

The Sparse Grids Matlab kit - a Matlab implementation of sparse grids for high-dimensional function approximation and uncertainty quantification

Chiara Piazzola^{1,2} and Lorenzo Tamellini¹

¹Istituto di Matematica Applicata e Tecnologie Informatiche “E. Magenes”, Consiglio Nazionale delle Ricerche, Via Ferrata, 5/A 27100, Pavia, Italy

²Department of mathematics, Technical University of Munich, Boltzmannstraße, 3 85748, Garching bei München, Germany
 chiara.piazzola@tum.de, tamellini@imati.cnr.it

Abstract

The Sparse Grids Matlab Kit is a collection of Matlab functions for high-dimensional interpolation and quadrature, based on the combination technique form of sparse grids. It is lightweight, high-level and easy to use, good for quick prototyping and teaching. However, it has some features that allow its use also in realistic applications: in particular, the Sparse Grids Matlab Kit is somehow geared towards Uncertainty Quantification, but it is flexible enough for other purposes. The goal of this paper is to provide an overview of the data structure and of mathematical aspects forming the backbone of the software, as well as to compare it with similar software available in literature.

1 Introduction

The aim of this manuscript is to introduce the Sparse Grids Matlab Kit as a tool for approximation of high-dimensional functions and Uncertainty Quantification (UQ). The Sparse Grids Matlab Kit is freely available under the BSD2 license on Github at <https://github.com/lorenzo-tamellini/sparse-grids-matlab-kit>, and full resources (past and current releases, documentation including the user manual [47], other release-related material) are available at <https://sites.google.com/view/sparse-grids-kit>; the first version was released in 2014 (14-4 “Ritchie”), and the current version was released in 2023 (23-5 “Robert”). It is written in Matlab, and its compatibility with Octave has been tested; it is extensively documented and commented (release 23-5 has about 9800 lines of code and 5300 lines of comment). The release contain several tutorials and a testing unit is also available.

From a mathematical point of view, the package implements the combination technique form of sparse grids: it is a high-level package, with syntax quite close to the mathematical description of sparse grids, which makes it (hopefully) easy to use, and therefore suitable for quick prototyping and didactic purposes (for instance, it has been used to write the codes of the book [36]). During the discussion, we will however point out a few functionalities (interface with the Matlab Parallel Toolbox, *evaluation recycling*, *buffering* strategy for adaptive sparse grids generation, full compatibility with the UM-Bridge protocol [53] for HTTP communication with external software) that make the Sparse Grids Matlab Kit usable for realistic UQ problems, see e.g. [11, 48, 8, 54], as well as for academic problems with hundreds of random variables, such as in [42, 18].

Name	Language	Ref.	Webpage
Dakota	C++	[1]	https://dakota.sandia.gov
PyApprox	Python	[30]	https://pypi.org/project/pyapprox
MUQ	C++, Python	[44]	https://mituq.bitbucket.io
UQLab	Matlab	[35]	https://uqlab.com
ChaosPy	Python	[20, 19]	https://chaospy.readthedocs.io
SG++	Python, Matlab, Java, C++	[46]	https://sgpp.sparsegrids.org/
Spinterp	Matlab	[31, 33, 32]	http://calgo.acm.org/847.zip ¹
UQTK	C++, Python	[14, 13]	https://sandia.gov/uqtoolkit
Tasmanian	C++, Python, Matlab, Fortran 90/95	[57, 59, 58]	https://github.com/ORNL/TASMANIAN
OpenTURNS	C++, Python	[4]	https://openturns.github.io/www/index.html
URANIE	C++, Python	[5]	https://www.salome-platform.org/?page_id=2019
UncertainSCI	Python	[39]	https://www.sci.utah.edu/cibc-software/uncertainsci.html

Table 1: List of high-dimensional approximation / UQ-related software.

As already mentioned, the Sparse Grids Matlab Kit is geared towards UQ, although it is general enough to be used for manipulation of high-dimensional functions in other frameworks. As such, it belongs to the same niche of a number of other software with surrogate modeling functionalities for parameter space exploration/UQ purposes; we provide a (knowingly incomplete) list in Tab. 1. The software in the table closest to the Sparse Grids Matlab Kit (in terms of language, functionalities and usability) is probably Spinterp, which is however no longer maintained and does not implement any UQ function. A deeper discussion is reported in Sect. 5, where a closer comparison in terms of functionalities between the Sparse Grids Matlab Kit and the other Matlab-based sparse-grids/UQ software (either natively implemented in Matlab or providing interfaces to software written in C++/Python), i.e., SG++, Spinterp, Tasmanian, is given.

The rest of the paper is organized as follows. Sect. 2 introduces the minimal mathematical background necessary to understand the entities implemented in the Sparse Grids Matlab Kit. Sect. 3 covers how sparse grids are generated in the Sparse Grids Matlab Kit and how they are stored in memory (data structure); note in particular that the Sparse Grids Matlab Kit provides two mechanisms to generate sparse grids: *a-priori* and *adaptive a-posteriori* (this is where we will discuss in particular the above-mentioned *buffering* of random variables). Sect. 4 discusses the main functionalities available in the Sparse Grids Matlab Kit, with special emphasis on knots recycling, parallelization, interface with the UM-Bridge protocol and conversion to polynomial chaos expansions, while Sect. 5 contains the already-mentioned comparison with the other Matlab sparse-grids/UQ software available in the literature.

2 Mathematical basics of sparse grids

We consider the two problems of a) approximating and b) computing weighted integrals of (the components of) a function $f : \mathbb{R}^N \rightarrow \mathbb{R}^V$ given some samples of f whose location we are free to choose. More specifically, we assume that f depends on N random variables $\mathbf{y} = (y_1, \dots, y_N) \in \Gamma$, with $\Gamma = \Gamma_1 \times \dots \times \Gamma_N \subset \mathbb{R}^N$ being the set of all possible values of \mathbf{y} . We denote by $\rho_n : \Gamma_n \rightarrow \mathbb{R}^+$ the probability density function (pdf) of each variable y_n , $n = 1, \dots, N$ and assume independence of y_1, \dots, y_N , such that the joint pdf of \mathbf{y} is

¹Last officially released version, to the best of knowledge of the authors of this manuscript. A later version can be found at https://people.sc.fsu.edu/~jburkardt/m_src/spinterp/spinterp.html

$$\rho(\mathbf{y}) = \prod_{n=1}^N \rho_n(y_n), \forall \mathbf{y} \in \Gamma.^2$$

The first step to build a sparse grid is to define a set of collocation knots for each variable y_n . We denote the number of knots to be used in each random variable by $K_n \in \mathbb{N}_+$, and introduce a discretization level $i_n \in \mathbb{N}_+$ and a so-called “level-to-knots function”

$$m : \mathbb{N}_+ \rightarrow \mathbb{N}_+ \text{ such that } m(i_n) = K_n. \quad (1)$$

Then, we denote by \mathcal{T}_{n,i_n} the set of $m(i_n)$ knots along y_n , i.e.,

$$\mathcal{T}_{n,i_n} = \left\{ y_{n,m(i_n)}^{(j_n)} : j_n = 1, \dots, m(i_n) \right\} \quad \text{for } n = 1, \dots, N. \quad (2)$$

Typical examples of level-to-knots functions are:

$$m(i) = i \quad (\text{linear}), \quad (3)$$

$$m(i) = 2(i-1) + 1 \quad (\text{2-step}), \quad (4)$$

$$m(1) = 1, m(i) = 2^{i-1} + 1 \quad (\text{doubling}). \quad (5)$$

The knots of \mathcal{T}_{n,i_n} are usually chosen according to the probability distribution of the random variables ρ_n for efficiency reasons. For the same reasons, it is also beneficial if the sequences of knots are *nested*, i.e., if $\mathcal{T}_{n,i_n} \subset \mathcal{T}_{n,j_n}$ with $j_n \geq i_n$. However, the Sparse Grids Matlab Kit does not require nestedness of these sequences, contrary to other software (as we will discuss later in Sect. 5).

Next, we introduce N -dimensional tensor grids obtained by taking the Cartesian product of the N univariate sets of knots just introduced. For this purpose we collect the discretization levels i_n in a multi-index $\mathbf{i} = [i_1, \dots, i_N] \in \mathbb{N}_+^N$ and denote the corresponding tensor grid by $\mathcal{T}_{\mathbf{i}} = \bigotimes_{n=1}^N \mathcal{T}_{n,i_n}$. Using standard multi-index notation, we can then write

$$\mathcal{T}_{\mathbf{i}} = \left\{ \mathbf{y}_{m(\mathbf{i})}^{(\mathbf{j})} \right\}_{\mathbf{j} \leq m(\mathbf{i})}, \quad \text{with} \quad \mathbf{y}_{m(\mathbf{i})}^{(\mathbf{j})} = \left[y_{1,m(i_1)}^{(j_1)}, \dots, y_{N,m(i_N)}^{(j_N)} \right] \text{ and } \mathbf{j} \in \mathbb{N}_+^N,$$

where $m(\mathbf{i}) = [m(i_1), m(i_2), \dots, m(i_N)]$ and $\mathbf{j} \leq m(\mathbf{i})$ means that $j_n \leq m(i_n)$ for every $n = 1, \dots, N$.

