \bigotimes_{j=1}^{\mathtt{n_d}} \boldsymbol{f}_j^m \approx \mathbb{K}^{-1} \cdot \boldsymbol{B}$
Initialize counter of PGD terms $m = 1$ and starting value of amplitude $\sigma_1 = 1$
while $\sigma_m > \eta^\star \sigma_1$ **do**
  Initialize: assign values to $\boldsymbol{u}^{\mathtt{old}}$ and $\boldsymbol{f}_j^{\mathtt{old}}$; compute $\sigma^{\mathtt{old}}$; normalize $\boldsymbol{u}^{\mathtt{old}}$ $\boldsymbol{f}_j^{\mathtt{old}}$.
  while $\varepsilon > \eta$ or $\varepsilon_\sigma > \eta$ **do**
    Initialize $\varepsilon = 1$; $\varepsilon_\sigma = 1$; $\sigma^{\mathtt{new}} = 1$;
    %% Space iteration
    Compute $\alpha_{\hat{m}} = \prod_{j=1}^{\mathtt{n_d}} \left[(\boldsymbol{\phi}_j^{\hat{m}} \odot \boldsymbol{f}_j^{\mathtt{old}})^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}} \right]$;
    $\beta_{\tilde{m}} = \prod_{j=1}^{\mathtt{n_d}} \left[\boldsymbol{\psi}_j^{\tilde{m}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}} \right]$
    Compute $\boldsymbol{K} = \sum_{\hat{m}=1}^{M_\phi} \alpha_{\hat{m}} \boldsymbol{K}^{\hat{m}}$;
    $\boldsymbol{g} = \sum_{\tilde{m}=1}^{M_\psi} \beta_{\tilde{m}} \boldsymbol{b}^{\tilde{m}} - \sum_{\ell=1}^{m-1} \sigma_\ell \prod_{j=1}^{\mathtt{n_d}} \left[\boldsymbol{f}_j^{\ell\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^{\mathtt{old}} \right] \boldsymbol{u}^\ell$;
    Compute $\boldsymbol{u}^{\mathtt{new}}$ such that $\boldsymbol{K}\boldsymbol{u}^{\mathtt{new}} = \boldsymbol{g}$
    %% Parametric iterations
    for $\gamma = 1 \ldots \mathtt{n_d}$ **do**
      Compute
      $\alpha_{\hat{m}} = \left[\boldsymbol{u}^{\mathsf{T}} \boldsymbol{K}^{\hat{m}\,\mathsf{T}} \boldsymbol{A}_0 \boldsymbol{u} \right] \prod_{j \neq \gamma} \left[(\boldsymbol{\phi}_j^{\hat{m}} \odot \boldsymbol{f}_j)^{\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j \right]$;
      $\beta_{\tilde{m}} = \left[\boldsymbol{b}^{\tilde{m}\,\mathsf{T}} \boldsymbol{A}_0 \boldsymbol{u} \right] \prod_{j \neq \gamma} \left[\boldsymbol{\psi}_j^{\tilde{m}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j \right]$;
      $\boldsymbol{g} = \sum_{\tilde{m}=1}^{M_\psi} \beta_{\tilde{m}} \boldsymbol{\psi}_\gamma^{\tilde{m}} -$
      $\sum_{\ell=1}^{m-1} \sigma_\ell \left(\boldsymbol{u}^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_0 \boldsymbol{u}^\ell \right) \left(\prod_{j \neq \gamma} \boldsymbol{f}_j^{\mathtt{old}\,\mathsf{T}} \boldsymbol{A}_j \boldsymbol{f}_j^\ell \right) \boldsymbol{f}_\gamma^\ell$;
      $\boldsymbol{a} = \sum_{\hat{m}=1}^{M_\phi} \alpha_{\hat{m}} \boldsymbol{\phi}_\gamma^{\hat{m}}$;
      Compute $\boldsymbol{f}_\gamma^{\mathtt{new}} = \boldsymbol{g} \oslash \boldsymbol{a}$
    end for
    Compute amplitude, amplitude error and update $\mathtt{old} \leftarrow \mathtt{new}$;
  end while
  Store: $\sigma_m \leftarrow \sigma^{\mathtt{new}}$; $\boldsymbol{u}^m \leftarrow \boldsymbol{u}^{\mathtt{new}}$; $\boldsymbol{f}_j^m \leftarrow \boldsymbol{f}_j^{\mathtt{new}}$, for $i = 1, 2, \ldots, \mathtt{n_d}$;
  Update $m \leftarrow m + 1$;
end while
Store: $M \leftarrow m$;

4 Numerical Illustrations

All the examples described in this section are produced with the `matlab` open-source routines hosted in GitHub, https://git.lacan.upc.edu/zlotnik/algebraicPGDtools. The package includes a tutorial with examples to ease the use of the routines.

