Adv Comput Math (2022) 48: 10
https://doi.org/10.1007/s10444-021-09922-2

Wavelet adaptive proper orthogonal decomposition for large-scale flow data

Philipp Krah[1] · **Thomas Engels**[2] · **Kai Schneider**[3] · **Julius Reiss**[4]

Received: 10 November 2020 / Accepted: 22 December 2021 / Published online: 17 February 2022

Abstract
The proper orthogonal decomposition (POD) is a powerful classical tool in fluid mechanics used, for instance, for model reduction and extraction of coherent flow features. However, its applicability to high-resolution data, as produced by three-dimensional direct numerical simulations, is limited owing to its computational complexity. Here, we propose a wavelet-based adaptive version of the POD (the wPOD), in order to overcome this limitation. The amount of data to be analyzed is reduced by compressing them using biorthogonal wavelets, yielding a sparse representation while conveniently providing control of the compression error. Numerical analysis shows how the distinct error contributions of wavelet compression and POD truncation can be balanced under certain assumptions, allowing us to efficiently process high-resolution data from three-dimensional simulations of flow problems. Using a synthetic academic test case, we compare our algorithm with the randomized singular value decomposition. Furthermore, we demonstrate the ability of our method analyzing data of a two-dimensional wake flow and a three-dimensional flow generated by a flapping insect computed with direct numerical simulation.

Keywords Proper orthogonal decomposition · Biorthogonal wavelets · Wavelet adaptive block-based grids · Fluid dynamics · Reduced order models

1 Introduction

The proper orthogonal decomposition (POD) [47] is one of the most important methods in modern data analysis of fluid flows. For large-scale data, as typically produced by high-resolution direct numerical simulation or high-resolution

Communicated by: Yang Wang

✉ Philipp Krah
philipp.krah@tu-berlin.de

Extended author information available on the last page of the article.

imaging, the POD finds low-dimensional descriptions by approximating the snapshot data $\mathcal{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_{N_s}\}$ in terms of a few orthogonal basis functions $\boldsymbol{\Psi}_k$, called modes:

$$\tilde{\mathbf{u}}_i = \sum_{k=1}^{r} a_{ki} \boldsymbol{\Psi}_k \quad \text{with} \quad a_{ki} = \langle \mathbf{u}_i, \boldsymbol{\Psi}_k \rangle, \qquad r \ll N_s \,. \tag{1}$$

This effectively reduces the high-dimensional data $\mathcal{U}$ to a set of $N_s \times r$ coefficients a_{ki}. It is known that this choice of basis is optimal in minimizing the approximation error $\sum_i \|\mathbf{u}_i - \tilde{\mathbf{u}}_i\|$ within a linear subspace (see, for example [55] or, in the context of matrices, the Eckart Young theorem [16]). However, it is not optimal with respect to memory efficiency and applicability, as for reconstructing any approximation on $\mathcal{U}$, the high-dimensional snapshots $\mathbf{u}$ and basis vectors $\boldsymbol{\Psi}_k$ need to be stored and processed. This often makes any nonlinear observable of $\mathcal{U}$ difficult to process on a desktop computer. Therefore, in this article, we aim at reducing the amount of data that is needed to calculate $\boldsymbol{\Psi}_k$, exploiting sparsity enabled by wavelet adaptation techniques.

POD has been applied successfully in Model Order Reduction (MOR) (for a review, see [7, 8] or the lecture notes of [55] for POD-MOR). In combination with Galerkin projection methods, POD is used to obtain reduced models of discretized partial differential equations (PDEs). It is the basis of many modern decompositions in fluid dynamics, like the shifted POD [41] used for analysis of transport phenomena or the spectral [46] and multiscale POD [38], which identify coherent structures with specific energies. However, most of the applications mentioned above have been only applied in one or two spatial dimensions, since the tremendous amount of data in fluid dynamics in three dimensions makes the computation of POD modes extremely expensive, if not unfeasible. Recent attempts to improve this are going in two directions: adaptivity and randomization.

Randomized methods are inspired by randomized numerical linear algebra (see the surveys of [27, 34]) using random projection matrices to approximate the column space of a possible tall and skinny snapshot matrix build from $\mathcal{U}$ onto a smaller surrogate matrix to solve the classical POD problem using snapshot POD [56] or singular value decomposition (SVD) [3]. After the POD modes are identified for the small system, they are projected back onto the original high-dimensional space. Although these methods allow for rapidly solving the POD problem for large-scale systems, they have two main drawbacks: the resulting modes are not sparse and the algorithm does not converge if the singular values do not decay rapidly [27].

Adaptive methods benefit from the sparse representation of the data already in the stage of production. For example, when generating the data numerically using finite element solvers like `FEniCS` [4, 5] or wavelet adaptive solvers such as `WABBIT` [19, 48, 49]. In contrast to randomized methods, the representation of the data is seen from an infinite-dimensional perspective, where each snapshot corresponds to a function over an infinite dimensional Hilbert space. This fact has both advantages and drawbacks: On the one hand, adaptation techniques allow direct relations to the "truth", i.e., the exact solution of the PDE or the underlying physical system. For example, [1, 2] obtain approximations of the exact solution of the PDE within a given tolerance by using dual wavelet expansions of the PDE systems' continuous residual together

with greedy reduced basis methods. Furthermore, adaptation is advantageous over equidistant grids because it distributes computational efforts to places where relevant information is located. On the other hand, adaptive methods require specifically tailored techniques for storing and processing the data. When ignoring gains in precision, adaptive methods are thus more complex and computationally demanding compared to non-adaptive equivalents for the same amount of processed data. Transferred to the POD problem this implies that the POD basis cannot be formed by means of a simple SVD or the *direct method* (as coined in [47]), since no snapshot matrix or correlations between (a theoretically infinite number of) space points can be computed. Alongside [20, 24, 25, 51], we therefore use the so-called *method of snapshots or strobes* [47], which only relies on correlations between a finite number of snapshots, computed with a simple inner product. Nevertheless, snapshot and direct methods are in principle equivalent in solving the POD problem [31, 54].

Although most authors use the method of snapshots to obtain POD modes, their precise implementations differ. In early works, Ullmann et al. [51] used two different approaches to compute the POD basis. In the first approach, they represent all snapshots and the resulting POD basis on a common finite element (FE) grid/space, which is somehow similar to the approach in [20]. This has the advantage that the snapshots and modes can be interpreted as Euclidean vectors of the same size with a single weighted inner product. In the second approach, the authors build common FE-spaces for pairs of snapshots to compute the correlation matrix and define the POD modes implicitly as a linear combination of the snapshots on their original FE-space. A variation of the first approach has been used by [25] to build reduced order models for the incompressible Navier-Stokes equations. Most recent advances [24] go one step further to avoid any common finite element space. Therefore, the authors of [24] reformulate the inner product between two snapshots in terms of the underlying finite element basis. Based on the work of [37], the authors in [24] are able to compute inner products between arbitrary FE- grids, when including cut finite elements. Instead of lifting the grids to a common reference grid, the overlap between different finite elements has to be calculated.

In [23] the POD is compared with orthogonal wavelets for extracting coherent vortices out of turbulent flows considering direct numerical simulation data of 2D drift-wave turbulence. Issues of computational complexity and memory requirements for storing the POD modes were discussed.

The present paper approaches the method of snapshots from a wavelet point of view as it entails various desirable conceptual features that are not shared by the other adaptive methods outlined above. It shares some basic ideas with [9, 53], although the results presented there are only in one space dimension. The approach presented here is integrated as a post-processing routine into an open source software called `WABBIT` ((W)avelet (A)daptive (B)lock (B)ased Solver for (I)nteractions with (T)urbulence), which is freely available at [49]. The basic adaptation technique grounds on the idea of [14] and the recent works [19, 48], in which a block-structured grid is refined or coarsened using biorthogonal wavelets. The framework is implemented using the MPI Library to exploit parallel computing architectures. The novelty of the present approach lies in the combination of POD and wavelet adaptation and the ability to balance wavelet compression and POD truncation errors. Here,

the hierarchical nature of wavelets enables efficient refinement and coarsening due to the scaling relations of wavelets.