A *tensor grid approximation* of $f(\mathbf{y})$ based on global Lagrangian interpolants collocated at these grid knots can then be written in the following form

$$\mathcal{U}_{\mathbf{i}}(\mathbf{y}) := \sum_{\mathbf{j} \leq m(\mathbf{i})} f\left(\mathbf{y}_{m(\mathbf{i})}^{(\mathbf{j})}\right) \mathcal{L}_{m(\mathbf{i})}^{(\mathbf{j})}(\mathbf{y}), \quad (6)$$

where $\left\{ \mathcal{L}_{m(\mathbf{i})}^{(\mathbf{j})}(\mathbf{y}) \right\}_{\mathbf{j} \leq m(\mathbf{i})}$ are N -variate Lagrange basis polynomials, defined as tensor products of univariate Lagrange polynomials, i.e.

$$\mathcal{L}_{m(\mathbf{i})}^{(\mathbf{j})}(\mathbf{y}) = \prod_{n=1}^N l_{n,m(i_n)}^{(j_n)}(y_n) \quad \text{with} \quad l_{n,m(i_n)}^{(j_n)}(y_n) = \prod_{k=1, k \neq j_n}^{m(i_n)} \frac{y_n - y_{n,m(i_n)}^{(k)}}{y_{n,m(i_n)}^{(j_n)} - y_{n,m(i_n)}^{(k)}}.$$

Similarly, the *tensor grid quadrature* of $f(\mathbf{y})$, i.e. the approximation of its weighted integral, can be computed by taking the integral of the Lagrangian interpolant in Eq. (6):

$$\begin{aligned} Q_{\mathbf{i}} &:= \int_{\Gamma} \mathcal{U}_{\mathbf{i}}(\mathbf{y}) \rho(\mathbf{y}) d\mathbf{y} = \sum_{\mathbf{j} \leq m(\mathbf{i})} f\left(\mathbf{y}_{m(\mathbf{i})}^{(\mathbf{j})}\right) \left(\prod_{n=1}^N \int_{\Gamma_n} l_{n,m(i_n)}^{(j_n)}(y_n) \rho(y_n) dy_n \right) \\ &= \sum_{\mathbf{j} \leq m(\mathbf{i})} f\left(\mathbf{y}_{m(\mathbf{i})}^{(\mathbf{j})}\right) \left(\prod_{i=1}^N \omega_{n,m(i_n)}^{(j_n)} \right) = \sum_{\mathbf{j} \leq m(\mathbf{i})} f\left(\mathbf{y}_{m(\mathbf{i})}^{(\mathbf{j})}\right) \omega_{m(\mathbf{i})}^{(\mathbf{j})}, \end{aligned} \quad (7)$$

²The assumption of independence is kept for simplicity but is actually not needed: a) approximation can be performed using only the marginal probability density functions of y_1, \dots, y_N , and b) while independence is needed for quadrature, in case y_1, \dots, y_N are correlated it is possible to use the theory of copulas [40] to introduce a change of variables where the new random variables are independent uniform random variables, and to set the sparse grid in this new set of variables.

where $\omega_{n,m(i_n)}^{(j_n)}$ are the standard quadrature weights obtained by computing the integrals of the associated univariate Lagrange polynomials, and $\omega_{m(\mathbf{i})}^{(\mathbf{j})}$ are their multivariate counterparts. Note that other choices could be done here, replacing Lagrange polynomials in Eq. (6) by e.g. splines as in [51], or trigonometric polynomials as in [37]; we discuss further this issue in Sect. 5, when comparing different sparse grid software. Naturally, the approximations $\mathcal{U}_{\mathbf{i}}$ and $\mathcal{Q}_{\mathbf{i}}$ are more and more accurate the higher the number of collocation knots in each random variable, and therefore one would ideally choose all the components of \mathbf{i} to be large, say $\mathbf{i} = \mathbf{i}^*$ with $i_n^* \gg 1, \forall n = 1, \dots, N$. The cost of these approximations could be however too large even for N moderately large, due to fact that they would require $\prod_{n=1}^N m(i_n^*)$ evaluations of f , i.e., their cost grows exponentially fast in N .

To circumvent this issue, the sparse grid method replaces $\mathcal{U}_{\mathbf{i}^*}$ with a linear combination of multiple coarser $\mathcal{U}_{\mathbf{i}}$, and similarly for $\mathcal{Q}_{\mathbf{i}^*}$ (from now on we use the generic symbol $\mathcal{F}_{\mathbf{i}}$ to denote both $\mathcal{U}_{\mathbf{i}}$ and $\mathcal{Q}_{\mathbf{i}}$). To this aim we introduce the so-called “detail operators” (univariate and multivariate). They are defined as follows, with the understanding that $\mathcal{F}_{\mathbf{i}}(\mathbf{y}) = 0$ when at least one component of \mathbf{i} is zero. Thus, we denote by \mathbf{e}_n the n -th canonical multi-index, i.e. $(\mathbf{e}_n)_k = 1$ if $n = k$ and 0 otherwise, and define

$$\begin{aligned} \textbf{Univariate detail: } \Delta_n[\mathcal{F}_{\mathbf{i}}] &= \mathcal{F}_{\mathbf{i}} - \mathcal{F}_{\mathbf{i}-\mathbf{e}_n} \text{ with } 1 \leq n \leq N, \\ \textbf{Multivariate detail: } \Delta[\mathcal{F}_{\mathbf{i}}] &= \bigotimes_{n=1}^N \Delta_n[\mathcal{F}_{\mathbf{i}}], \end{aligned} \tag{8}$$

where taking tensor products of univariate details amounts to composing their actions, i.e.

$$\Delta[\mathcal{F}_{\mathbf{i}}] = \bigotimes_{n=1}^N \Delta_n[\mathcal{F}_{\mathbf{i}}] = \Delta_1[\dots[\Delta_N[\mathcal{F}_{\mathbf{i}}]]].$$

By replacing the univariate details with their definitions, we can then see that this implies that the multivariate operators can be evaluated by evaluating certain full-tensor approximations $\mathcal{F}_{\mathbf{i}}$, and then taking linear combinations:

$$\Delta[\mathcal{F}_{\mathbf{i}}] = \Delta_1[\dots[\Delta_N[\mathcal{F}_{\mathbf{i}}]]] = \sum_{\mathbf{j} \in \{0,1\}^N} (-1)^{\|\mathbf{j}\|_1} \mathcal{F}_{\mathbf{i}-\mathbf{j}}.$$

Observe that by introducing these detail operators a hierarchical decomposition of $\mathcal{F}_{\mathbf{i}}$ can be obtained; indeed, the following telescopic identity holds true:

$$\mathcal{F}_{\mathbf{i}} = \sum_{\mathbf{j} \leq \mathbf{i}} \Delta[\mathcal{F}_{\mathbf{j}}]. \tag{9}$$

Example 1 (Telescopic identity) *As an example of the previous identity, consider the case $N = 2$. Recalling that by definition $\mathcal{F}_{[j_1, j_2]} = 0$ when either $j_1 = 0$ or $j_2 = 0$, it can be seen that*

$$\begin{aligned} \sum_{[j_1, j_2] \leq [2, 2]} \Delta[\mathcal{F}_{[j_1, j_2]}] &= \Delta[\mathcal{F}_{[1, 1]}] + \Delta[\mathcal{F}_{[1, 2]}] + \Delta[\mathcal{F}_{[2, 1]}] + \Delta[\mathcal{F}_{[2, 2]}] \\ &= \mathcal{F}_{[1, 1]} + (\mathcal{F}_{[1, 2]} - \mathcal{F}_{[1, 1]}) + (\mathcal{F}_{[2, 1]} - \mathcal{F}_{[1, 1]}) + (\mathcal{F}_{[2, 2]} - \mathcal{F}_{[2, 1]} - \mathcal{F}_{[1, 2]} + \mathcal{F}_{[1, 1]}) \\ &= \mathcal{F}_{[2, 2]}. \end{aligned}$$

The crucial observation that allows to get to sparse grids is that, under suitable regularity assumptions for $f(\mathbf{y})$, not all of the details in the hierarchical decomposition in Eq. (9) contribute equally to the approximation, i.e., some of them can be discarded and the resulting formula will retain good approximation properties at a fraction of the computational cost: roughly speaking, the multi-indices to be discarded are those corresponding to “high-order” details, i.e., those for which $\|\mathbf{j}\|_1$ is sufficiently large, see e.g. [7].

Example 2 (Dropping high-order details) *Following the example above and replacing the constraint $[j_1, j_2] \leq [2, 2]$ with $\|\mathbf{j}\|_1 \leq 3$ to drop the “highest-order” detail, we obtain the following approximation, that we call sparse grid approximation/quadrature of f :*

$$\mathcal{F}_{[2,2]} \approx \sum_{j_1+j_2 \leq 3} \Delta[\mathcal{F}_{[j_1, j_2]}] = \Delta[\mathcal{F}_{[1,1]}] + \Delta[\mathcal{F}_{[1,2]}] + \Delta[\mathcal{F}_{[2,1]}] = -\mathcal{F}_{[1,1]} + \mathcal{F}_{[1,2]} + \mathcal{F}_{[2,1]}.$$

In general, upon collecting the multi-indices to be retained in the sum in a multi-index set $\mathcal{I} \subset \mathbb{N}_+^N$ the sparse grids approximation of f and of its weighted integral can finally be written as (see e.g. [63]):

$$f(\mathbf{y}) \approx \mathcal{U}_{\mathcal{I}}(\mathbf{y}) = \sum_{\mathbf{i} \in \mathcal{I}} \Delta[\mathcal{U}_{\mathbf{i}}(\mathbf{y})] = \sum_{\mathbf{i} \in \mathcal{I}} c_{\mathbf{i}} \mathcal{U}_{\mathbf{i}}, \quad c_{\mathbf{i}} := \sum_{\substack{\mathbf{j} \in \{0,1\}^N \\ \mathbf{i} + \mathbf{j} \in \mathcal{I}}} (-1)^{\|\mathbf{j}\|_1} \quad (10)$$

$$\int_{\Gamma} f(\mathbf{y}) \rho(\mathbf{y}) \, d\mathbf{y} \approx \mathcal{Q}_{\mathcal{I}}(\mathbf{y}) = \sum_{\mathbf{i} \in \mathcal{I}} \Delta[\mathcal{Q}_{\mathbf{i}}(\mathbf{y})] = \sum_{\mathbf{i} \in \mathcal{I}} c_{\mathbf{i}} \mathcal{Q}_{\mathbf{i}}, \quad (11)$$

and the sparse grid is defined as

$$\mathcal{T}_{\mathcal{I}} = \bigcup_{\substack{\mathbf{i} \in \mathcal{I} \\ c_{\mathbf{i}} \neq 0}} \mathcal{T}_{\mathbf{i}}. \quad (12)$$

Remark 1 *The sparse grid approximation in Eq. (10) is not necessarily an interpolant operator, i.e., the sparse grid approximation $\mathcal{U}_{\mathcal{I}}(\mathbf{y})$ evaluated at the sparse grid knots might be different from the corresponding values of $f(\mathbf{y})$. More specifically, a sparse grid is interpolatory only if it is built with nested knots. In the following, with a slight abuse of terminology we will nonetheless refer to the operation of evaluating $\mathcal{U}_{\mathcal{I}}(\mathbf{y})$ by means of Eq. (10) as sparse grid interpolation. Another commonly used terminology, especially in the field of uncertainty quantification, is to refer to $\mathcal{U}_{\mathcal{I}}(\mathbf{y})$ as sparse grid surrogate model.*

The right-most equalities in Eqs. (10) and (11) are known as the *combination technique* form of the sparse grids approximation and quadrature (see [27]), which is the form implemented in the Sparse Grids Matlab Kit. Another possibility would be to implement the first form, i.e., the sum of detail operators $\sum_{\mathbf{i} \in \mathcal{I}} \Delta[\mathcal{U}_{\mathbf{i}}(\mathbf{y})]$, which is known as the *hierarchical* form of sparse grids; in Sect. 5 we specify which software implements which form. In particular, implementing the hierarchical form requires introducing a basis for the detail operators $\Delta[\mathcal{F}_{\mathbf{i}}]$ in Eq. (8) rather than for the tensor interpolants $\mathcal{U}_{\mathbf{i}}$: this request naturally suggest using as basis the hierarchical form of piecewise polynomials, such as the classical hat-functions, which in turn opens the way to the so-called *local adaptivity* of sparse grids, see e.g. [45, 15, 43, 34]. Using piecewise polynomials to introduce a basis for the detail operators $\Delta[\mathcal{F}_{\mathbf{i}}]$ is not mandatory though, and it would be possible to use global Lagrange polynomials even in this context, using the hierarchical form of Lagrange polynomials described e.g. in [16, 10]. Note that the equivalence between the two forms is true only if \mathcal{I} is chosen as downward closed, i.e.

$$\forall \mathbf{k} \in \mathcal{I}, \quad \mathbf{k} - \mathbf{e}_n \in \mathcal{I} \text{ for every } n = 1, \dots, N \text{ such that } k_n > 1.$$

see Fig. 1. The choice of implementing the combination technique instead of the hierarchical form allows to keep the data structure to a minimum, and guarantees ease of use and high-level, “close-to-the-math” coding.