4.1 PGD Separation of the Butterfly Curve Family

The function with six arguments given by

$$\boldsymbol{\Phi}(\theta, a, b, c, d, e) = a \, \exp(\cos\theta) - b \, \cos(c\,\theta) + \sin^d(\theta/e), \tag{54}$$

is defining a five-dimensional set of curves expressed as a polar representation of the function, $r(\theta) = \boldsymbol{\Phi}(\theta, a, b, c, d, e)$, each curve corresponding to a given value of the five parameters (a, b, c, d, e). Due to their shape, this family is denoted as the *butterfly curve*, see [9]. The ranges of the arguments of $\boldsymbol{\Phi}$ are taken to be $\theta \in [0, 2\pi]$, $a \in [-1, 1]$, $b \in [-3, 3]$, $c \in [0, 4]$, $d \in [0, 5]$ and $e \in [1, 12]$. Multidimensional function $\boldsymbol{\Phi}$ is discretized in to a six-dimensional tensor $\boldsymbol{\Phi} \in \mathbb{R}^{n_\theta \times n_a \times n_b \times n_c \times n_d \times n_e}$, obtained by uniform sampling along each of the six sectional dimensions, being $n_\theta = 100$, $n_a = 20$, $n_b = 20$, $n_c = 20$, $n_d = 6$, and $n_e = 20$ the number of sampling point in each dimension. This tensorial object is selected as a benchmark for high-order tensor separation because it allows a graphic representation (the butterfly

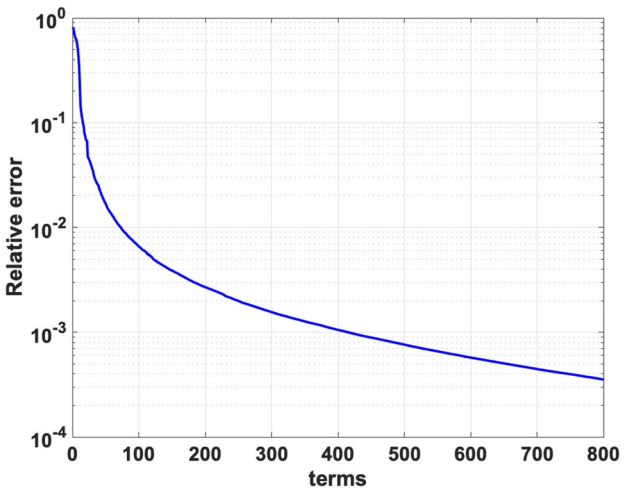

Fig. 1 PGD separation of the butterfly curve. Convergence of the relative error in Frobenius norm, with the number of terms of the separated approximation