Moreover, the underlying multiresolution analysis (MRA) [36] allows to approximate any finite energy function with a given precision. Nonlinear approximation [13] using thresholding of the wavelet coefficients yields then sparse and efficient representations to reduce memory and CPU time requirements.

Furthermore, the locally uniform Cartesian grid structure of each block enables us to apply the method to images or other 2D/3D equidistant fields in space. This allows us to directly compare our results with randomized methods, which need several passes over the data in case the singular values of the snapshot matrix decay slowly [27]. Our results show indeed that we can avoid this by sparse adaptation of the data. This serves as motivation to propose a strategy of how to balance a priori rank truncation and wavelet compression error.

The present paper is organized as follows. In Section 2, we introduce the wavelet adaptive framework with a detailed description of our implementation (Section 2.1) and a brief summary of the applied wavelet adaptation scheme (Section 2.2). The wavelet adaptive POD is outlined in Section 3.3, followed by an error estimation in Section 3.4, where we provide a strategy for balancing adaptation and truncation errors. Next, we examine and discuss the behavior of our algorithm with the help of three different examples in Section 4. In the first example a 2D synthetic test case is provided, where the influence of various parameters is studied and the results are compared to the randomized SVD. Thereafter, Section 4.2 presents two data sets in the context of computational fluid dynamics: data from a direct numerical simulation of a 2D wake flow past a cylinder (Section 4.2.1) and a 3D block-based adaptive simulation of a bumblebee in forward flight (see Section 4.2.4). Finally, we summarize our results and provide a short outlook for future research (Section 5).

2 Numerical methods and implementation

In the following, we describe the numerical methods used in the wPOD algorithm and give detailed insight into its implementation, when handling multiple block-based adaptive grids. The basic wavelet adaptation technique used for our algorithm has been already discussed in [19, 48]. We hence limit the presentation here to changes specific to our algorithm. In the interest of readability, we will assume two-dimensional data, thus all quantities with a subscript α are indexed over $\alpha = 1, 2$.

2.1 Block-structured grid and implementation

Multiresolution representations require a dedicated data structure. Here, spatial data is divided into a set of nested blocks, which are organized in a tree. We use a collection of trees, which we call forest, in order to store multiple snapshots together with their designated tree simultaneously.

Computational grid With each tree in the forest $\mathcal{F}$, we associate a multiresolution grid Ω_i on a rectangular domain $\mathbb{D} \coloneqq [0, L_1] \times [0, L_2] \subset \mathbb{R}^2_+$. As illustrated in Fig. 1 for the 2D case, the grid Ω_i is composed of blocks

$$\mathcal{B}_p^j = \{\boldsymbol{x} = \boldsymbol{x}_p + \left(k_1 \Delta x_1^j, k_2 \Delta x_2^j\right) \mid k_1 = 0, \ldots, B_1 - 1\,, k_2 = 0, \ldots, B_2 - 1\} \quad (2)$$

of equal size $B_1 \times B_2$. The subdivision of the grid is controlled by the tree level $j = J_{\min}, J_{\min}+1, \ldots, J_{\max}$. With increasing tree level $j \to j+1$ the lattice spacing of the block is divided by two, i.e., $\Delta x_\alpha^j = 2^{-j} L_\alpha/(B_\alpha - 1)$. Here, $L_\alpha > 0$ is the size of the computational domain. Gradedness of the resulting grid is enforced by allowing adjacent blocks to differ only by one tree level. The level and location of a block are encoded in a unique tree code as indicated in Fig. 1. The computational grid Ω_i is the union of all blocks in the tree,

$$\Omega_i = \bigcup_{j=J_{\min}}^{J_{\max}} \bigcup_{p \in \Lambda_i^j} \mathcal{B}_p^j, \quad (3)$$

where Λ_i^j is the set of all block IDs at a given tree level j and tree i. The block IDs $p \in \Lambda_i^j$ are called *tree code*. The synchronization between blocks is done by using an overlapping layer covering g points (light gray area in Fig. 1). These additional points are called *ghost nodes* and they form the *ghost node layer*. The size of the ghost node layer is adjusted according to the spatial support of the chosen wavelet, here $g = 6$ for CDF 4/4 wavelets. If neighboring blocks are not on the same tree level, we interpolate or decimate the corresponding nodes when synchronizing the ghost nodes. Redundant nodes on the boundary of adjacent blocks always belong to the block with lower tree level, i.e., coarser lattice spacing. The definition of the grid as well as the reasons for choosing this particular design are detailed in appendix B of [19].

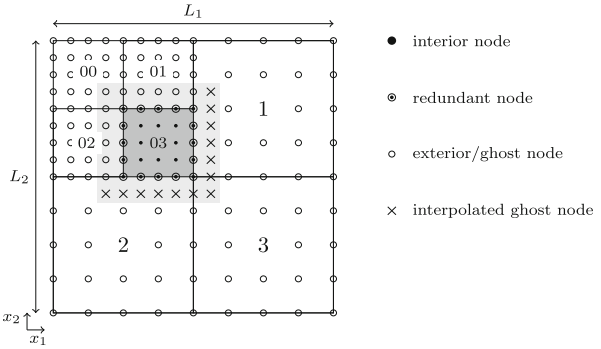


Fig. 1 A 2D grid consisting of seven blocks. A single block (dark gray) consists of an odd number of interior and redundant grid nodes $B_\alpha = 5$ in direction x_α, where $\alpha = 1, 2$. The block includes a ghost node layer (light gray) which is synchronized with neighboring blocks. The size of this layer depends on the support of the chosen wavelet. Ghost nodes are interpolated, if the levels of the neighboring blocks differ

Forest composed of multiple trees For the administration of all blocks in the forest, we use a *multi-tree structure* illustrated in Fig. 2. Here each tree $\mathcal{T}_i$ holds a collection of blocks $\mathcal{B}_p$ and block values $u^{(p)}$

$$\mathcal{T}_i = \left\{ (\mathcal{B}_p, u^{(p)}(\mathbf{x})) \mid \mathbf{x} \in \mathcal{B}_p,\, p \in \Lambda_i^j,\, j = J_{\min}, \ldots, J_{\max} \right\}, \tag{4}$$

where a block is a leaf at the end of a branch, which can be uniquely identified by a tree code $p \in \Lambda_i^j$ and a tree ID i. The tree ID identifies the corresponding grid Ω_i and the tree code determines the topology, such as block position $\mathbf{x}_p$ and lattice spacing Δx_i. In the following, the collection of trees is called *forest*

$$\mathcal{F} = \{\mathcal{T}_i \mid i = 1, \ldots, N_{\text{tree}}\}. \tag{5}$$

In Section 3.3 we will use the forest to hold multiple spatial fields as time or parameter samples of the wPOD algorithm.