Coming back to the choice of the set \mathcal{I} , the optimal choice depends on the decay rate of the detail operators, which in turn depends on the regularity of f , see e.g. [7, 41, 10]. One very classical choice of set \mathcal{I} is the following one:

$$\mathcal{I}_{\text{sum}}(w) = \left\{ \mathbf{i} \in \mathbb{N}_+^N : \sum_{n=1}^N (i_n - 1) \leq w \right\}, \quad (13)$$

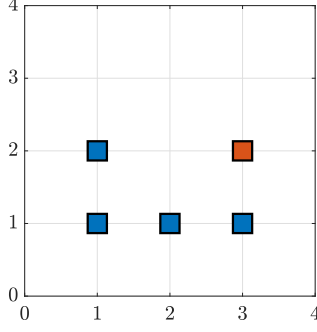


Figure 1: Downward closedness of a multi-index set. The set of the multi-indices marked in blue is downward closed. Instead, the multi-index $[3, 2]$ (in red) violates the rule in Eq. (2): the multi-index $[2, 2] = [3, 2] - \mathbf{e}_1$ is not contained in the multi-index set and hence the set of blue and red multi-indices is not downward closed.

for some $w \in \mathbb{N}$. In particular, together with the *doubling* level-to-knots, Eq. (5), this results in the famous *Smolyak* grid. Other choices are e.g.

$$\text{Tensor product set: } \mathcal{I} = \left\{ \mathbf{i} \in \mathbb{N}_+^N : \max_{n=1}^N g_n(i_n - 1) \leq w \right\}, \quad (14)$$

$$\text{Total degree set: } \mathcal{I} = \left\{ \mathbf{i} \in \mathbb{N}_+^N : \sum_{n=1}^N g_n(i_n - 1) \leq w \right\}, \quad (15)$$

$$\text{Hyperbolic cross set: } \mathcal{I} = \left\{ \mathbf{i} \in \mathbb{N}_+^N : \prod_{n=1}^N (i_n)^{g_n} \leq w \right\}, \quad (16)$$

$$\text{Multi-index box set: } \mathcal{I} = \left\{ \mathbf{i} \in \mathbb{N}_+^N : i_n \leq b_n \right\}, \quad (17)$$

for $g_1, \dots, g_N \in \mathbb{R}$, $w, b_1, \dots, b_N \in \mathbb{N}$, where, in particular, g_1, \dots, g_N are typically called *anisotropy weights* and w *sparse-grid level*. We refer e.g. to [3] for a thorough discussion about the motivations for introducing the sets above. The set \mathcal{I} could also be built adaptively based on suitable *profit* indicators; this leads to *adaptive sparse grids* algorithms, that will be discussed in Sect. 3.1.

Example 3 Let $N = 2$ and consider the downward closed multi-index set reported in Fig. 1, i.e. $\mathcal{I} = \{[1, 1], [1, 2], [2, 1], [3, 1]\}$. Let us first exemplify the combination technique form of the sparse grid approximation and quadrature in Eqs. (10) and (11), respectively. We use again the generic symbol $\mathcal{F}_{\mathbf{i}}$ to denote both $\mathcal{U}_{\mathbf{i}}$ and $\mathcal{Q}_{\mathbf{i}}$ and obtain

$$\mathcal{F}_{\mathcal{I}}(\mathbf{y}) = c_{[1,1]} \mathcal{F}_{[1,1]}(\mathbf{y}) + c_{[1,2]} \mathcal{F}_{[1,2]}(\mathbf{y}) + c_{[2,1]} \mathcal{F}_{[2,1]}(\mathbf{y}) + c_{[3,1]} \mathcal{F}_{[3,1]}(\mathbf{y})$$

with

$$\begin{aligned} c_{[1,1]} &= (-1)^{\| [0,0] \|_1} + (-1)^{\| [1,0] \|_1} + (-1)^{\| [0,1] \|_1} = -1, \\ c_{[1,2]} &= (-1)^{\| [0,0] \|_1} = +1, \\ c_{[2,1]} &= (-1)^{\| [0,0] \|_1} + (-1)^{\| [1,0] \|_1} = 0, \\ c_{[3,1]} &= (-1)^{\| [0,0] \|_1} = +1. \end{aligned}$$

Since $c_{[2,1]} = 0$, only three Lagrangian interpolant/quadrature operators explicitly appear in the combination technique formulas (10) and (11), i.e.

$$\mathcal{F}_{\mathcal{I}}(\mathbf{y}) = -\mathcal{F}_{[1,1]}(\mathbf{y}) + \mathcal{F}_{[1,2]}(\mathbf{y}) + \mathcal{F}_{[3,1]}(\mathbf{y}),$$

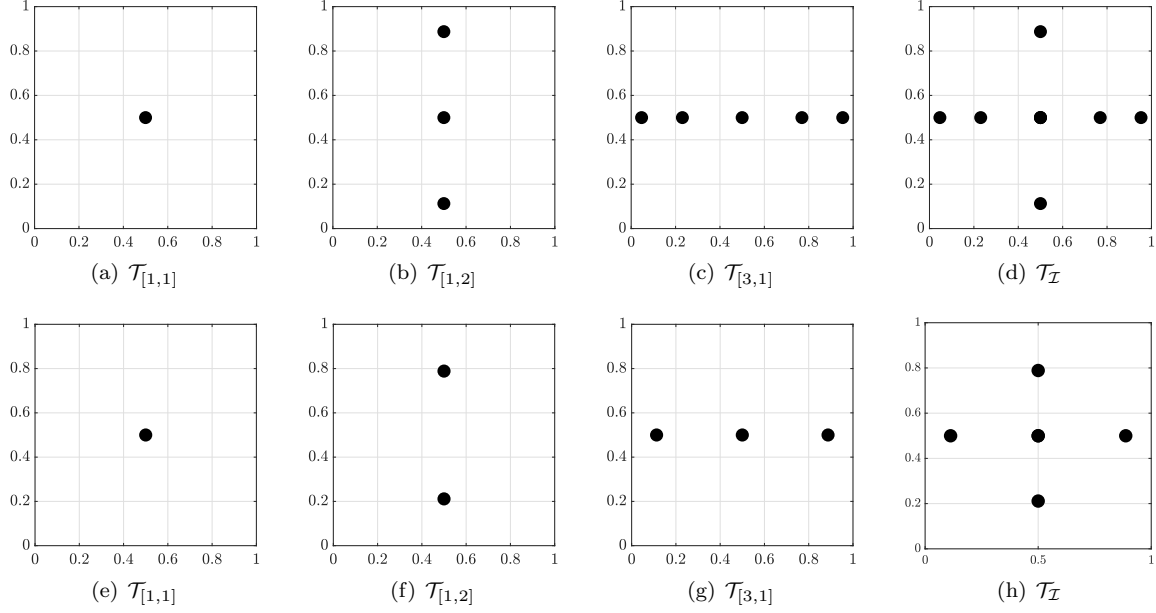


Figure 2: Top row: tensor grids (panels a,b,c) and the sparse grid (panel d) for Example 3 with level-to-knots *doubling*. Bottom row: tensor grids (panels e,f,g) and the sparse grid (panel h) for the same Example with level-to-knots *linear*.

and only the corresponding three tensor grids contribute to the sparse grid (cf. Eq. (12)). Then, considering the doubling level-to-knots function, cf. Eq. (5), the resulting tensor grids consist of one, three, and five grid knots, respectively:

$$\begin{aligned}\mathcal{T}_{[1,1]} &= \{[y_{1,1}^1 \ y_{2,1}^1]\}, \\ \mathcal{T}_{[1,2]} &= \{[y_{1,1}^1 \ y_{2,3}^1], [y_{1,1}^1 \ y_{2,3}^2], [y_{1,1}^1 \ y_{2,3}^3]\}, \\ \mathcal{T}_{[3,1]} &= \{[y_{1,5}^1 \ y_{2,1}^1], [y_{1,5}^2 \ y_{2,1}^1], [y_{1,5}^3 \ y_{2,1}^1], [y_{1,5}^4 \ y_{2,1}^1], [y_{1,5}^5 \ y_{2,1}^1]\}.\end{aligned}$$

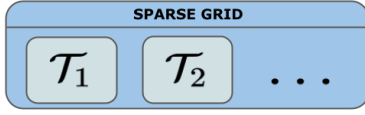
Choosing Gauss-Legendre knots as univariate collocation knots on $\Gamma_1 = \Gamma_2 = [0,1]$ leads to the tensor grids displayed in Fig. 2a,b,c and the sparse grid of Fig. 2d. If instead we consider the linear level-to-knots function, cf. Eq. (3), the three tensor grids will consist of one, two and three knots

$$\begin{aligned}\mathcal{T}_{[1,1]} &= \{[y_{1,1}^1 \ y_{2,1}^1]\}, \\ \mathcal{T}_{[1,2]} &= \{[y_{1,1}^1 \ y_{2,2}^1], [y_{1,1}^1 \ y_{2,2}^2]\}, \\ \mathcal{T}_{[3,1]} &= \{[y_{1,3}^1 \ y_{2,1}^1], [y_{1,3}^2 \ y_{2,1}^1], [y_{1,3}^3 \ y_{2,1}^1]\}.\end{aligned}$$

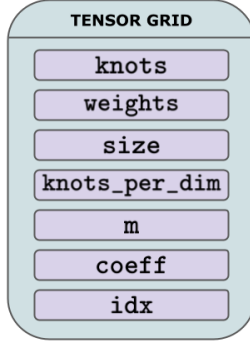
and the same choice of Gauss-Legendre knots as univariate collocation knots on $\Gamma_1 = \Gamma_2 = [0,1]$ leads to the tensor grids displayed in Fig. 2e,f,g and the sparse grid of Fig. 2h.