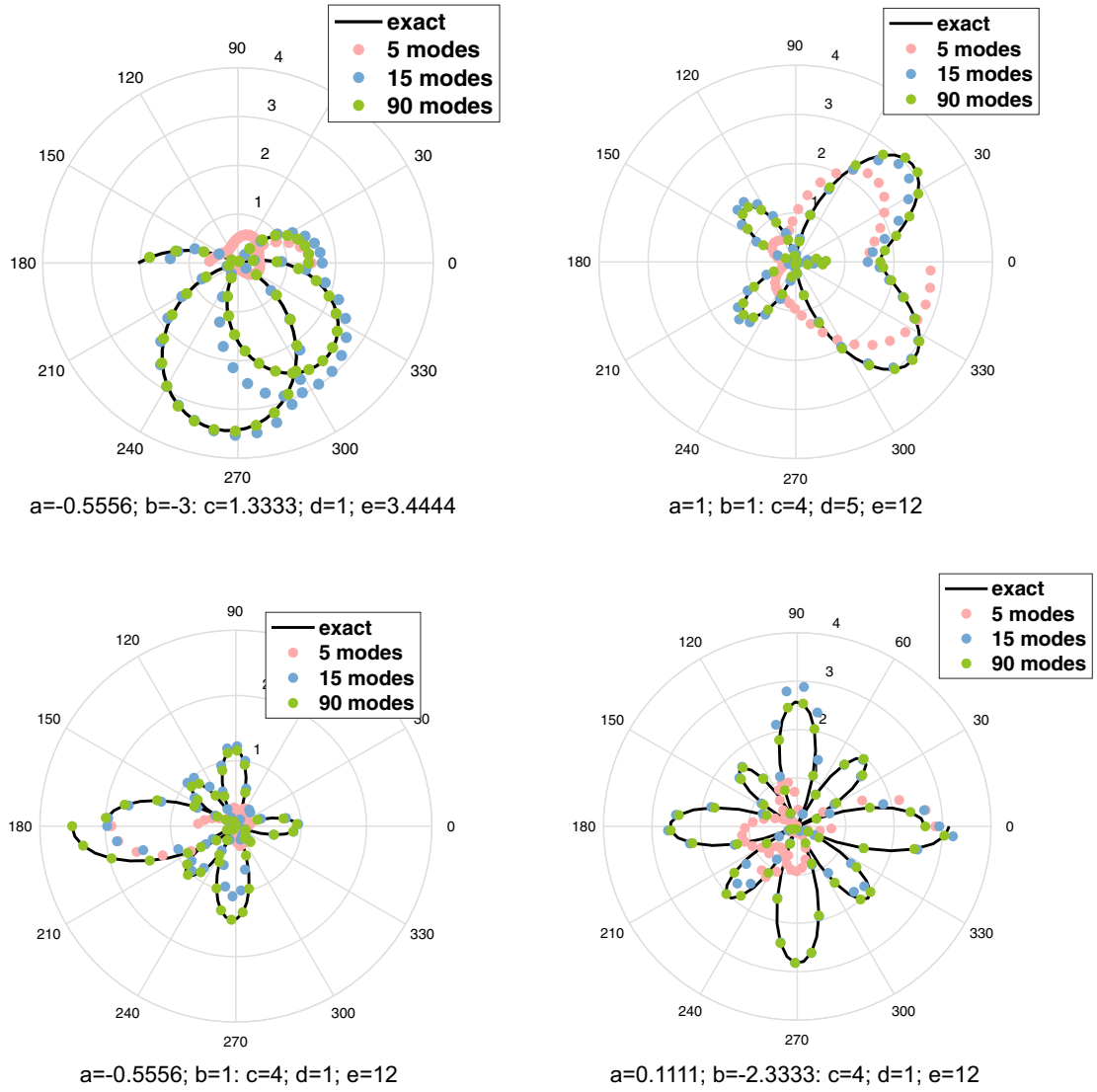

Fig. 2 PGD separation of the butterfly curve. Illustration of the approximation for four different sets of parameters. In each case three PGD solutions (corresponding to 5, 15 and 90 modes) are compared with the exact butterfly curve

curves) of both the original tensor and its separated approximations, see [14]. Function $\boldsymbol{\Phi}$ defined in (54) inducing the butterfly curve does not accept an exact separated representation as described in (1).

Note that the six-dimensional tensor $\boldsymbol{\Phi}$ has a number of $n_\theta \times n_a \times n_b \times n_c \times n_d \times n_e$ entries (in this case it amounts to 96 millions) and the storage of the full tensor requires $\sim$ 732 MB of memory. The PGD separation as described in Algorithm 1 and available from the GitHub account is used to obtain a separable expression of tensor $\boldsymbol{\Phi}$ up to 800 terms, namely

$$\boldsymbol{\Phi} \approx \boldsymbol{F}^M = \sum_{m=1}^{M} \boldsymbol{f}_\theta^m \otimes \boldsymbol{f}_a^m \otimes \boldsymbol{f}_b^m \otimes \boldsymbol{f}_c^m \otimes \boldsymbol{f}_d^m \otimes \boldsymbol{f}_e^m, \tag{55}$$

being M the number of PGD terms that in this case goes up to 800. This allows easily computing the evolution of the error norm with the number of terms in the separation (55), that is along the greedy algorithm, namely $\|\boldsymbol{\Phi} - \boldsymbol{F}^M\|$ for $M = 1, 2, \ldots$. Here, $\|\cdot\|$ stands for the Frobenius norm. Figure 1 depicts the evolution of the relative error with the number of terms, M. Note that the error associated with the more accurate approximation ($M = 800$) is 3.5×10^{-4} and that instead of the ~732 Mb of storage memory of the full tensor, the separated version with 800 terms has $800 \times (n_\theta + n_a + n_b + n_c + n_d + n_e) = 148{,}800$ entries that amount to ~1.1 MB. That is, accepting an error of less than 0.1% allows a compression factor reducing 700 times the storage memory.