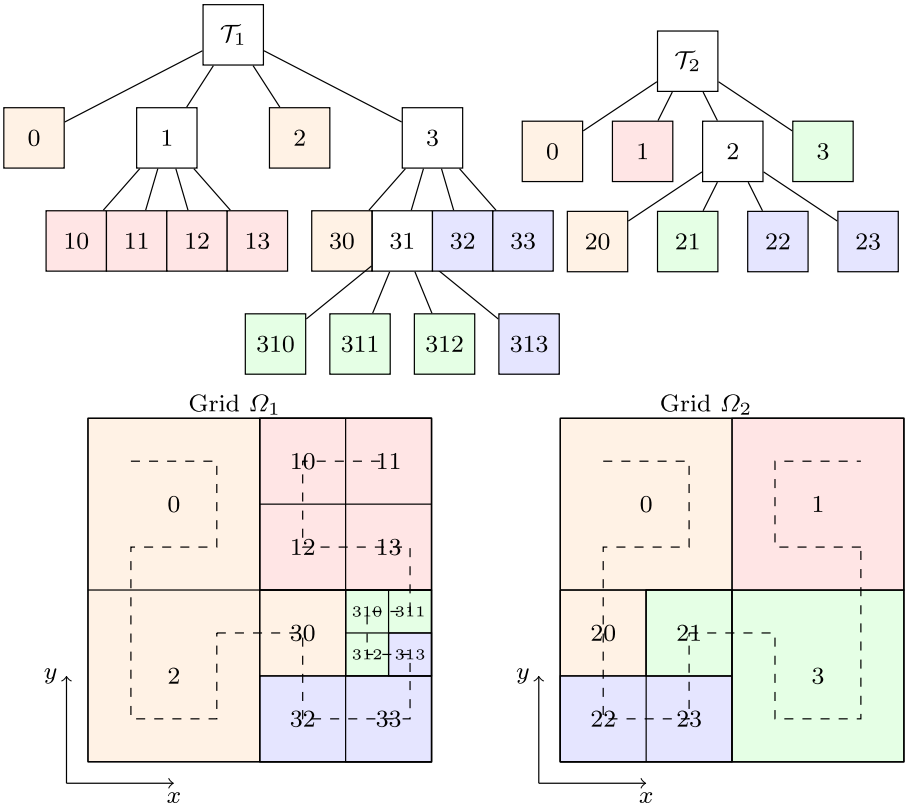


Fig. 2 A forest of two trees and their associated grids Ω_i. Top part visualizes the tree structure for tree $i = 1, 2$. Colored leafs at the end of a branch correspond to blocks on the grid as shown in bottom part. The block color encodes the processor that holds the block. The blocks are distributed among processors using space filling curves (dashed lines)

Light and heavy data storage To distinguish between administrative and physical information, we separate our data structures into light and heavy data.

The *light data* (`lgt_n`, `lgt_active`, `lgt_block`) is the minimal information necessary to organize the topology of our grids. From the light data we can determine neighbor relations between the blocks and keep track of the processors holding the block. It is therefore synchronized among all processing units. Figure 19 in Appendix C illustrates an example of light data in the case of the tree structure shown in Fig. 2. All light data are stored in the `lgt_block` array. Each row holds the necessary information of one block, such as tree ID, grid refinement level, tree code and *refinement status/coarsening indicator* (see Appendix A). The row index is called light-ID (`lgt_id`). It is ordered lexicographically in the process-ID: $\texttt{lgt\_id} = (i_{\text{proc}} - 1)N_{\text{blocks}} + j$, with $j = 1, \ldots, N_{\text{blocks}}$. In this way, we relate the position of the block with the process-ID i_{proc}. During the execution of the algorithm, blocks can be created or deleted. To avoid expensive memory allocation, we set the block inactive by marking the rows in `lgt_block` with -1. From `lgt_block` we compute active block lists (`lgt_active`) for each tree. In this way, we are able to manipulate blocks on different grids efficiently, by only looping over the active lists of a given tree specified by its `tree_id` (compare with Fig. 19).

Besides the light administrative structure, we have to store large data fields with the physical information of the state vector quantity $u(\mathbf{x})$ and all neighbor relations of the blocks. These data are called *heavy data* and they are equally distributed among the processors using the index of the space filling curve (Hilbert/Z-curve) [57]. The index can be easily calculated from the tree code. In Fig. 2 the *Hilbert-curve* is visualized with the dashed line passing through the lattice and the processors-IDs are encoded with the color of the block.

Using *space filling curves* has two major advantages for our algorithm. Firstly, due to locality properties of the space filling curve the communication between adjacent blocks which do not share the same processors is kept at a minimum. Secondly, the uniqueness of the curve ensures that trees with the same tree structure (i.e., same grid Ω_i) have identical processor distribution. This is advantageous when performing pointwise operations between trees.

2.2 Block-based wavelet adaptation

Our adaptation algorithm is block-based, meaning that blocks are coarsened or refined depending on the local regularity of the sampled function. Blocks can be coarsened by leaving out every second point and merged with their four or eight neighboring blocks in two and three space dimensions, respectively. Blocks can be refined via interpolation at all dyadic points and subdividing them into four (2D) or respectively eight (3D) new blocks. The procedure of recursive dyadic refinement is known as interpolatory subdivision and was first introduced by Deslauriers and Dubuc in [11, 12]. It was later shown by Donoho in [15] that the resulting interpolation or scaling function can be used for constructing a multiresolution analysis. Our approach is based on the discrete point value multiresolution framework of Harten [28–30], which uses interpolating scaling functions of Deslauriers and Dubuc for

discrete data representation. However, to obtain a better scale and frequency separation between the different scales we use lifted wavelets. Hence a low pass filter is applied before downsampling the block. The filter coefficients of the coarsening and refinement procedure (see Table 2) define the underlying wavelet scheme. The used lifted Deslauriers-Dubuc wavelets correspond to biorthogonal Cohen-Daubechies-Feauveau (CDF) wavelets [10]. We refer the reader to Appendix A for a detailed explanation of the adaptation algorithm. In the following the adaptation algorithm will be denoted by $u^\epsilon = \texttt{adapt}(u, \epsilon, J_{\min}, J_{\max})$. Here, the input and output functions $u, u^\epsilon \in L^2(\mathbb{D})$ are continuous and given in the block-based format outlined above. The adaptation algorithm is limited to a minimal and maximal refinement level $J_{\min}, J_{\max}$. Depending on the wavelet threshold ϵ blocks are coarsened, if the absolute difference between a block sample and its interpolated values from a coarser level is smaller than a level-dependent threshold that scales with ϵ (for details we refer to Appendix A.3). The difference is referred to as a *detail coefficient* of the wavelet basis. Therefore, the relative error between the input and output, i.e., the wavelet compression, can be bounded by the wavelet threshold:

$$\frac{\|u - u^\epsilon\|}{\|u\|} \leq \epsilon. \tag{6}$$

Here, the normalization of the wavelet basis determines the norm used in Eq. (6). For this study we use biorthogonal wavelets, which are normalized in L^2. In the following we will denote all fields with an upper index ϵ, where details have been filtered with the adaptation algorithm and can be thus expressed in a wavelet series with filtered coefficients. Figure 3 shows an example of such a field. Here one snapshot of a direct numerical simulation is block-decomposed and filtered for different threshold values ϵ.

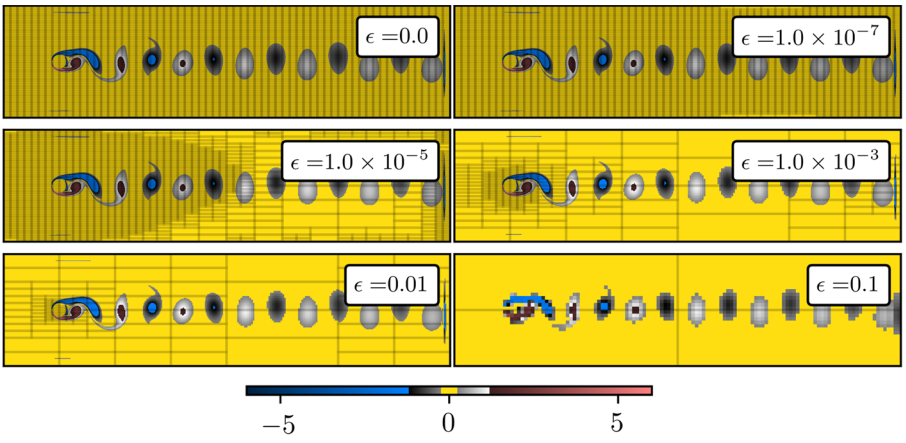


Fig. 3 Block-based adaptation of a flow past a cylinder for different thresholds ϵ. Shown is the vorticity field $\omega^\epsilon = \partial_x v_y^\epsilon - \partial_y v_x^\epsilon$, which is computed after having applied the wavelet adaptation to the full state vector $\mathbf{u}^\epsilon = (v_x^\epsilon, v_y^\epsilon, p^\epsilon)$. Each block represents $B_1 \times B_2 = 65 \times 17$ points. More details can be found in Section 4.2.1

In Fig. 3 the block boundaries are visualized by a rectangular box. All blocks have the same resolution $B_1 \times B_2 = 65 \times 17$. Smaller blocks correspond to higher tree levels, i.e., smaller lattice spacing. With increasing ϵ the number of blocks decreases, since less details are above the threshold.