3 The Sparse Grids Matlab Kit: Sparse grid data structure

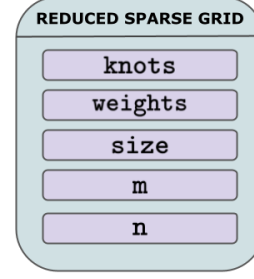
As described in the previous section, defining a sparse grid requires choosing a family \mathcal{T}_{n,i_n} of collocation knots for each variable y_n , a level-to-knots function $m(\cdot)$ and a multi-index set \mathcal{I} to be input in the combination technique formulas of Eqs. (10) and (11). In the Sparse Grids Matlab Kit, the same steps are to be followed to define a sparse grid, as can be seen in Listing 1, which creates the second sparse grid of Example 3. In



(a) Sparse grid structure array



(b) Tensor grid structure



(c) Reduced sparse grid structure

Figure 3: Sparse grid data structure: a sparse grid is stored in a structure array, each structure corresponding to one tensor grid and collecting seven fields that identify the current grid.

Listing 1, Line 1 defines the function to be used to compute the collocation knots, Line 2 sets the level-to-knots function, Line 3 specifies the multi-index set and finally Line 7 creates the sparse grid. The purpose of Line 8 will be made clearer in a moment.

```

1 knots = @(n) knots_uniform(n,0,1);    % Gauss-Legendre knots on [0,1]
2 lev2knots = @lev2knots_lin;
3 I = [1 1;
4       1 2;
5       2 1;
6       3 1];
7 S = create_sparse_grid_multiidx_set(I,knots,lev2knots);
8 Sr = reduce_sparse_grids(S);

```

Listing 1: Basic creation of a sparse grid.

As already discussed, the knots should be chosen according to the distribution of each random variable y_n , then e.g. the choice above is appropriate for uniform random variables. The Sparse Grids Matlab Kit supports uniform, normal, exponential, gamma, beta and triangular distributions, and provides at least two choices of collocation knots for almost all of them (namely, Gaussian and weighted Leja knots, see [50, 38] respectively³); more options are provided for specific choices of random variables, such as midpoint, equispaced and Clenshaw–Curtis knots [62] for uniform random variables, and Genz–Keister [24] for normal distributions; the Sparse Grids Matlab Kit user manual [47] reports some discussion on the algorithms used to compute the knots and quadrature weights for the various choices. Several choices are also available for the level-to-knots functions, including but not limited to the ones reported in Eqs. (3)-(5), as well as for generating the multi-index sets in Eqs. (14)-(17). We refer to the Sparse Grids Matlab Kit user manual [47] for a thorough discussion on the various options available.

The data structure chosen to store sparse grids is also quite close to the mathematical formalism: following closely the definition of a sparse grid in Eq. (12) as a union of tensor grids, the Sparse Grids Matlab Kit stores a sparse grid as a **struct array**, where each **struct** of the array stores a single tensor grid (see Fig. 3a). Of course, only tensor grids whose coefficient in the combination technique formula is non-zero get stored, see Eqs. (10) and (11).

The **struct** storing a tensor grid are also quite minimal and contain only seven fields, see Fig. 3b: a matrix **knots** storing the knots of the tensor grid \mathcal{T}_i ; a vector **weights** for the corresponding quadrature weights multiplied by the combination technique coefficient of the current tensor grid, i.e., $\omega_{m(i)}c_i$, an integer **size** for the number of knots/weights, a cell array **knots_per_dim** containing the the univariate

³Gaussian collocation knots are not provided for triangular distribution.

sets of collocation knots \mathcal{T}_{n,i_n} , a vector **m** containing the number of knots in each dimension (such that **size** = **prod(m)**), a vector **idx** containing the multi-index from which the tensor grid is generated (such that **m** = **lev2knots(idx)**) and an integer **coeff** storing the coefficient c_i of the combination technique formula, see Eq. (10).

As already discussed in Example 3, the sparse grid generated by Listing 1 is formed by three tensor grids $\mathcal{T}_{[1\ 1]}$, $\mathcal{T}_{[3\ 1]}$, $\mathcal{T}_{[1\ 2]}$, with combination techniques coefficients $c_{[1\ 1]} = -1$, $c_{[3\ 1]} = 1$, $c_{[1\ 2]} = 1$. Consequently, the sparse grids structure array **s** is composed by three structures, one for each tensor grid. Conversely, the multi-index $[2\ 1]$ has combination coefficients $c_{[2\ 1]} = 0$ and therefore the grid $\mathcal{T}_{[2\ 1]}$ does not get stored:

```
>>S =
  1x3 struct array with fields:
    knots
    weights
    size
    knots_per_dim
    m
    coeff
    idx

>> S(1)
ans =
  struct with fields:
    knots: [2x1 double]
    weights: -1
    size: 1
    knots_per_dim: {[0.5000] ...
[0.5000]}
    m: [1 1]
    coeff: -1
    idx: [1 1]
>> S(1).knots
ans =
    0.5000
    0.5000

>> S(2)
ans =
  struct with fields:
    knots: [2x2 double]
    weights: [0.5000 0.5000]
    size: 2
    knots_per_dim: {[0.5000] ...
[0.2113 0.7887]}
    m: [1 2]
    coeff: 1
    idx: [1 2]
>> S(2).knots
ans =
    0.5000    0.5000
    0.2113    0.7887

>> S(3)
ans =
  struct with fields:
    knots: [2x3 double]
    weights: [0.2778 0.4444 0.2778]
    size: 3
    knots_per_dim: {[0.1127 ...
0.5000 0.8873] [0.5000]}
    m: [3 1]
    coeff: 1
    idx: [3 1]
>> S(3).knots
ans =
    0.1127    0.5000    0.8873
    0.5000    0.5000    0.5000
```

The listing above shows that the same knot $[y_1, y_2] = [0.5, 0.5]$ appears in more than one tensor grid, and is thus stored more than once; this phenomenon is even more pronounced (actually, desired!) when nested sequences of knots are used. Therefore, it is useful to have a disposal a compact representation of a sparse grid, where duplicate knots are stored only once, together with their “lumped” quadrature weight, obtained taking the linear combination of the quadrature weights of each instance of the repeated knot with the combination coefficient weights in Eq. (10). This representation is called *reduced* sparse grid and is created by the function **reduce_sparse_grid** (Line 8 of Listing 1), which essentially detects (up to a certain tolerance, tunable by the user) the identical knots, deletes the possible repetitions and computes the corresponding quadrature weights. The resulting data structure is a single **struct** (see Fig. 3c), with five fields: a matrix **knots** for all the “uniqued” collocation knots of the sparse grid, a vector **weights** for their corresponding “lumped” quadrature weights, an integer **size** containing the number of knots / weights, and finally two vectors of indices **m** and **n** mapping from the non-uniqued list of knots [**S.knots**] and the reduced version [**Sr.knots**] and vice-versa.

Note that both extended and reduced formats are useful for working with sparse grids and should always be stored in memory. This implies a certain redundancy in memory storage, and reducing a sparse grid takes a non-negligible computational time, as we further discuss in Example 4 below. However, having the two structures at hand considerably simplifies coding operations on sparse grids such as interpolation or conversion to Polynomial Chaos Expansion (these operations are described in details in Sect. 4), and hides a lot of complexity from the final user.

Remark 2 *Once the family of knots and the level-to-knots function are known, it would be in principle possible to perform the sparse grid reduction comparing indices of knots rather than coordinates, which is*

faster and does not need to use a tolerance. However, we decided to go for the admittedly slower alternative and compare coordinates rather than indices, because this makes it easier for a generic user to introduce their own family of collocation knots.

Indeed, especially for non-nested knots, determining if sets of knots at different levels (not just consecutive ones) have knots in common, i.e., assessing $\mathcal{T}_{n,k} \cap \mathcal{T}_{n,j}$ for generic choices of $k, j \in \mathbb{N}$ is not straightforward. This information is however needed if we were to use only indices when reducing a grid, thus the user would need to provide such information in a suitable format (and possibly precompute it offline – again up to a tolerance), which might be not easy to do.

Comparing coordinates instead allows much more flexibility in the way knots are provided by the user. In particular, the user does not even need to precompute knots offline, but could also just provide a function that computes them at each call, knowing that the reduce operation is robust to difference in coordinates induced by floating point arithmetics. This is actually what happens for Clenshaw–Curtis and Gaussian knots, that are not precomputed and tabulated but rather computed at each call.

Example 4 (Computational cost) Let us measure memory usage and CPU time⁴ for generating Smolyak sparse grids in reduced format for increasing $N = 2, \dots, 10$ and for fixed $w = 3$ or $w = 5$, i.e., using Eq. (13) to generate the multi-index set and the level-to-knots function in Eq. (5). In this example, we use the so-called Clenshaw–Curtis knots, a choice that ensures that the grids generated will be nested, see [47, 62]. In essence, this requires clocking a listing analogous to Listing 1 with suitable adjustments to the definitions at Lines 1–3. We display the resulting sparse grid size and the computational time in Fig. 4a, and the percentage of computational time taken by the reduction step in Fig. 4b.

We then perform the dual experiment, i.e., we measure memory usage and CPU time for generating reduced Smolyak sparse grids for fixed $N = 3$ or $N = 5$ and increasing $w = 2, \dots, 10$. Sparse grid size and total CPU time are now reported in Fig. 4c while Fig. 4d shows the percentage of time taken by the reduction step. Both the computational time and the sparse grid size can be seen to grow faster with respect to w than N . Moreover, the time taken by the reduction step is mildly impacting on the total time when keeping w to small values and increasing N (panel b), whereas steadily increasing with w (panel d). This phenomenon is partially due to the chosen level-to-knots function and using another type of level-to-knots function can be expected to result in a lower percentage of time being spent on reduction.

3.1 Adaptive sparse grid generation

The straightforward way of generating a sparse grid is by specifying *a-priori* the multi-index set \mathcal{I} , i.e. before sampling the function f . However, as already mentioned, a *greedy adaptive* approach in which the multi-index set (and hence the approximation of f) is constructed in an iterative way, relying on some heuristic criteria based on the values of the function f obtained so far, is often beneficial.