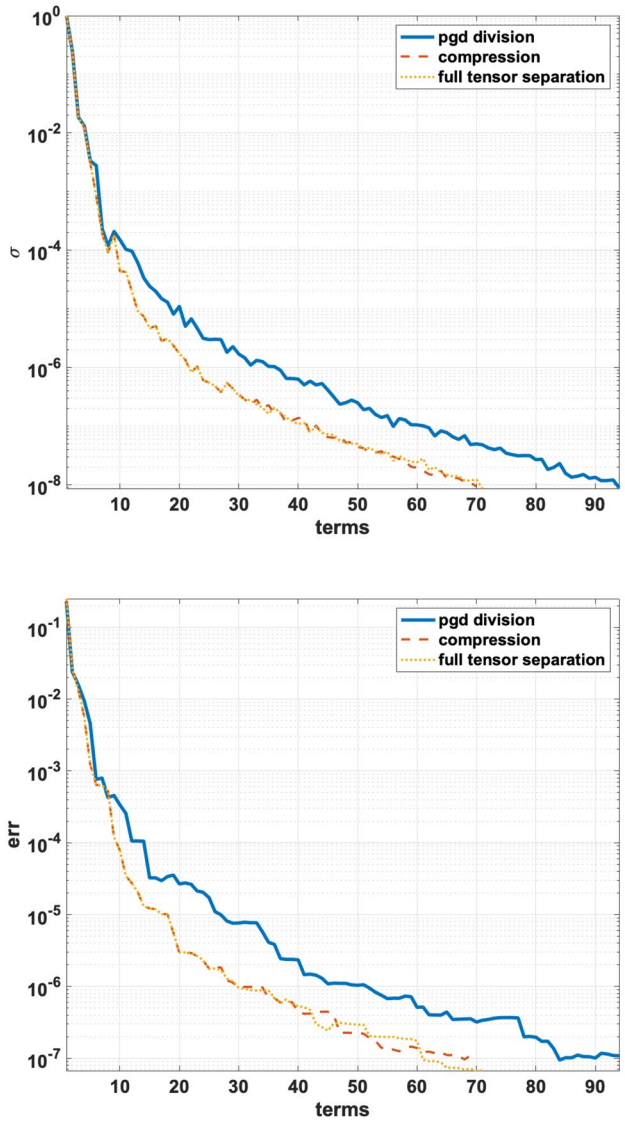

Fig. 3 Top panel: evolution of amplitude σ with the number of terms of: (i) the pgd-Hadamard-division $\boldsymbol{P}$ (solid line), (ii) its compressed version $\boldsymbol{P}^{\text{comp}}$ (dashed line), and (iii) direct separation of $\boldsymbol{R}^{\text{full}}$, the reference solution (dividing full tensors) (doted line). Bottom panel: convergence of the relative error in infinite norm with the number of terms in the solution for (i), (ii) and (iii) with respect to $\boldsymbol{R}^{\text{full}}$

In Fig. 2 four different instances of the Butterfly curve are displayed for six values of the parameters in the discretized range. In each of the four cases, the full order tensor is associated with the exact curve, in black, and three approximations corresponding to using 5, 15 and 90 PGD terms (in pink, blue and green, respectively). Note that, according to the graph in Fig. 1, 90 terms provide a solution with an error lower than 1%. Therefore, solutions with numbers of modes larger than 90 are indistinguishable from the exact solution to the naked eye.

4.2 PGD Division, Compression and Separation

The PGD Hadamard division (Algorithm 2) is presented next by using a synthetic example. Two three-dimensional tensors $\boldsymbol{D}$ and $\boldsymbol{E}$ are built based on the following functions,

$$E(x,y,z) = \sum_{m=1}^{3} f_x^m(x) f_y^m(y) f_z^m(z), \quad \text{and}$$

$$D(x,y,z) = \sum_{m=1}^{3} g_x^m(x)\, g_y^m(y)\, g_z^m(z)$$

being $f_x^m(x) = \sin(\pi x m)$, $f_y^m(y) = y^m$ and $f_z^m(z) = mz$ and $g_x^m(x) = \cos(\pi x m) + 1$, $g_y^m(y) = y^{1/m} + 1$, $g_z^m(z) = z + m$.

The corresponding separated tensors $\boldsymbol{E}$ and $\boldsymbol{D}$ are obtained by evaluating previous functions f_i^m and g_i^m, for $i = x, y, z$ and $m = 1, 2, 3$. Each spatial direction lies in the $[0, 1]$ range, that is $(x, y, z) \in [0, 1]^3$, and are discretized with 70, 130 and 190 uniformly spaced points (respectively to x, y, and z). The Hadamard division algorithm devised in Sect. 3.1 produces a separated tensor $\boldsymbol{P} \approx \boldsymbol{E} \oslash \boldsymbol{D}$, without reconstructing the full tensor versions of $\boldsymbol{E}$ and $\boldsymbol{D}$. Note that each full tensor requires storing $70 \times 130 \times 190 = 1{,}729{,}000$ entries, wether their separated versions are expressed with only $3 \times (70 + 130 + 190) = 1\,170$ entries. In this toy example, the direct Hadamard division of the full tensors is performed to obtain a reference value $\boldsymbol{R}^{\text{full}} = \boldsymbol{E} \oslash \boldsymbol{D}$. Thus, the error is measured as $\|\boldsymbol{P} - \boldsymbol{R}^{\text{full}}\|_\infty / \|\boldsymbol{R}^{\text{full}}\|_\infty$.