3 Algorithm

In the following, we introduce the proper orthogonal decomposition (POD) as the basis of our algorithm in Section 3.1. This section aims at providing a broad overview on the state of the art techniques to compute PODs for large data sets in fluid dynamics, namely the snapshot POD and the randomized singular value decomposition (rSVD). For a more detailed explanation we refer the reader to [54] for the POD and [27] for the rSVD. In Section 3.3 we generalize the snapshot POD using wavelet adapted snapshot fields. We also propose a strategy for balancing wavelet thresholding and POD truncation error in Section 3.4.

3.1 The snapshot POD and randomized SVD

Given are samples of a continuous vector-valued L^2-function $\mathbf{u}(\boldsymbol{x}, \mu) \in \mathbb{R}^K$, $K > 0$, $\mathbf{x} \in \mathbb{D}$ sampled in some parameter interval $\{\mu_i\}_{i=1}^{N_\mathrm{s}}$. The samples are called snapshots and are in the following indexed by $\mathbf{u}_i = \mathbf{u}(\boldsymbol{x}, \mu_i)$. Our algorithm solves the following POD optimization problem

$$\min_{\{\boldsymbol{\Psi}_k\}} \sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i - \sum_{k=1}^{r} \langle \mathbf{u}_i, \boldsymbol{\Psi}_k \rangle \boldsymbol{\Psi}_k \|^2, \quad \text{such that} \quad \langle \boldsymbol{\Psi}_k, \boldsymbol{\Psi}_l \rangle = \delta_{kl}\,, \tag{7}$$

with the L^2 inner product $\langle \cdot, \cdot \rangle$ and associated norm $\| \cdot \| = \sqrt{\langle \cdot, \cdot \rangle}$. We solve it with the *method of snapshots or strobes* [47] since for data where the spatial resolution is much larger than the number of snapshots N_s, Eq. (7) is reduced to a small eigenvalue problem of size $N_\mathrm{s} \times N_\mathrm{s} \sim \mathcal{O}(100)$

$$\mathbf{C}\,\mathbf{v}_k = \lambda_k \mathbf{v}_k \quad \text{for} \quad k = 1, \ldots, r\,, \tag{8}$$

for the correlation matrix

$$\mathbf{C}_{ij} = \frac{1}{N_\mathrm{s} V} \langle \mathbf{u}_i, \mathbf{u}_j \rangle\,, \tag{9}$$

together with the relation:

$$\boldsymbol{\Psi}_k = \frac{1}{\sqrt{\lambda_k N_\mathrm{s}}} \sum_{i=1}^{N_\mathrm{s}} (\mathbf{v}_k)_i \mathbf{u}_i \quad k = 1, \ldots, r\,. \tag{10}$$

The method of snapshots is strongly connected to the *singular value decomposition* (SVD) [54], because left singular vectors correspond to the solution of Eq. (7) and right singular vectors to $\mathbf{v}_k$, when assuming Euclidean space [54]. Respectively, the eigenvalues $\lambda_k = \sigma_k^2 \geq 0$ are squares of the singular values. Furthermore, it is known from the Eckart-Young-Mirsky theorem [16] that the resulting approximation

error in the Frobenius norm, when truncating after the rth mode, is given by the sum $\sum_{k=r+1}^{N_s} \sigma_k^2$ of the remaining singular values.

However, caution must be taken when using this method instead of the SVD, because the condition number $\kappa(\mathbf{U}) := \sigma_{\max}(\mathbf{U})/\sigma_{\min}(\mathbf{U})$ of the associated snapshot matrix $\mathbf{U}$ is squared: $\kappa(\mathbf{U}^T\mathbf{U}) = \kappa(\mathbf{U})^2$. This can lead to inaccuracy of POD modes with small singular values (see the famous example of Lächli [33]). Nevertheless, one is often willing to accept this potential error in favor of a smaller problem size.

Another way of reducing the problem size, without squaring the condition number, is using a *randomized SVD* algorithm, which is outlined in [27]. Here, an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{M\times q}$, $q \ll N_s$ is formed which approximates the column space of $\mathbf{U} \approx \mathbf{QB}$. With its help, a singular value decomposition of the small $q \times N_s$ matrix $\mathbf{B} = \mathbf{Q}^T\mathbf{U}$ is computed and the left singular vectors $\boldsymbol{\Psi}_k^{\mathbf{B}}$ are projected back onto the full space via: $\boldsymbol{\Psi}_k = \mathbf{Q}\boldsymbol{\Psi}_k^{\mathbf{B}}$. Usually the orthogonal matrix is formed by a QR decomposition taking q random samples of the column space of $\mathbf{U}$. For a target number of r modes one usually oversamples $q = r + n$ by taking $n = 5$ or $n = 10$ additional random samples [27]. However, if the singular values decay slowly, $\mathbf{Q}$ may not represent $\mathbf{U}$ well enough and costly tricks, like *Power Iterations* using additional passes over the data, have to be applied [27]. Moreover, as pointed out in [27] for very large matrices $\mathbf{U}$ the data cannot be loaded into fast memory and therefore the transfer from the slow memory typically dominates the arithmetic. In contrast, the wPOD algorithm presented in the next section seeks to avoid these problems by reducing the relevant information of each single snapshot to fit it into the fast memory.

3.2 The weighted inner product and pointwise tree operations

In the wPOD algorithm, we follow the same approach as in the method of snapshots. explained in the previous section (Section 3.1). However, we use the sparsity of our wavelet block-based data representation to allow for an memory efficient computation of the POD basis. In contrast to the representation in terms of the snapshot matrix, as used by the SVD, our data is represented in terms of a forest $\mathcal{F}$ or a collection of trees (see Section 2.1). Here each snapshot $\mathbf{u}_i(\mathbf{x}) := \mathbf{u}(\mathbf{x}, \mu_i)$ is associated with a tree $\mathcal{T}_i$ on a hierarchical structured multiresolution grid Ω_i, $i = 1, \ldots, N_s$. The leaves of the tree correspond to blocks, where each block p stores coefficients $\{u_p^j[k_1, k_2]\}_{k_1,k_2}$ of the underlying basis $\{\phi_{k_1,k_2}^j\}_{k_1,k_2}$ at tree level j. The interpolating basis allows to represent the data in a continuous form, when summing over all blocks $p \in \Lambda_i^j$ of each tree level j:

$$\mathbf{u}_i(\mathbf{x}) = \sum_{j=1}^{J_{\max}} \sum_{p\in\Lambda_i^j} \mathbf{u}_i^{(p)}(\mathbf{x}) \quad \text{for} \quad i = 1, \ldots, N_s \tag{11}$$