The adaptive algorithm implemented in the Sparse Grids Matlab Kit is described in details in [42] and extends the original one by Gerstner and Griebel in [25]; an alternative approach was proposed by Stoyanov and Webster in [59, 37].

Roughly speaking, the Gerstner–Griebel algorithm starts with the trivial multi-index set $\mathcal{I} = \{[1, 1, \dots, 1]\}$ and iteratively adds to \mathcal{I} the multi-index \mathbf{i} with the largest heuristic *profit* indicator choosing from a set of candidates, called *reduced margin* of \mathcal{I} and defined as follows:

$$\mathcal{R}_{\mathcal{I}} = \{\mathbf{i} \in \mathbb{N}_+^N \text{ s.t. } \mathbf{i} \notin \mathcal{I} \text{ and } \mathbf{i} - \mathbf{e}_n \in \mathcal{I} \ \forall n \in \{1, \dots, N\} \text{ s.t. } i_n > 1\}.$$

Note that the condition $\mathbf{i} \in \mathcal{R}_{\mathcal{I}}$ is requested to guarantee that $\mathcal{I} \cup \{\mathbf{i}\}$ is *downward closed*, cf. Eq. (2). The role of the profit indicator is to balance error reduction and additional computational costs brought in by each multi-index \mathbf{i} (where the cost is measured as the number of new evaluations of f needed to add \mathbf{i} to

⁴This test was carried out in Matlab 2019b on a standard laptop with processor Intel(R) Core(TM) i7-8665U CPU 2.10/4.80 GHz and 16 GB RAM

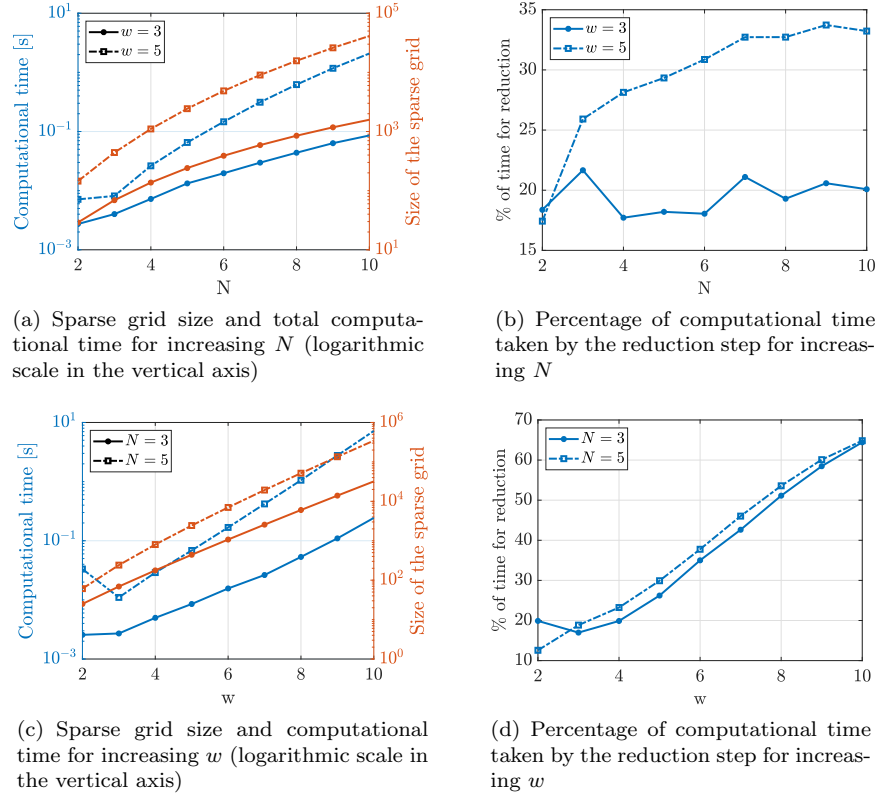


Figure 4: Computational cost and size of sparse grids for different values of N and w .

\mathcal{I}); in other words, it quantifies the fact that ideally we would like to add to the sparse grid multi-indices that carry a large reduction in interpolation/quadrature error for a minimal extra cost. Convergence of this algorithm was recently proved for certain classes of problems, see [16, 21].

The computation of the profit of a multi-index \mathbf{i} actually requires *evaluating* f at the new collocation knots that would be added to the sparse grid: this justifies why the algorithm is typically referred to as *a-posteriori* adaptive algorithm; this is in sense a sub-optimal procedure, since a certain computational work is invested in assessing the profit of multi-indices which might then turn out to be “useless”. A variant of the algorithm where the computation of the profit is based instead on *error estimators* (that do not require evaluating f at the new knots) is proposed in [28]; this version is of course only valid for the class of problems that admit said error estimator.

We also mention that this algorithm is sometimes referred to as *dimension-adaptive*: indeed, in the framework introduced so far, the algorithm will add more knots in the variables that are deemed more important, but these knots are spread throughout the whole support of the random variables (any clustering being a consequence only of the marginal pdfs ρ_1, \dots, ρ_N) rather than localized in certain regions of the support where the algorithm has detected local features of f . The latter algorithm is the *locally-adaptive* one that we already mentioned in the previous section and can be obtained with suitable modifications of the framework discussed here; for more information, we refer again to [45, 15, 43, 34].

The Sparse Grids Matlab Kit extends the Gerstner–Griebel *a-posteriori dimension-adaptive* algorithm in several ways:

- it can use non-nested knots;
- it can operate on vector-valued functions;

- it can use several profit definitions (see user manual);
- it improves the performance of the Gerstner–Griebel algorithm for when the function f at hand is “very high-dimensional”, $N \gg 1$, by implementing the so-called *dimension-buffering*, see [42, 47]. Indeed, when $N \gg 1$, the size of the reduced margin $\mathcal{R}_{\mathcal{I}}$ grows very quickly, which in turn implies a quick growth of the number of evaluations of f . In this case, if we know that y_1, \dots, y_N are “sorted decreasingly according to their importance”⁵, the Sparse Grids Matlab Kit adaptive algorithm starts by exploring only an initial subset of dimensions (the most relevant ones) and then gradually adds more dimensions to the approximation, thus limiting the number of indices in the candidate set. More specifically, the algorithm splits the random variables in three groups: *activated*, *buffered* (or non-activated), and *neglected*. A variable y_b is said to be *buffered* if the algorithm has computed the profit of the “first non-trivial multi-index” in random variable y_k , i.e., of $\mathbf{b}_k = [1 \ 1 \ 1 \ \dots] + \mathbf{e}_k$ but \mathbf{b}_k has not been selected yet (i.e., its profit is not the highest one in the list of candidates); when \mathbf{b}_k gets selected, \mathbf{b}_k is moved from the list of candidates to \mathcal{I} and the variable y_k becomes *activated*. Then, when the adaptive algorithm is run in “buffered mode”, with N_{buf} variables:

- it begins considering only $N_{cur} = N_{buf}$ variables $y_1, \dots, y_{N_{cur}}$, with list of candidates $\mathcal{R}_{\mathcal{I}} = \{\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{N_{buf}}\}$; the other variables $y_{N_{buf}+1}, y_{N_{buf}+2}, \dots$ are *neglected*;
- as soon as \mathbf{b}_k gets selected (i.e., y_k becomes *activated*) for some $k \in \{1, \dots, N_{cur}\}$, the algorithm buffers the first *neglected* variable $\mathbf{b}_{N_{cur}+1}$, i.e., $\mathbf{b}_{N_{cur}+1}$ is added to list of candidates $\mathcal{R}_{\mathcal{I}}$, and the number of current variables N_{cur} is increased by 1.

In this way, at each iteration the algorithm is forced to explore candidates with at most N_{buf} *buffered* variables. This approach is also discussed in [10, 52], but only for the special case $N_{buf} = 1$.

4 The Sparse Grids Matlab Kit: operations on sparse grids

The Sparse Grids Matlab Kit implements a number of operations on sparse grids:

- evaluation of f over the collocation knots of the sparse grid;
- interpolation (cf. Remark 1) and quadrature; these operation in practice are the solution to the problems of approximating and integrating f that were the motivation for introducing sparse grids in the first place, cf. beginning of Sect. 2;
- computation of gradients and Hessians (by finite differences) of the sparse grid interpolant;
- conversion to Polynomial Chaos Expansions (PCE) and computation of PCE-based Sobol indices for sensitivity analysis.

We discuss below the most noteworthy features implemented in the code, and refer the reader to the manual [47] for a thorough discussion of each functionality with practical examples.

4.1 Evaluation recycling

Evaluation of f on the collocation knots of a sparse grid is of course conceptually straightforward – it is simply a matter of looping through the knots of a sparse grid and calling the evaluation of f on each of them. To this end, the Sparse Grids Matlab Kit provides a convenience wrapper function (**evaluate_on_sparse_grids**), to which f is passed as anonymous function (@-functions in Matlab). This wrapper can take as input a list of collocation knots where the function has already been evaluated (either another sparse grid or e.g. coming

⁵This might be e.g. the case when f is the solution of a PDE with uncertain coefficients represented by a Karhunen–Loève expansion.

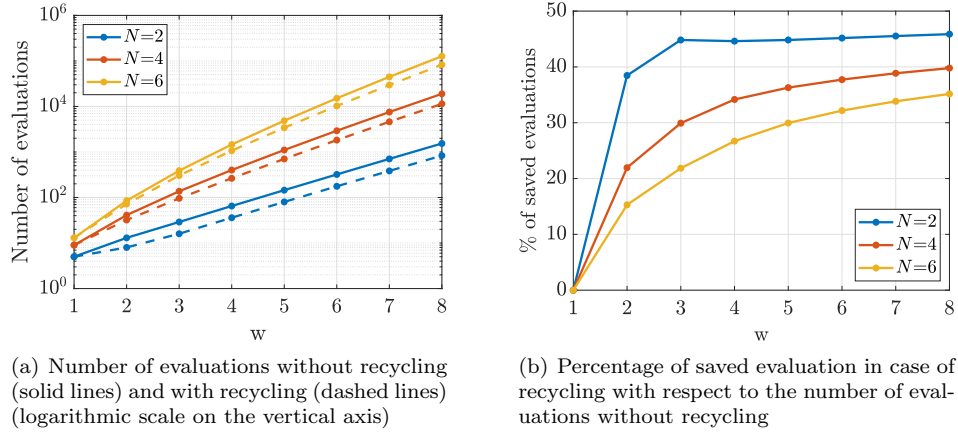


Figure 5: Assessment of the recycling functionality: test for different values of w .

from a Monte Carlo sampling), and will detect which ones of these evaluations can be “recycled”, to reduce the amount of calls to f . This algorithm proceeds in two different ways depending on whether the list of knots is another sparse grid or not.