The same example is used to test the PGD compression. Once $\boldsymbol{P}$ is computed, it can be readily compressed into $\boldsymbol{P}^{\text{comp}}$ using Algorithm 1. Also the reference solution $\boldsymbol{R}^{\text{full}}$ is expressed in separable format with the separation strategy devised in Sect. 2, this is denoted by $\boldsymbol{R}^{\text{comp}}$ and one would expect that it performs similarly to $\boldsymbol{P}^{\text{comp}}$.

Figure 3 shows the evolution of the modal amplitude (left panel) and the convergence of the error with the number of modes (right panel) for $\boldsymbol{P}$, $\boldsymbol{P}^{\text{comp}}$ and $\boldsymbol{R}^{\text{comp}}$. In the error convergence curves, the error is computed in all cases with respect to $\boldsymbol{R}^{\text{full}}$. In this example 84 terms for $\boldsymbol{P}$ suffice to obtain a separable representation with relative error in infinite norm of 10^{-7}. As expected, the solutions $\boldsymbol{P}^{\text{comp}}$ and $\boldsymbol{R}^{\text{comp}}$ that result from compressing $\boldsymbol{P}$ and $\boldsymbol{R}^{\text{full}}$ behave similarly, both in the evolution of the amplitude of their terms and in the evolution of the relative error.

4.3 PGD Linear System Solver

The PGD linear system solver is used next to solve a higher-dimensional linear system arising from the parameterization of an advection-diffusion problem,

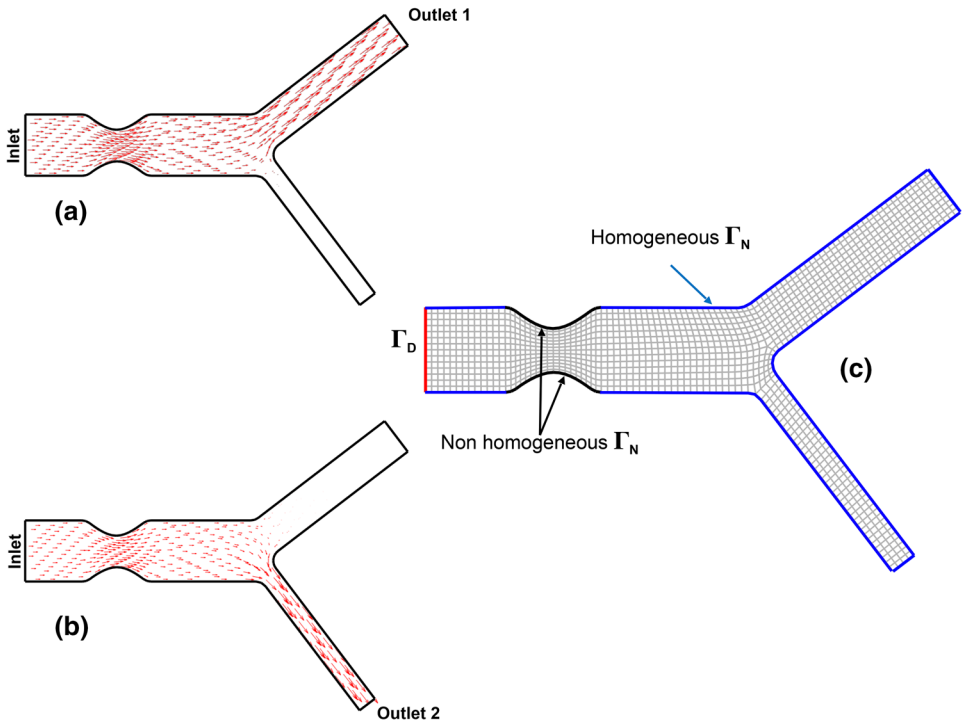

Fig. 4 Potential flow velocity map distributions: **a** $\boldsymbol{v}_{p1}$ for the upper branch, and **b** $\boldsymbol{v}_{p2}$ for the lower branch. **c** Domain for the convection diffusion problem