We can thus introduce the *snapshot set*:

$$\mathcal{U} = \{\mathbf{u}(\mathbf{x}, \mu_1), \ldots, \mathbf{u}(\mathbf{x}, \mu_{N_s}) \mid \mathbf{x} \in \mathbb{D}\}, \tag{12}$$

as the continuous counterpart of the snapshot matrix $\mathbf{U}$. As mentioned earlier, our algorithm is capable of handling 2D and 3D data fields on a rectangular domain

$\mathbb{D} \subset \mathbb{R}^d$, $d \in \{2, 3\}$. Note that with the new data representation in terms of functions in the L^2 Hilbert space, the inner product in the POD formulation has changed to

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle \coloneqq \int_{\mathbb{D}} \mathbf{u}_i^T(\mathbf{x}) \mathbf{u}_j(\mathbf{x}) \, \mathrm{d}\mathbf{x} \ . \tag{13}$$

However, for inner products or any pointwise operation (+,-) between snapshots $\mathbf{u}_i, \mathbf{u}_j$, represented on locally different grids Ω_i, Ω_j, it is required that both coefficient vectors $\underline{\mathbf{u}}_i, \underline{\mathbf{u}}_j$ are of same length, i.e., expressed in the same basis. In contrast to the discussed FEM methods [25, 51], this can be achieved very efficiently, because the hierarchical grid definition allows to merge two grids by the union $\Omega_{ij} = \Omega_i \cup \Omega_j$ for the two snapshots involved. Figure 4 visualizes the grid merging procedure. In Fig. 4a, the initial grids Ω_i and Ω_j are displayed together with their processor distribution. In this example both trees have a fundamentally different tree structure and processor distribution. In areas where Ω_i has large details, Ω_j does not and vice versa. The aim of the union of both grids is to merge them, such that no detail of both trees gets lost. This implies that merging both trees only involves refinement operations, which are cheap when using wavelet up-sampling.

As explained in Section 2.1 trees with identical tree structure have identical processor distribution because of our load balancing strategy by space filling curves. The `hvy-data`, i.e., grid quantities of Ω_i and Ω_j are therefore on the same processor (see Fig. 4b) after unification.

The unification enables us to calculate the inner product in Eq. (13) as a weighted inner product

$$\langle \mathbf{u}_i, \mathbf{u}_j \rangle = \langle \underline{\mathbf{u}}_i, \underline{\mathbf{u}}_j \rangle_{\boldsymbol{M}} = \underline{\mathbf{u}}_i^T \boldsymbol{M} \underline{\mathbf{u}}_j, \tag{14}$$

where $\underline{\mathbf{u}}_i$ represent the vectorized entries of tree i and $\boldsymbol{M}$ is a positive definite and symmetric matrix (explicitly given in Appendix A.4). In FEM literature this matrix is often called mass matrix and has been used by [25, 51] in a similar approach. With Eq. (14) we are able to calculate the inner product and the associated norm of

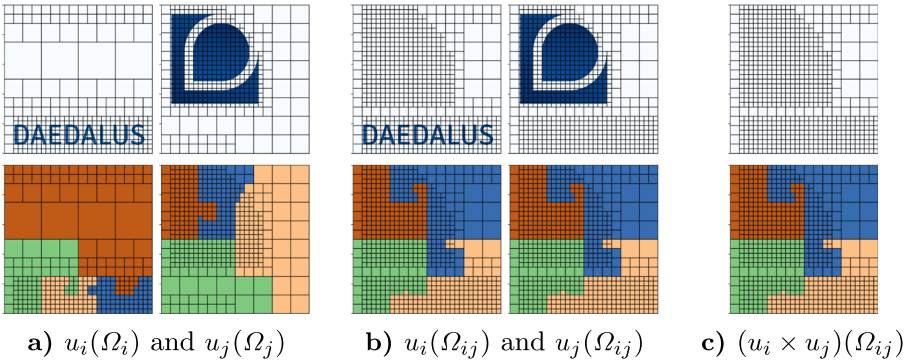


Fig. 4 Visualization of pointwise operations on multiresolution grids Ω_i and Ω_j. On the top row the scalar fields u_i and u_j are shown, where blue colors represent 1 and white colors 0. Below each field the corresponding processor distribution grid is displayed. Each color of the processing grid represents one of the four processing units. Initial trees (Fig. 4a), unified tree structures (Fig. 4b) and the processed field (Fig. 4c)

our block-based multiresolution fields. Replacing the L^2 inner products by weighted inner products therefore generalizes the POD minimization problem in Eq. (7) to multiresolution fields. With this intermediate result we are able to go one step further to combine wavelet adaptation and POD truncation.

3.3 The wPOD algorithm

The wPOD algorithm proceeds in the following steps:

1. **Read and Coarsen Data** $\mathbf{u}_i^\epsilon \leftarrow \texttt{adapt}(\mathbf{u}_i, \epsilon, J_{\min}, J_{\max})$
   In the first step of the algorithm we read all block-decomposed snapshots in $\mathcal{U}$ and coarsen them for a given threshold ϵ using the wavelet adaptation scheme, if the input fields are not already adapted. The adapted fields are denoted by $\mathbf{u}^\epsilon$. This part of the algorithm is essential because it allows to keep only the most relevant information (see Section 2.2) of the input data using wavelet compression and therefore makes handling of large data feasible (see Section 4).
2. **Computation of Correlation Matrix** $(\mathbf{C})_{ij}^\epsilon = \frac{1}{N_\mathrm{s} V}\langle \mathbf{u}_i^\epsilon, \mathbf{u}_j^\epsilon \rangle$
   The main computational effort of the algorithm is the construction of all elements of the correlation matrix $\mathbf{C}^\epsilon \in \mathbb{R}^{N_\mathrm{s}\times N_\mathrm{s}}$, for which the inner product of locally different resolved snapshots needs to be computed. For any pairwise operation $(+, -, \langle\cdot,\cdot\rangle)$, we refine to a union of both grids as shown in Fig. 4. After the operations (+,-) the resulting field is adapted again to the predefined threshold ϵ.
3. **Solving the Eigenvalue Problem** $\mathbf{C}^\epsilon \mathbf{v}_k^\epsilon = \lambda_k^\epsilon \mathbf{v}_k^\epsilon$
   After the correlation matrix $\mathbf{C}^\epsilon$ is constructed, we diagonalize it with Jacobi's method for real symmetric matrices `DSYEV` implemented in `LAPACK` [6]. As described in [6] sec. 8, the chosen method computes all the eigenvalues and eigenvectors close to machine precision. We therefore neglect errors made during the diagonalization. In contrast to the construction of $\mathbf{C}^\epsilon$, the computational effort needed for diagonalization is relatively small.
4. **Construction of POD Modes** $\boldsymbol{\Psi}_k^\epsilon = \frac{1}{\sqrt{\lambda_j^\epsilon N_\mathrm{s}}} \sum_{i=1}^{N_\mathrm{s}} (\mathbf{v}_k^\epsilon)_i \mathbf{u}_i^\epsilon$
   The elements of the correlation matrix are the inner products between two snapshots, i.e., the ith row/column contains the coefficients of $\mathbf{u}_i$ represented by a linear combination of all snapshots in $\mathcal{U}$. Diagonalizing $\mathbf{C}^\epsilon$ means finding a basis of coefficient vectors $\mathbf{v}_k^\epsilon \in \mathbb{R}^{N_\mathrm{s}}$ which generate an optimal representation of $\mathcal{U}$. The representation in terms of orthonormal modes $\{\boldsymbol{\Psi}_k^\epsilon\}$ is computed according to Eq. (10). The summation in Eq. (10) proceeds in multiple steps. In the first step we copy $\boldsymbol{\Psi}_k^\epsilon \leftarrow (\mathbf{v}_k^\epsilon)_1 \mathbf{u}_1^\epsilon$, after which we iteratively sum up $\boldsymbol{\Psi}_k^\epsilon \leftarrow \boldsymbol{\Psi}_k^\epsilon + (\mathbf{v}_k^\epsilon)_i \mathbf{u}_i^\epsilon$ for $i = 2, \ldots, N_\mathrm{s}$ in the second step and divide by the normalization factor.
5. **Computation of POD Modes Coefficients** $a_{ki}^\epsilon = \frac{1}{V}\langle \boldsymbol{\Psi}_k^\epsilon, \mathbf{u}_i^\epsilon \rangle$
   The computation of the modes coefficients involves again a computation of the inner product between the orthonormal modes $\boldsymbol{\Psi}_k^\epsilon$ and the snapshots $\mathbf{u}_i^\epsilon$. In most cases, this step needs less evaluations of the scalar product since the number of modes r should be small, $r \ll N_\mathrm{s}$.