- If the list of knots is actually another sparse grid (which needs to be passed both in reduced and non-reduced format): this fact is used to speed up the search for knots in common, by comparing essentially the multi-indices in the two grids and then using the vectors \mathbf{m} and \mathbf{n} from the reduced format (see Sect. 3) rather than comparing the actual coordinates of the knots (i.e. comparing mostly integer numbers, which is of course faster than comparing floating point numbers). Of course, larger savings are obtained if nested collocation knots are used when generating the two grids.
- If the list of knots is unstructured (e.g., a simple list of knots): the same algorithm used when reducing a sparse grid is employed to compare the coordinates of the knots in the list with the knots in the sparse grid.

Example 5 (Recycling) In Fig. 5a we compare the number of evaluations of f over a Smolyak grid with Clenshaw–Curtis (i.e. nested) knots for $N = 2, 4, 6$ and increasing values of the level w , with and without recycling from the previous grid (i.e., recycling the evaluations at level $w - 1$ to evaluate f over the grid at level w). The same information is shown in Fig. 5b, where we display the percentage of evaluations saved when making use of the recycling functionality, and indeed observe that in this case the saving is considerable (30 – 50% of the evaluations). However, note that this amount depends on the type of multi-index set, the level-to-knots function and the type of knots used (nested/non-nested).

4.2 Parallelization

Having wrapped evaluations of f in the dedicated function `evaluate_on_sparse_grids` also allows to switch on/off the parallel toolbox of Matlab to allocate the evaluations of f on the workers available in the Matlab session in a way that is transparent to the user. As soon as at least N_{par} evaluations of f are requested, the code takes care of changing from `for` loop (serial execution) to `parfor` loop (parallel execution): the user controls only the value of N_{par} , that should be set depending on the CPU time required by a single evaluation of f . Indeed, for very fast evaluations of f the communication time between the parallel workers and the central Matlab instance might exceed the evaluation time. We conclude this paragraph mentioning some technical aspects:

- The allocation of the evaluations of f on the workers is completely delegated to the built-in scheduler of Matlab, and, in particular, this means that we are assuming that evaluating f requires the same CPU time for every value of \mathbf{y} (which is not necessarily the case when f is the result of a complex PDE solver).
- The only part of the Sparse Grids Matlab Kit that makes explicit use of the Matlab parallel environment is the evaluation of f over the grid knots. All other operations (sparse grid generation and reduction, other operations on sparse grid mentioned at the beginning of this section) do not have an explicit parallel implementation.
- If the adaptive algorithm is used, at each iteration several candidate indices are tested by adding them to the current sparse grid. Then, the question rises whether to consider candidate indices *sequentially* and then execute **parfor** loops only on the knots requested by each index or conversely to gather all new knots requested by all candidates and parallelize them all in a single **parfor** loop. We implement the first strategy, since a) in this way we can use the fast version of the evaluation recycling (indeed, the union of all new knots needed is not a sparse grid, whereas by testing one index at a time we can compare two sparse grids: the one with it and without it) and b) it improves code modularity.

4.3 Interface with external software by UM-Bridge protocol

Another advantage of having wrapped evaluations of f in **evaluate_on_sparse_grids** is that in principle external software for evaluating f can also be simply connected to the Sparse Grids Matlab Kit by e.g. encapsulating system calls to such solver in the @-functions) taken as inputs by **evaluate_on_sparse_grids**.

A particularly efficient way of doing so is through the UM-Bridge software [53], which implements a standardized HTTP protocol (available in several languages, including Matlab) to put in communication UQ software with complex software for evaluating $f(\mathbf{y})$ (e.g. when $f(\mathbf{y})$ is the numerical solution of a PDE). The fact that communication is via HTTP messages also allows interfacing the Sparse Grids Matlab Kit with software running on remote servers. In particular, if such server allows parallel requests, activating Matlab **parfor** as discussed above will automatically enable parallelism on the server. More specifically, the Sparse Grids Matlab Kit would control e.g. M Matlab workers running on a laptop, whose only task is to transparently control P processors each on the remote server, solving the model in parallel. Since the workload of the Matlab workers is minimal (just sending an HTTP message), one can activate $M \gg 1$ workers, thus opening the door to large-scale UQ applications. Connecting UM-Bridge and the Sparse Grids Matlab Kit takes 5 lines of code, see manual [47] for a minimal example and [54] for a naval engineering UQ application, in which the solver runs remotely in parallel on the Google Cloud Platform.

4.4 Interpolation and quadrature

The interpolation procedure is provided by the function **interpolate_on_sparse_grid** and consists in evaluating Eq. (10). The implementation follows closely the mathematical formulation:

- it loops through the tensor grids, creating a Lagrange interpolant of f on each tensor grid and evaluating them at the requested knots (the standard form of Lagrange polynomials is implemented in the Sparse Grids Matlab Kit – another possibility would be to implement their barycentric form);
- the evaluations on each tensor grid are combined with the combination technique coefficients c_i .

Note that for this operation both the *extended* and *reduced* versions of the sparse grid are needed. This is because of two reasons:

1. the information about tensor grids (knots, combination technique coefficients) is available only in the *extended* format;

2. the values of f on the sparse grid knots (needed to evaluate the Lagrange interpolants) provided by **evaluate_on_sparse_grid** are stored in containers (vectors for scalar-valued f , matrices for vector-valued f) whose number of elements is the number of unique knots in the *reduced* grid; in other words, the values of f are available in compressed format. However, to compute the Lagrange interpolant on each tensor grid one then needs to know where the value of f at each node of each tensor grid is stored in such container: to this end, we need to use the vectors \mathbf{m} and \mathbf{n} provided by the *reduced* grid format.

Similarly, the quadrature function **quadrature_on_sparse_grid** provides an implementation for the quadrature formula in Eq. (11). Note however that the implementation of **quadrature_on_sparse_grid** actually does not require looping through the tensor grids, therefore it does not need the *extended* and *reduced* format, but only the *reduced* one. Indeed, the quadrature weights in the reduced format of the sparse grid already take into account both the possible occurrences of the knots in multiple grids as well as the combination technique coefficients, therefore the only operation that **quadrature_on_sparse_grid** needs to implement is the linear combination of the evaluations of f with the quadrature weights stored in the reduced grid format.

4.5 Polynomial Chaos Expansion and Sobol indices computation

The sparse-grid approximation of a function f is based on Lagrange interpolation polynomials, and hence is a *nodal* approximation. However, sometimes it is interesting to work with *modal* approximations instead, specifically with the generalized Polynomial Chaos Expansion (gPCE) [65, 17, 26, 60], i.e., an expansion of f over multi-variate ρ -orthonormal polynomials [23, 61]:

$$f(\mathbf{y}) \approx \sum_{\mathbf{p} \in \Lambda} d_{\mathbf{p}} \mathcal{P}_{\mathbf{p}}(\mathbf{y}), \quad (18)$$

where $\Lambda \subset \mathbb{N}^N$ is a multi-index set⁶ and $\mathcal{P}_{\mathbf{p}} = \prod_{n=1}^N P_{p_n}(y_n)$ are products of N univariate ρ_n -orthonormal polynomials of degree p_n . Similarly to the multi-index set \mathcal{I} on which a sparse grid is based, also the multi-index set Λ can be prescribed either *a-priori* based on the regularity of f [3, 55] or *adaptively* [6, 9]; the coefficients $d_{\mathbf{p}}$ can be computed in several ways, e.g. by quadrature [64], least squares fitting [6], or compressed sensing approaches [29].

The strategy provided by the Sparse Grids Matlab Kit consists in computing both the multi-index set Λ and the coefficients $d_{\mathbf{p}}$ by re-expressing the sparse-grid interpolant over the ρ -orthogonal basis of choice; in other words, the Sparse Grids Matlab Kit performs a change of basis to represent the same polynomial from a linear combination of Lagrange polynomials (the sparse-grid interpolant) to a linear combination of ρ -orthogonal polynomials. The algorithm that performs the conversion was introduced in [22] (see [12] for a similar approach) and proceeds in two steps:

- each tensor interpolant $\mathcal{U}_{\mathbf{i}}$ in the sparse-grid approximation, cf. Eq. (10), is converted into a linear combination of N -variate ρ -orthogonal polynomials, which requires solving the following Vandermonde-like linear system for each tensor interpolant (see [22] for details):

$$\sum_{\substack{\mathbf{p} \in \mathbb{N}^N: \\ \mathbf{p} \leq m(\mathbf{i}) - \mathbf{1}}} \tilde{d}_{\mathbf{p}} \mathcal{P}_{\mathbf{p}}(\mathbf{y}_k) = \mathcal{U}_{\mathbf{i}}(\mathbf{y}_k) \quad \forall \mathbf{y}_k \in \mathcal{T}_{\mathbf{i}};$$

- if the same ρ -orthogonal polynomial $\mathcal{P}_{\mathbf{p}}$ is generated by more than one tensor interpolant, the corresponding coefficient $d_{\mathbf{p}}$ in the final gPCE expansion (18) is the linear combination of the partial coefficients $\tilde{d}_{\mathbf{p}}$ with coefficients $c_{\mathbf{i}}$ of the combination technique, see again Eq. (10).

Also in this case, we need to loop over the tensor grids, therefore both the *extended* and *reduced* format of a sparse grid are needed in the implementation. Note that the algorithm works in such a way that Λ

⁶Note that here multi-indices can have entries with value zero.

is completely determined by the choices of the level-to-knots function and of the multi-index set of the sparse grid from which the conversion procedure begins. The condition number of the Vandermonde-like linear systems depends on the choice of the families of collocation knots used to build the tensor grids. In particular, the matrix becomes orthogonal if the tensor grids are built using ρ -Gaussian collocation knots and we want to compute the gPCE expansion over the corresponding ρ -orthogonal polynomials (e.g. Gauss–Legendre knots and Legendre polynomials, Gauss–Hermite knots and Hermite polynomials, etc); the drawback of using ρ -Gaussian collocation knots is that they are typically non-nested.

The Sparse Grids Matlab Kit (function `convert_to_modal`) supports conversion to Legendre, Hermite, Laguerre, generalized Laguerre, and probabilistic Jacobi polynomials, which are the ρ -orthogonal polynomials for uniform, normal, exponential, gamma, and beta probability density functions (all random variables for which the Sparse Grids Matlab Kit provides collocation knots for generation of the corresponding sparse grids). Moreover, conversion to Chebyshev polynomials is also available, since they are a valid alternative to Legendre polynomials for expanding functions with respect to the uniform measure, even though they are not orthogonal with respect to it. The evaluation of the orthonormal polynomials is obtained by means of the well-known three-term recursive formulas, see [23].