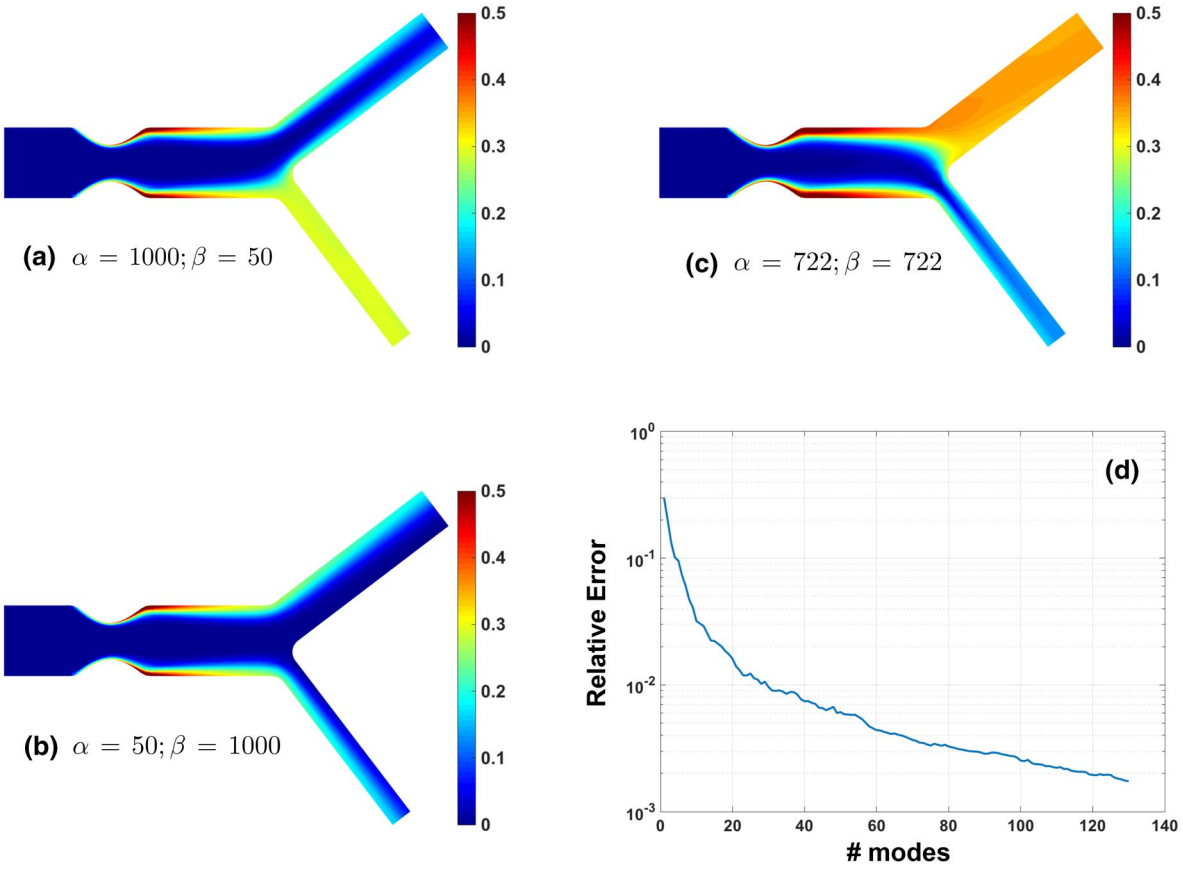

Fig. 5 Concentration distribution $\boldsymbol{u}_{PGD}$ of the stationary convection diffusion system, using the PGD linear system solver, with predominant flow in **a** upper branch, **b** lower branch and **c** both branches opened. **d** Relative error PGD versus finite element convergence

$$\begin{cases} -\nabla \cdot (\nu \nabla u) + \boldsymbol{v} \cdot \nabla u = 0 & \text{in } \Omega \\ u = u_D & \text{on } \Gamma_D \\ \boldsymbol{n} \cdot \nu \nabla u = u_N(\boldsymbol{x}) & \text{on } \partial\Omega \setminus \Gamma_D = \Gamma_N \end{cases} \tag{56}$$

where ν is the diffusivity, and $\boldsymbol{v}(\boldsymbol{x})$ is a vector field describing the advection velocity. The velocity $\boldsymbol{v}(\boldsymbol{x})$ is taken to be parametric, as a linear combination of two fields $\boldsymbol{v}_1(\boldsymbol{x})$ and $\boldsymbol{v}_2(\boldsymbol{x})$, shown in Fig. 4a, b. Thus, parameters α and β are introduced such that

$$\boldsymbol{v}(\boldsymbol{x}, \alpha, \beta) = \alpha\, \boldsymbol{v}_1(\boldsymbol{x}) + \beta\, \boldsymbol{v}_2(\boldsymbol{x}), \tag{57}$$

being the ranges of variation of the parameters such that $\alpha \in [50, 1000]$ and $\beta \in [50, 1000]$.