In summary, our algorithm generates sparse modes $\boldsymbol{\Psi}_k^\epsilon$, $k = 1, \ldots, r$, with amplitudes a_{ki}^ϵ to approximate any of the snapshots $\mathbf{u}_i \in \mathcal{U}$ in terms of a linear subspace

$$\mathbf{u}_i(\mathbf{x}) \approx \tilde{\mathbf{u}}_i^\epsilon(\mathbf{x}) = \sum_{k=1}^{r} a_{ki}^\epsilon \boldsymbol{\Psi}_k^\epsilon(\mathbf{x}) \quad \text{for} \quad i = 1, \ldots, N_\mathrm{s}\,. \tag{15}$$

In this notation, the upper index ϵ denotes the quantities, which are indirectly affected by the wavelet threshold (e.g., a_{ki}^ϵ, C_{ij}^ϵ) or directly expressed as a truncated wavelet series (e.g., $\tilde{\mathbf{u}}_i^\epsilon(\mathbf{x})$, $\boldsymbol{\Psi}_k^\epsilon(\mathbf{x})$). Furthermore, $\tilde{\mathbf{u}}$ denotes the truncation after the rth mode.

A comparison of the CPU time required in the individual steps of the wPOD algorithm is shown in Fig. 20 (Appendix C).

3.4 Error estimation

Here, we discuss the dependency of the approximation error on the wavelet threshold ϵ and the truncation rank r. Additionally, we explain how to choose both values in order to obtain a given accuracy.

Therefore, we provide an error estimate of the approximation $\tilde{\mathbf{u}}_i^\epsilon$ in Eq. (15). The approximation projects our data $\mathcal{U} = \{\mathbf{u}_1, \ldots, \mathbf{u}_{N_\mathrm{s}}\}$ onto a linear subspace spanned by a set of r orthonormal modes $\{\boldsymbol{\Psi}_k^\epsilon\}_{k=1,\ldots,r}$. The sparsity of the modes is determined by the wavelet threshold ϵ (see Section 2.2) and the dimension of the subspace r shall be much smaller than the number of snapshots: $r \ll N_\mathrm{s}$. For given r, ϵ we define the relative error of our approximation in the L^2-norm,

$$\mathcal{E}_\mathrm{wPOD}(r, \epsilon) := \frac{\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i - \tilde{\mathbf{u}}_i^\epsilon\|^2}{\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i\|^2} \tag{16}$$

which can be split into two contributions, i.e., compression and truncation errors. We thus have,

$$\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i(\mathbf{x}) - \tilde{\mathbf{u}}_i^\epsilon(\mathbf{x})\|^2 = \sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i - \mathbf{u}_i^\epsilon + \mathbf{u}_i^\epsilon - \tilde{\mathbf{u}}_i^\epsilon\|^2 \tag{17}$$

$$\leq \underbrace{\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i - \mathbf{u}_i^\epsilon\|^2}_{\text{compression error}} + \underbrace{\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i^\epsilon - \tilde{\mathbf{u}}_i^\epsilon\|^2}_{\text{POD truncation error}}\,. \tag{18}$$

Hereby, the relative error arising from thresholding details is defined by

$$\mathcal{E}_\mathrm{wavelet}(\epsilon) := \frac{\|\mathbf{u}_i - \mathbf{u}_i^\epsilon\|}{\|\mathbf{u}_i\|} \tag{19}$$

and the relative error due to the truncation after the rth POD mode in Eq. (15) is:

$$\mathcal{E}_\mathrm{POD}(\epsilon, r) := \frac{\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i^\epsilon - \tilde{\mathbf{u}}_i^\epsilon\|^2}{\sum_{i=1}^{N_\mathrm{s}} \|\mathbf{u}_i^\epsilon\|^2} = \frac{\sum_{k=r+1}^{N_\mathrm{s}} \lambda_k^\epsilon}{\sum_{k=1}^{N_\mathrm{s}} \lambda_k^\epsilon} \tag{20}$$

Further details can be found in Section 3.3. Using Eqs. (6) and (18)–(20), one obtains for the total relative error of the wPOD (see Appendix B):

$$\mathcal{E}_{\mathrm{wPOD}}(\epsilon, r) \leq \mathcal{E}_{\mathrm{POD}}(0, r) + \mathcal{M}_r \epsilon + \epsilon^2 \approx \mathcal{E}_{\mathrm{POD}}(0, r) + \epsilon^2 , \tag{21}$$

$$\text{where} \qquad \mathcal{M}_r = \frac{\sum_{k=r+1}^{N_\mathrm{s}} l_k}{\sum_{k=1}^{N_\mathrm{s}} \lambda_k} , \tag{22}$$

when assuming perturbed eigenvalues $\lambda_k^\epsilon = \lambda_k + l_k \epsilon$, with perturbation $l_k \in \mathbb{R}$. An error bound for the perturbation of the eigenvalues is given for a simplified correlation matrix in [9]. $\mathcal{M}_r$ is often small, in which case it can be neglected.

This is further discussed in the numerical example section. Note that in the limit $\epsilon \to 0$ the wPOD error yields exactly the POD error. In the limit $r \to \infty$ the POD error vanishes and we are left with the wavelet compression error. These limits are visualized for our numerical studies in Figs. 13 and 16.

In most of the applications the wavelet threshold ϵ will be chosen according to the available memory of the hardware. If memory limitations are not an issue, it is advantageous to balance the wavelet and POD truncation error for better efficiency. Assuming that the approximation in Eq. (21) holds $\mathcal{E}_{\mathrm{POD}}(\epsilon, r) \approx \mathcal{E}_{\mathrm{POD}}(0, r)$ (i.e., $\mathcal{M}_r \leq \epsilon$), we can treat the two errors $\mathcal{E}_{\mathrm{wavelet}}, \mathcal{E}_{\mathrm{POD}}$ independently. For a predefined error $\mathcal{E}^*$, we first fix the compression error choosing $\epsilon^* \leq \sqrt{\mathcal{E}^*/2}$ and adjust the truncation error by $\texttt{err} = \mathcal{E}^* - (\epsilon^*)^2$. The truncation rank r^* is then chosen to compensate the additional error introduced by the wavelet compression. In the case when $\mathcal{M}_r$ is expected to be larger than ϵ, the errors cannot be balanced without having an estimate of the POD eigenvalues. Eigenvalues λ_k^ϵ, which are smaller than the compression error, are not reliable. Therefore, we recommend the conservative setting, choosing $\epsilon^* < \mathcal{E}^*$, for a first estimate.

4 Numerical results

In this section, we test the wPOD algorithm, outlined in Section 3.3, on 2D and 3D numerical data and assess its efficiency and precision. The algorithm is integrated into the open source software package `WABBIT` [49] and can be called as a post-processing routine.[1]

We provide two types of case studies: a synthetic test case (see Section 4.1) in 2D, which is used to benchmark our code. We also compare it to the randomized singular value decomposition (rSVD), outlined in Section 3.1, and case studies for 2D and 3D data obtained by numerical simulation of the incompressible Navier-Stokes equations in Section 4.2.