An example of a situation when having the gPCE expansion of f is helpful is the computation of the Sobol indices for global sensitivity analysis of f [56, 2]. An efficient way to compute such Sobol indices is indeed to perform some algebraic manipulations on the coefficients of the gPCE, c_p , see [60, 22]; to this end, the Sparse Grids Matlab Kit provides a wrapper function `compute_sobol_indices_from_sparse_grid` which calls `convert_to_modal` and performs such algebraic manipulations. Another reason to perform the conversion to gPCE is to inspect the spectral content of the sparse grid approximation, to verify how much the nodal representation is storing “redundant information”, see e.g. [18].

5 Comparison with other software

In this section, we provide a comparison of the Matlab software for sparse grids and sparse-grids-based UQ, either natively written in Matlab or that provides an interface to Matlab, see Tab. 2. With reference to Tab. 1, we thus compare the Sparse Grids Matlab Kit with SG++, Tasmanian and Spinterp; we neglect instead UQLab, since the latter does not provide full sparse grids functionalities (more precisely, it provides sparse grids quadrature but not sparse grids interpolation). We focus on a comparison in terms of functionalities rather than on computational efficiency since the typical CPU-intensive utilization scenario of this kind of software is the construction of surrogate models for UQ (as well as PDE-based optimization) purposes, in which case the computational cost is largely dominated by the evaluation of the function f , and only a small fraction of cost is ascribable to the actual sparse grid functionalities.

The comparison table is divided in several “thematic” blocks. We begin by providing the essential software information: native Matlab/interface and whether the software is currently maintained or not. We then move to comparing the basics features of sparse grids provided by each piece of software, as discussed in Sect. 2: the sparse grid forms implemented (combination technique / hierarchical), the supported basis functions and knots – more specifically whether non-nested knots can be used. The third block focuses on adaptive algorithms: the dimension-adaptive and locally-adaptive (cf. Sect. 2 and 3.1), and some connected features, most notably the *dimension buffering* discussed in Sect. 3.1. Next, we move to features connected to evaluations of f : the evaluation recycling (cf. Sect. 4.1), parallel evaluation (cf. Sect. 4.2), and possibility to connect to external software to evaluate f (cf. Sect. 4.3). The last two blocks deal with additional features: derivatives (i.e., gradients and Hessians) and UQ functionalities.

The main take-away point of this comparison is that the four software have actually quite little overlap: they all provide Lagrange-based interpolation and gradient computation, and support recycling evaluation (which in a way can be considered the minimum requirement if a piece of software wants to be any usable for practical purposes) and dimension-adaptivity. Other than this, they all come with their own sets on unique features. In general, Tasmanian and the Sparse Grids Matlab Kit have more UQ functionalities, whereas SG++ can be thought as a more generalistic sparse grids software, since it provides also additional modules

that implement functionalities for PDE solving and data mining (not discussed here). Spinterp provides an interesting implementation of the dimension-adaptive algorithm, that a) blends the Gerstner–Griebel and the Smolyak a-priori grid construction by setting a so-called *balancing* parameter that can range from 1 (100% of sparse grid knot generated by the adaptive algorithm) to 0 (Smolyak construction); b) can drop multi-indices added to the approximation with little profit, see [32]. Tasmanian provides the largest choice of basis functions, whereas Sparse Grids Matlab Kit has some unique functionalities for UQ: dimension buffering for adaptivity, supports PCE and Sobol indices, as well as computation of Hessians which could be useful e.g. in parameter identification by Bayesian approaches (see e.g. [49] for more details).

6 Conclusions

In this manuscript we have introduced the combination technique form of the sparse grid methodology for approximating and computing integrals of high-dimensional functions, in particular for UQ purposes. The Sparse Grids Matlab Kit is a Matlab software that can be used to this end. We have discussed the data structure of the software and the mathematical aspects of the functionalities implemented in it, and we have compared it with other Matlab software for sparse grids and UQ (Spinterp, Tasmanian, SG++). Compared to alternative software, the Sparse Grids Matlab Kit is the one providing most tools for UQ, such as dimension-buffering for adaptivity, support for PCE and Sobol indices.

Acknowledgments

Lorenzo Tamellini and Chiara Piazzola have been supported by the PRIN 2017 project 201752HKKH8 “Numerical Analysis for Full and Reduced Order Methods for the efficient and accurate solution of complex systems governed by Partial Differential Equations (NA-FROM-PDEs)”. Lorenzo Tamellini has been also supported by the Research program CN00000013 “National Centre for HPC, Big Data and Quantum Computing – Spoke 6 - Multiscale Modelling & Engineering Applications”. Chiara Piazzola has been also supported by the Alexander von Humboldt Foundation. The authors gratefully acknowledge several persons who contributed to the development of the package either by providing implementation for some functions or by using the software and reporting success cases, bugs and missing features. In particular: Fabio Nobile (early version of the code and continued support throughout the development of the project), Alessandra Sordi and Maria Luisa Viticchiè (early contributions to the code), Francesco Tesei and Diane Guignard (adaptive sparse grids), Giovanni Porta (Sobol indices and conversion to PCE), Björn Sprungk (adaptive sparse grids and weighted Leja knots), Francesca Bonizzoni (compatibility with Octave). Finally, we thank Miroslav Stoyanov and Dirk Pflüger for the help in crafting Tab. 2.

Feature	Tasmanian	SG++	spinterp	Sparse Grids Matlab Kit
Native Mat- lab/Interface	interface	interface	native	native
Currently maintained	yes	yes	no	yes
Combitec / hierarchical	both	mainly hierarchical ^b	hierarchical	combitec
Lagrange basis	yes	yes	yes	yes
Piecewise pol. basis	yes	yes	yes	no
Splines basis	no	yes	no	no
Trigonometric basis	yes	no	no	no
Non-nested knots?	yes	no	no	yes
Dimension- adaptive	Webster- Stoyanov	Gerstner- Griebel	Gerstner- Griebel [‡]	Gerstner- Griebel
with non-nested knots	no	no	no	yes
Buffering	implicit [†]	no	no	yes
Local adaptivity	yes	yes	no	no
Parallel eval. of f	OpenMP, CUDA/Hip	OpenMP	no	Matlab Parallel Toolbox
knot recycling	yes	yes	yes	yes
Connection to external f	yes, through LibEnsamble [♣]	no [#]	no [◇]	yes, through UM-Bridge
Gradients	yes	yes, in optimization module	yes (exact)	yes
Hessian	no	no	no	yes
Random vars. beyond uniform?	yes	no [#]	no	yes
Computation of PCE	no	no [#]	no	yes
Computation of Sobol idx	no	yes	no	yes

Table 2: Comparative table of Matlab software for sparse grids and sparse-grids-based uncertainty quantification. Annotations:

^b : limited support for combitec provided by the combitec module

[‡] : with blending of dimension-adaptive and Smolyak construction, and dropping of multi-indices with small profit [32]

[†] : once the anisotropy estimate in the Webster–Stoyanov algorithm is reliable

[♣] : <https://libensemble.readthedocs.io>

[#] : supported through interface with Dakota [1], but not in the Matlab interface;

[◇] : only through Matlab system calls

References

- [1] B. Adams, W. Bohnhoff, K. Dalbey, M. Ebeida, J. Eddy, M. Eldred, R. Hooper, P. Hough, K. Hu, J. Jakeman, M. Khalil, K. Maupin, J. Monschke, E. Ridgway, A. Rushdi, D. Seidl, J. Stephens, L. Swiler, and J. Winokur. Dakota, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 6.15 user’s manual. Technical report, Sandia National Laboratories, November 2021.
- [2] G. E. B. Archer, A. Saltelli, and I. M. Sobol. Sensitivity measures, anova-like techniques and the use of bootstrap. *Journal of Statistical Computation and Simulation*, 58(2):99–120, 1997.
- [3] J. Bäck, F. Nobile, L. Tamellini, and R. Tempone. Stochastic spectral Galerkin and collocation methods for PDEs with random coefficients: a numerical comparison. In *Spectral and High Order Methods for Partial Differential Equations*, volume 76 of *Lecture Notes in Computational Science and Engineering*, pages 43–62. Springer, 2011.
- [4] M. Baudin, A. Dutfoy, B. Iooss, and A.-L. Popelin. *OpenTURNS: An Industrial Software for Uncertainty Quantification in Simulation*, pages 1–38. Springer International Publishing, Cham, 2017.
- [5] Blanchard, Jean-Baptiste, Damblin, Guillaume, Martinez, Jean-Marc, Arnaud, Gilles, and Gaudier, Fabrice. The uranie platform: an open-source software for optimisation, meta-modelling and uncertainty analysis. *EPJ Nuclear Sci. Technol.*, 5:4, 2019.
- [6] B. Blatman, G. Sudret. Adaptive sparse polynomial chaos expansion based on least angle regression. *Journal of Computational Physics*, 230(6):2345 – 2367, 2011.
- [7] H. J. Bungartz and M. Griebel. Sparse grids. *Acta Numer.*, 13:147–269, 2004.
- [8] M. Chiappetta, C. Piazzola, M. Carraturo, L. Tamellini, A. Reali, and F. Auricchio. Sparse-grids uncertainty quantification of part-scale additive manufacturing processes. *ArXiv*, (2210.06839), 2022.
- [9] A. Chkifa, A. Cohen, R. Devore, and C. Schwab. Sparse adaptive Taylor approximation algorithms for parametric and stochastic elliptic PDEs. *ESAIM: Mathematical Modelling and Numerical Analysis*, 47(1):253–280, 2013.
- [10] A. Chkifa, A. Cohen, and C. Schwab. High-dimensional adaptive sparse polynomial interpolation and applications to parametric PDEs. *Foundations of Computational Mathematics*, 14(4):601–633, 2014.
- [11] I. Colombo, F. Nobile, G. Porta, A. Scotti, and L. Tamellini. Uncertainty Quantification of geochemical and mechanical compaction in layered sedimentary basins. *Computer Methods in Applied Mechanics and Engineering*, 328:122–146, 2018.
- [12] P. Constantine, M. S. Eldred, and E. T. Phipps. Sparse pseudospectral approximation method. *Comput. Methods Appl. Mech. Engrg.*, 229/232:1–12, 2012.
- [13] B. Debusschere, K. Sargsyan, C. Safta, and K. Chowdhary. Uncertainty Quantification Toolkit (UQTK). In R. Ghanem, D. Higdon, and H. Owhadi, editors, *Handbook of Uncertainty Quantification*, pages 1807–1827. Springer International Publishing, Cham, 2017.
- [14] B. J. Debusschere, H. N. Najm, P. P. Pébay, O. M. Knio, R. G. Ghanem, and O. P. Le Maître. Numerical Challenges in the Use of Polynomial Chaos Representations for Stochastic Processes. *SIAM Journal on Scientific Computing*, 26(2):698–719, 2004.
- [15] A. Eftekhari and S. Scheidegger. High-dimensional dynamic stochastic model representation. *SIAM Journal on Scientific Computing*, 44(3):C210–C236, 2022.