The discrete version of the problem before enforcing the essential boundary conditions (those on Γ_D) reads,

$$\left(\boldsymbol{K}_\nu + \alpha \boldsymbol{C}_{v_1} + \beta \boldsymbol{C}_{v_2}\right) \boldsymbol{u} = \boldsymbol{f}, \tag{58}$$

where $\boldsymbol{K}_\nu$ is the stiffness matrix corresponding to the second order operator, $\boldsymbol{C}_{v_1}$ and $\boldsymbol{C}_{v_2}$ the convection matrices corresponding to the velocity fields $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ respectively, and $\boldsymbol{f}$ the vector accounting for the Neumann boundary conditions (which in this case are independent of the parameters). As indicated in Fig. 4c, the Neumann boundary conditions are homogeneous everywhere on Γ_N, except in the zone of the narrowing, where $u_N \neq 0$ enforces a flux that accounts for an injection of the mass (or heat) that increases locally the concentration (or temperature). Note that the discrete form (58) is straightforwardly expressed in an affine decomposition, namely

$$\boldsymbol{K}(\alpha, \beta) = \sum_{m=1}^{3} \boldsymbol{K}^m\, g^m(\alpha)\, h^m(\beta) \tag{59}$$

with $\boldsymbol{K}^1 = \boldsymbol{K}_\nu$, $g^1(\alpha) = h^1(\beta) = 1$; $\boldsymbol{K}^2 = \boldsymbol{C}_{v_1}$, $g^2(\alpha) = \alpha$, $h^2(\beta) = 1$; $\boldsymbol{K}^3 = \boldsymbol{C}_{v_2}$, $g^3(\alpha) = 1$, $h^3(\beta) = \beta$.

The essential (or Dirichlet) boundary conditions are implemented by suppressing the equations corresponding to the prescribed degrees of freedom and bringing to the right-hand-side the corresponding columns of $\boldsymbol{K}(\alpha, \beta)$ multiplied by the prescribed value of the concentration u.

In practice (58) results in the following reduced version

$$\left(\boldsymbol{K}^\star_\nu + \alpha\, \boldsymbol{C}^\star_{v_1} + \beta\, \boldsymbol{C}^\star_{v_2}\right) \boldsymbol{u}^\star = \boldsymbol{f}^\star - \boldsymbol{f}^\star_\nu - \alpha \boldsymbol{f}^\star_{v_1} - \beta \boldsymbol{f}^\star_{v_2}, \tag{60}$$

where the matrices and vectors with the subscript $\star$ do not include the prescribed degrees of freedom, vectors $\boldsymbol{f}^\star_\nu$, $\boldsymbol{f}^\star_{v_1}$ and $\boldsymbol{f}^\star_{v_2}$ are the linear combination of the columns of matrices $\boldsymbol{K}_\nu$, $\boldsymbol{C}_{v_1}$ and $\boldsymbol{C}_{v_2}$ weighted with the prescribed values of the concentration u.

The parametric dimensions are discretized with 100 equally spaced points each. Thus, functions g^m and h^m, $m = 1, 2, 3$ are described by vectors of 100 components and system (60) is readily written in the format of (41). Then, the input for Algorithm 3 is ready and the solution provided is seen as a Vademecum (explicit parametric solution), containing is a separated format the concentration fields for all possible values of the parameters α and β.

Figure 5a–c show, in the form of a concentration map (distribution of $u(\boldsymbol{x})$) the PGD solution at some specific values of α and β. Figure 5a illustrates a solution in which the advection is dominated by $\boldsymbol{v}_1$, Fig. 5b with advection dominated by $\boldsymbol{v}_2$ and Fig. 5c with an intermediate situation.

The evolution of the error (with respect to the full Finite Element solution for all possible values of the parameters in the discretized 100×100 grid) measured in an Euclidean norm (or $\mathcal{L}_2$ norm) for the four-dimensional space (two space dimensions plus two parametric dimensions) is displayed in Fig. 5d. It is worth noting that with less than 100 PGD terms the error is reduced to 0.5%.

5 Closure

The PGD Least-Squares strategy to produce a separated approximation of a multidimensional tensor is devised as the discrete form of approximating a multivariate function. This philosophy is generalized to operate with separated objects, performing complex operations (division, matrix inversion...) in the separated format, that is without generating the full multidimensional tensor and therefore precluding the *curse of dimensionality*.

A set of `matlab` routines implementing the *Encapsulated PGD toolbox*, that is the strategies described in the paper, is openly released at https://git.lacan.upc.edu/zlotnik/algebraicPGDtools.