[1] The program has several options: `wabbit-post --POD -- nmodes=<r> --error=<err> --memory=<RAM> --adapt=<ϵ> --components=<K> --list=<ComponentList1> ····· list=<ComponentListK>` in which the number of modes r or the truncation error `err` can be specified. For the latter r is automatically chosen from the error criterion given in Section 3.3. The algorithm requires specifying the number of components K together with K lists of files which stores the snapshots of each component in a HDF5 format. Furthermore, the memory and adaptation level should be chosen in accordance with given resources.

4.1 Synthetic test case

For the synthetic test case we define a combination of dyadic structures, inspired by [38]:

$$u(x, y, t) = \sum_{k=1}^{R} a_k(t)\Psi_k(x, y)\,,$$

of $R = 15^2$ orthogonal modes $\Psi_k : [0, 30]^2 \to \mathbb{R}$ and temporal amplitudes $a_k : [0, 2\pi] \to \mathbb{R}$. The modes are smooth, two-dimensional bumps

$$\Psi_{m+15n+1}(x, y) = b(\sqrt{(x - x_m)^2 + (y - y_n)^2}) \quad \text{for} \quad n, m = 0, \ldots, 14\,. \tag{23}$$

$$b(x) = \begin{cases} \exp\left(-\frac{1}{1-x^2}\right), & x \in (-1, 1) \\ 0, & \text{otherwise} \end{cases} \tag{24}$$

placed at $(x_m, y_n) = (1+2m, 1+2n)$ in a checkerboard pattern. Note that the modes are orthogonal because of their non-overlapping support. Furthermore, we choose oscillating amplitudes:

$$a_k(t) = e^{-k/\Delta\lambda} \sin(\pi f_k t) \quad \text{for} \quad k = 1, \ldots, 15^2 \tag{25}$$

with randomly shuffled frequencies $f_k \in \{1, \ldots, 15^2\}$ and moderate decrease in magnitude: $\Delta\lambda = 3$. We choose $N_s = 2^7$ equally spaced snapshots on a $N_x \times N_y = 1024 \times 1024$ initial grid. Before starting the algorithm the initial grid has to be partitioned into blocks. This is done using a python routine available in the `WABBIT` software package [17]. In our studies we block-decompose the initial grid in three different configurations to compare the effect of different block sizes. The sizes of the blocks are $B = N_x/2^{J_{\max}} + 1 = 17, 33, 65$ with $J_{\max} = 6, 5, 4$, respectively.

4.1.1 Wavelet compression

First we examine the compression of the data with varying block size and thresholds ϵ with $10^{-15} \leq \epsilon \leq 10$. Figure 5 shows the adaptation of a single snapshot for $\epsilon = 1.0, 2.2 \times 10^{-2}, 1.0 \times 10^{-5}$. For larger ϵ the number of blocks decreases, leading to stronger compression of the data and increasing compression errors.

This behavior is quantified for varying block size B and ϵ in Fig. 6. Here we plot the relative compression error $\mathcal{E}_{\text{wavelet}}$ and compression factor $C_f \leq 1$, i.e., the

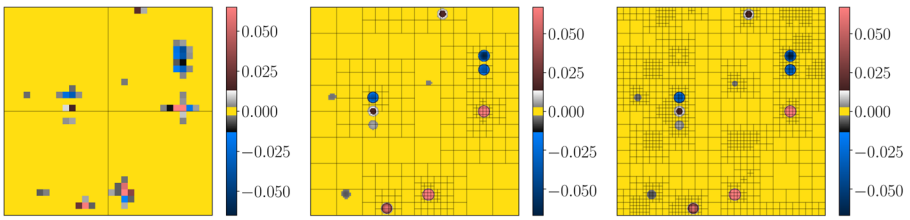


Fig. 5 Block-based adaptation of $u^\epsilon(x, y, t)$ at $t = 42\Delta t$ for $\epsilon = 1.0, 2.2 \times 10^{-2}, 1.0 \times 10^{-5}$ (from left to right) and with $B = 17$

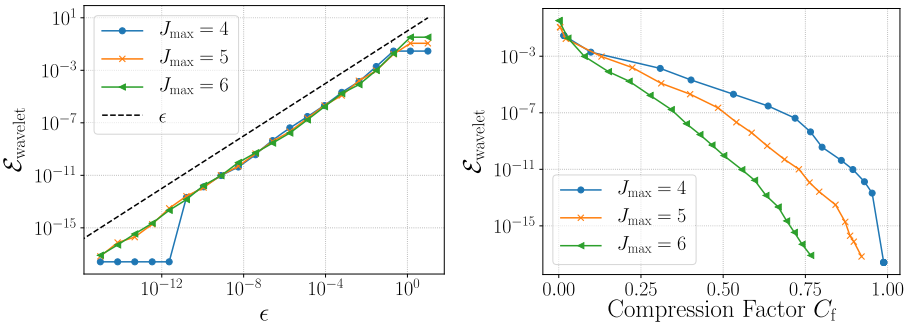


Fig. 6 Compression error $\mathcal{E}_{\text{wavelet}}$ (left) defined in Eq. (19) and compression factor C_{f} (right) of the bump test case using different block sizes $B = N_x/2^{J_{\max}} + 1$, with maximal refinement levels $J_{\max} = 4, 5, 6$. The compression factor is the fraction between the sparse and dense number of grid points / blocks

fraction between the number of blocks at a given threshold and the total number of blocks available needed for the full grid. As can be seen from Fig. 6, a higher maximal refinement level $J_{\max}$, i.e., smaller blocks, corresponds to smaller overall compression factor, while the compression error $\mathcal{E}_{\text{wavelet}}$ is approximately the same. This observation is expected, because smaller blocks enable better resolution of local structures, however increasing the data handling effort. For all the compression curves in the numerical examples in Figs. 6 and 11 we see the classical saddle shaped error curve: with rapid error decay for small $C_{\text{f}} \lesssim 0.05$ (i.e., large $\epsilon \gtrsim 10^{-2}$) until a plateau is reached with a saddle point from which it begins to decay again. Regardless of the final error of our algorithm, it is recommended to set ϵ at the onset of the plateau, since after the plateau is reached only little gain in precision is achieved.

Furthermore we emphasize that the compression error scales linear in ϵ, independent from the chosen block size, which is an important property for the error control of our algorithm. The sudden drop of the error for $J_{\max} = 4$ is due to the fact, that after $\epsilon \lesssim 10^{-11}$ the blocks are refined to the maximal level.

4.1.2 POD truncation

Next we study the impact of the wavelet compression on the computed POD modes and the overall approximation error. For $\epsilon = 10^{-5}$, Fig. 7 visualizes the first three modes and the corresponding amplitudes obtained with the wPOD algorithm. Note that the modes and amplitudes of the POD problem Eq. (7) are only unique up to an orthogonal transformation. Hence the initial input structures Ψ_k, a_k, defined in Eqs. (23) and (25), are not exactly recovered by the wPOD. However, we see that the magnitude of the amplitudes decreases and its frequency increases with increasing mode number. Moreover one observes that the computational grid is nicely adapted to the structure of the modes.