- [16] M. Eigel, O. G. Ernst, B. Sprungk, and L. Tamellini. On the convergence of adaptive stochastic collocation for elliptic partial differential equations with affine diffusion. *SIAM Journal on Numerical Analysis*, 60(2):659–687, 2022.
- [17] O. G. Ernst, A. Mugler, H.-J. Starkloff, and E. Ullmann. On the convergence of generalized polynomial chaos expansions. *ESAIM: Mathematical Modelling and Numerical Analysis*, 46(02):317–339, 2012.
- [18] O. G. Ernst, B. Sprungk, and L. Tamellini. Convergence of Sparse Collocation for Functions of Countably Many Gaussian Random Variables (with Application to Lognormal Elliptic Diffusion Problems). *SIAM Journal on Numerical Analysis*, 56(2):877–905, 2018.
- [19] J. Feinberg, V. G. Eck, and H. P. Langtangen. Multivariate polynomial chaos expansions with dependent variables. *SIAM Journal on Scientific Computing*, 40:199–223, 2018.
- [20] J. Feinberg and H. P. Langtangen. Chaospy: an open source tool for designing methods of uncertainty quantification. *Journal of Computational Science*, 11:46–57, 2015.
- [21] M. Feischl and A. Scaglioni. Convergence of adaptive stochastic collocation with finite elements. *Computers & Mathematics with Applications*, 98:139–156, 2021.
- [22] L. Formaggia, A. Guadagnini, I. Imperiali, V. Lever, G. Porta, M. Riva, A. Scotti, and L. Tamellini. Global sensitivity analysis through polynomial chaos expansion of a basin-scale geochemical compaction model. *Computational Geosciences*, 17(1):25–42, 2013.
- [23] W. Gautschi. *Orthogonal Polynomials: Computation and Approximation*. Oxford University Press, Oxford, 2004.
- [24] A. Genz and B. D. Keister. Fully symmetric interpolatory rules for multiple integrals over infinite regions with Gaussian weight. *J. Comput. Appl. Math.*, 71(2):299–309, 1996.
- [25] T. Gerstner and M. Griebel. Dimension-adaptive tensor-product quadrature. *Computing*, 71(1):65–87, 2003.
- [26] R. Ghanem, D. Higdon, and H. Owhadi. *Handbook of Uncertainty Quantification*. Handbook of Uncertainty Quantification. Springer International Publishing, 2016.
- [27] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, North Holland, 1992.
- [28] D. Guignard and F. Nobile. A posteriori error estimation for the stochastic collocation finite element method. *SIAM Journal on Numerical Analysis*, 56(5):3121–3143, 2018.
- [29] J. Hampton and A. Doostan. Compressive sampling of polynomial chaos expansions: Convergence analysis and sampling strategies. *Journal of Computational Physics*, 280:363 – 386, 2015.
- [30] J. D. Jakeman. Pyapprox: Enabling efficient model analysis. *OSTI Technical Report*, (1879614), 8 2022.
- [31] A. Klimke. *Uncertainty modeling using fuzzy arithmetic and sparse grids*. PhD thesis, Universität Stuttgart, Shaker Verlag, Aachen, 2006.
- [32] A. Klimke. Sparse grid interpolation toolbox user’s guide v. 5.1. Technical Report 2007/17, Universität Stuttgart, 2008.
- [33] A. Klimke and B. Wohlmuth. Algorithm 847: Spinterp: Piecewise multilinear hierarchical sparse grid interpolation in matlab. *ACM Trans. Math. Softw.*, 31(4):561–579, dec 2005.

- [34] X. Ma and N. Zabaras. An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations. *Journal of Computational Physics*, 229(10):3884–3915, 2010.
- [35] S. Marelli and B. Sudret. UQLab: A Framework for Uncertainty Quantification in Matlab. In M. Beer, S.-K. Au, and J. W. Hall, editors, *Vulnerability, Uncertainty, and Risk*, pages 2554–2563. American Society of Civil Engineers, 2014.
- [36] J. Martínez-Frutos and F. Periago. *Optimal Control of PDEs under Uncertainty: An introduction with application to optimal shape design of structures*. Springer International Publishing, 2018.
- [37] Z. Morrow and M. Stoyanov. A method for dimensionally adaptive sparse trigonometric interpolation of periodic functions. *SIAM Journal on Scientific Computing*, 42(4):A2436–A2460, 2020.
- [38] A. Narayan and J. D. Jakeman. Adaptive Leja Sparse Grid Constructions for Stochastic Collocation and High-Dimensional Approximation. *SIAM Journal on Scientific Computing*, 36(6):A2952–A2983, 2014.
- [39] A. Narayan, Z. Liu, J. A. Bergquist, C. Charlebois, S. Rampersad, L. Rupp, D. Brooks, D. White, J. Tate, and R. S. MacLeod. UncertainSCI: Uncertainty quantification for computational models in biomedicine and bioengineering. *Computers in Biology and Medicine*, 152:106407, 2023.
- [40] R. B. Nelsen. *An introduction to copulas*. Springer Series in Statistics. Springer, New York, second edition, 2006.
- [41] F. Nobile, L. Tamellini, and R. Tempone. Convergence of quasi-optimal sparse-grid approximation of Hilbert-space-valued functions: application to random elliptic PDEs. *Numerische Mathematik*, 134(2):343–388, 2016.
- [42] F. Nobile, L. Tamellini, F. Tesei, and R. Tempone. An adaptive sparse grid algorithm for elliptic PDEs with lognormal diffusion coefficient. In J. Garcke and D. Pflüger, editors, *Sparse Grids and Applications – Stuttgart 2014*, volume 109 of *Lecture Notes in Computational Science and Engineering*, pages 191–220. Springer International Publishing Switzerland, 2016.
- [43] M. Obersteiner and H.-J. Bungartz. A generalized spatially adaptive sparse grid combination technique with dimension-wise refinement. *SIAM Journal on Scientific Computing*, 43(4):A2381–A2403, 2021.
- [44] M. Parno, A. Davis, L. Seelinger, and Y. Marzouk. Mit uncertainty quantification (MUQ) library, 2014.
- [45] D. Pflüger, B. Peherstorfer, and H.-J. Bungartz. Spatially adaptive sparse grids for high-dimensional data-driven problems. *Journal of Complexity*, 26(5):508–522, 2010. SI: HDA 2009.
- [46] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Verlag Dr. Hut, 2010.
- [47] C. Piazzola and L. Tamellini. The sparse grids matlab kit user manual - v. 23-5 robert. <https://sites.google.com/view/sparse-grids-kit>, 2023.
- [48] C. Piazzola, L. Tamellini, R. Pellegrini, R. Broglia, A. Serani, and M. Diez. Comparing Multi-Index Stochastic Collocation and Multi-Fidelity Stochastic Radial Basis Functions for Forward Uncertainty Quantification of Ship Resistance. *Engineering with Computers*, 2022.
- [49] C. Piazzola, L. Tamellini, and R. Tempone. A note on tools for prediction under uncertainty and identifiability of SIR-like dynamical systems for epidemiology. *Mathematical Biosciences*, 332:108514, 2021.
- [50] A. Quarteroni, R. Sacco, and F. Saleri. *Numerical mathematics*, volume 37 of *Texts in Applied Mathematics*. Springer-Verlag, Berlin, second edition, 2007.

- [51] M. F. Rehme, F. Franzelin, and D. Pflüger. B-splines on sparse grids for surrogates in uncertainty quantification. *Reliability Engineering & System Safety*, 209:107430, 2021.
- [52] C. Schillings and C. Schwab. Sparse, adaptive Smolyak quadratures for Bayesian inverse problems. *Inverse Problems*, 29(6), 2013.
- [53] L. Seelinger, V. Cheng-Seelinger, A. Davis, M. Parno, and A. Reinarz. UM-Bridge: Uncertainty quantification and modeling bridge. *Journal of Open Source Software*, 8(83):4748, 2023.
- [54] L. Seelinger, A. Reinarz, J. Benezech, M. B. Lykkegaard, L. Tamellini, and R. Scheichl. Lowering the Entry Bar to HPC-Scale Uncertainty Quantification. *ArXiv*, (2304.14087), 2023.
- [55] J. Shen and L.-L. Wang. Sparse spectral approximations of high-dimensional problems based on hyperbolic cross. *SIAM J. Numer. Anal.*, 48(3):1087–1109, 2010.
- [56] I. M. Sobol’. Sensitivity estimates for nonlinear mathematical models. *Math. Modeling Comput. Experiment*, 1(4):407–414 (1995), 1993.
- [57] M. Stoyanov. User manual: Tasmanian sparse grids. Technical Report ORNL/TM-2015/596, Oak Ridge National Laboratory, One Bethel Valley Road, Oak Ridge, TN, 2015.
- [58] M. Stoyanov. Adaptive sparse grid construction in a context of local anisotropy and multiple hierarchical parents. In *Sparse Grids and Applications-Miami 2016*, pages 175–199. Springer, 2018.
- [59] M. K. Stoyanov and C. G. Webster. A dynamically adaptive sparse grids method for quasi-optimal interpolation of multidimensional functions. *Computers & Mathematics with Applications*, 71(11):2449–2465, 2016.
- [60] B. Sudret. Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering and System Safety*, 93(7):964 – 979, 2008.
- [61] G. Szegő. *Orthogonal polynomials*. Colloquium Publications - American Mathematical Society. American Mathematical Society, 1939.
- [62] L. N. Trefethen. Is Gauss quadrature better than Clenshaw-Curtis? *SIAM Rev.*, 50(1):67–87, 2008.
- [63] G. Wasilkowski and H. Wozniakowski. Explicit cost bounds of algorithms for multivariate tensor product problems. *Journal of Complexity*, 11(1):1–56, 1995.
- [64] D. Xiu. Efficient collocational approach for parametric uncertainty analysis. *Communications in Computational Physics*, 2(2):293–309, 2007.
- [65] D. Xiu and G. Karniadakis. The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM J. Sci. Comput.*, 24(2):619–644, 2002.