The toolbox character of these algorithms is strengthen by the definition of a class of objects corresponding to separated tensors, and the corresponding elementary operations.

The application of this strategy to several illustrative test cases shows the efficiency and implementation convenience of using the toolbox.

Acknowledgements This work is partially funded by Generalitat de Catalunya (Grant No. 1278 SGR 2017-2019) and Ministerio de Economía y Empresa and Ministerio de Ciencia, Innovación y Universidades (Grant No. DPI2017-85139-C2-2-R).

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Beylkin G, Mohlenkamp MJ (2002) Numerical operator calculus in higher dimensions. Proc Nal Acad Sci 99(16):10246–10251
2. Beylkin G, Mohlenkamp MJ (2005) Algorithms for numerical analysis in high dimensions. SIAM J Sci Comput 26(6):2133–2159
3. Borzacchiello D, Chinesta F, Malik M, García-Blanco R, Díez P (2016) Unified formulation of a family of iterative solvers for power systems analysis. Electr Power Syst Res 140:201–208. https://doi.org/10.1016/j.epsr.2016.06.021
4. Chinesta F, Keunings R, Leygue A (2014) The proper generalized decomposition for advanced numerical simulations. A primer. Springer briefs in applied sciences and technology. Springer, Cham. https://doi.org/10.1007/978-3-319-02865-1
5. Chinesta F, Leygue A, Bordeu F, Aguado JV, Cueto E, González D, Alfaro I, Ammar A, Huerta A (2013) PGD-based computational vademecum for efficient design, optimization and control. Arch Comput Methods Eng 20:31–59. https://doi.org/10.1007/s11831-013-9080-x
6. Díez P, Zlotnik S, García-González A, Huerta A (2018) Algebraic PGD for tensor separation and compression: an algorithmic approach. C R Mécanique 346(7):501–5014. https://doi.org/10.1016/j.crme.2018.04.011
7. Doostan A, Iaccarino G (2009) A least-squares approximation of partial differential equations with high-dimensional random inputs. J Comput Phys 228(12):4332–4345
8. Espig M, Hackbusch W, Litvinenko A, Matthies HG, Zander E (2012) Efficient analysis of high dimensional data in tensor formats. In: Garcke J, Griebel M (eds) Sparse grids and applications, vol 88. Lecture notes in computational science and engineering. Springer, Berlin, Heidelberg, pp 31–56. https://doi.org/10.1007/978-3-642-31703-3_2
9. Fay TH (1989) The butterfly curve. Am Math Mon 96:442–443
10. García-Blanco R, Borzacchiello D, Chinesta F, Díez P (2017) Monitoring a PGD solver for parametric power flow problems with goal-oriented error assessment. Int J Numer Methods Eng 111:529–552. https://doi.org/10.1002/nme.5470
11. García-Blanco R, Díez P, Borzacchiello D, Chinesta F (2017) Algebraic and parametric solvers for the power flow problem: towards real-time and accuracy-guaranteed simulation of electric systems. Arch Comput Methods Eng. https://doi.org/10.1007/s11831-017-9223-6
12. Grasedyck L, Kressner D, Tobler C (2013) A literature survey of low-rank tensor approximation techniques. GAMM-Mitteilungen 36(1):53–78
13. Kolda T, Bader B (2009) Tensor decompositions and applications. SIAM Rev 51:455–500
14. Modesto D, Zlotnik S, Huerta A (2015) Proper generalized decomposition for parameterized Helmholtz problems in heterogeneous and unbounded domains: application to harbor agitation. Comput Methods Appl Mech Eng 295:127–149. https://doi.org/10.1016/j.cma.2015.03.026
15. Nouy A (2017) Low-rank tensor methods for model order reduction. In: Ghanem R, Higdon D, Owhadi H (eds) Handbook of uncertainty quantification. Springer, Cham. https://doi.org/10.1007/978-3-319-12385-1_21
16. Oseledets IV (2011) Tensor-train decomposition. SIAM J Sci Comput 33(5):2295–2317
17. Rozza G (2014) Fundamentals of reduced basis method for problems governed by parametrized PDEs and applications. In: Chinesta F, Ladevèze P (eds) Separated representations and pgd-based model reduction. CISM international centre for mechanical sciences, vol 554. Springer, Vienna, pp 153–227. https://doi.org/10.1007/978-3-7091-1794-1_4
18. Sibileau A, García-González A, Auricchio F, Morganti S, Díez P (2018) Explicit parametric solutions of lattice structures with proper generalized decomposition (PGD): applications to the design of 3D-printed architectured materials. Comput Mech 62(4):871–891. https://doi.org/10.1007/s00466-017-1534-9