Additionally, we estimate the truncation error $\mathcal{E}_{\text{POD}}(\epsilon, r)$ and the total error $\mathcal{E}_{\text{wPOD}}(\epsilon, r)$ in the L^2-norm, as defined in Eq. (16). We compare the errors for $10^{-5} \leq \epsilon \leq 1.0$, $r \leq 30$ in Fig. 8. In these plots the impact of the wavelet adaptation, corresponding to blue lines with $\epsilon > 0$, is visualized and compared to the classical

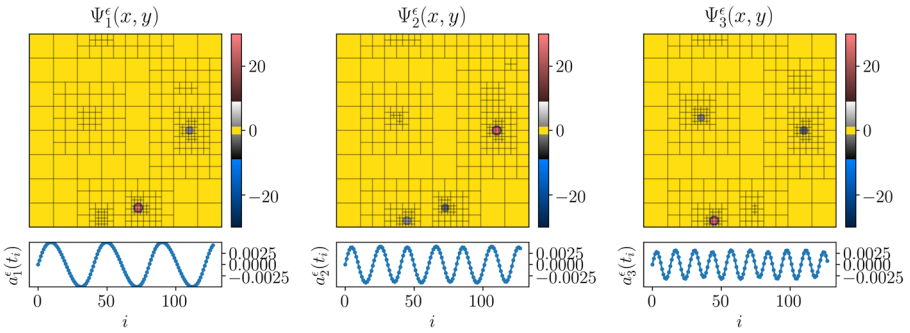


Fig. 7 First three modes Ψ_k^ϵ and their amplitudes $a_k^\epsilon(t_i)$, $k = 1, 2, 3$, for $\epsilon = 10^{-5}$. Blocks are of size $B = 17$

snapshot POD procedure corresponding to $\epsilon = 0$, which is drawn in black. The exponential decay of the eigenvalue spectra, given by the magnitude of the input modes $|a_k| \sim \exp(k/\Delta\lambda)$, is nicely recovered in the $\epsilon = 0$ case up to values $r \lesssim 60$ below

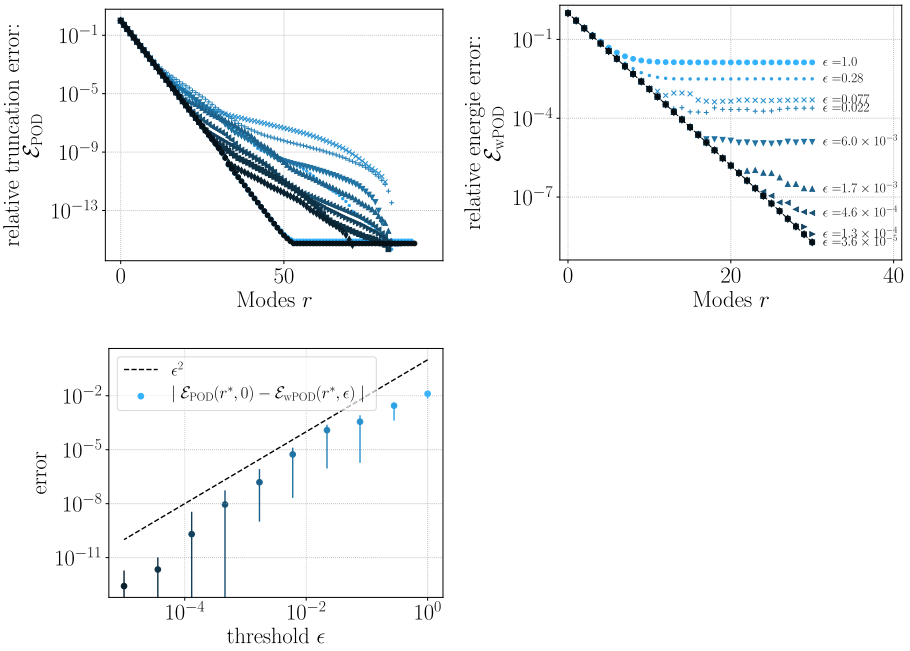


Fig. 8 Relative errors $\mathcal{E}_{\text{POD}}$ (top, left) (see definition in Eq. (20)) and $\mathcal{E}_{\text{wPOD}}$ (top, right) (see definition in Eq. (16)) as a function of the truncation rank r and the wavelet threshold ϵ. Difference between classical POD and wPOD compared to the compression error defined in Eq. (16) (bottom). The error bars indicate the minimal and maximal value of $\Delta\mathcal{E}(r,\epsilon) = |\mathcal{E}_{\text{POD}}(r,0) - \mathcal{E}_{\text{wPOD}}(r,\epsilon)|$ for all ranks $1 \le r \le 30$ and the markers the mean $\overline{\Delta\mathcal{E}}(\epsilon) = 1/30 \sum_{r=1}^{30} \Delta\mathcal{E}(r,\epsilon)$. The colors vary from bright blue at $\epsilon = 1.0$ to black at $\epsilon = 0.0$. Blocks are of size $B = 17$

machine precision. For increasing ϵ we see that the eigenvalues become increasingly distorted, as expected. However, the eigenvalue distortion does not influence the overall approximation error if epsilon is chosen with care (see Section 3.4). In fact the total approximation error converges approximately with ϵ^2 to the exact values as the difference in Fig. 8 (bottom) shows. The presented results are independent of the chosen block size B, although only $B = 17$ is used in the shown figures.

4.1.3 Comparison to randomized SVD

The overall purpose of our method is to be able to fit the snapshot data into the fast memory by tuning ϵ, in order to compute the POD without having to address the slow memory again, after the data has been compressed. This enables us to cope with large data sets. Therefore, a comparison to other methods suited for large data like the randomized SVD suggests itself. We follow the algorithm outlined in Section 3.1 taken from [27]. For a fair comparison we refrain from power iterations, which would need additional passes over the slow memory and we use $n = 5$ extra random samples as suggested in [27]. Hence for a target rank of $r^* = 30$ we take $q = 35$ random samples of our snapshot matrix $\mathbf{U} \in \mathbb{R}^{M \times N_\mathrm{s}}$, $M = 1024^2$, $N_\mathrm{s} = 2^7$ to compute an orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{M \times q}$, which approximates the column rank of $\mathbf{U}$. The computation of $\mathbf{Q}$ however is only feasible if the random samples fit in the fast memory. Therefore, the minimum amount of memory needed by the rSVD is given by $S_\mathrm{rSVD} = Mq$ in units of the floating-point arithmetic. In contrast the wPODs memory requirements in units of the floating-point arithmetic: $S_\mathrm{wPOD} = N^\mathrm{tot}_\mathrm{blocks} B^2$ depend on the total number of blocks $N^\mathrm{tot}_\mathrm{blocks}$ in the snapshot set. Nevertheless, S_wPOD is tunable with ϵ, but increasing ϵ also increases the compression error. To compare both methods we estimate $\mathcal{E}_\mathrm{wPOD}(r, \epsilon)$ as before in a range from $10^{-5} \leq \epsilon \leq 1$ and r up to $r^* = 30$ together with the relative error

$$\mathcal{E}_\mathrm{(r)SVD}(r) = \|\mathbf{U} - \tilde{\boldsymbol{\Psi}} \tilde{\boldsymbol{\Sigma}} \tilde{\boldsymbol{V}}^T\|_F / \|\mathbf{U}\|_F \tag{26}$$

of the truncated (r)SVD in the Frobenius norm. Here $\tilde{\boldsymbol{\Psi}} \in \mathbb{R}^{M \times r}$, $\tilde{\boldsymbol{V}} \in \mathbb{R}^{N_\mathrm{s} \times r}$ are matrices, with columns composed of all spatial modes as orthonormal vectors $\boldsymbol{\Psi}_k \in \mathbb{R}^M$ and the corresponding temporal coefficients $\mathbf{v}_k \in \mathbb{R}^{N_\mathrm{s}}$ and $\tilde{\boldsymbol{\Sigma}} \in \mathbb{R}^{r \times r}$ contain the POD eigenvalues λ_k as singular values $\sigma_k = \sqrt{\lambda_k}$ on the diagonals. A direct comparison of the total approximation errors $\mathcal{E}_\mathrm{(r)SVD}$, $\mathcal{E}_\mathrm{wPOD}$ defined in Eqs. (16) and (26), respectively, is shown in the left of Fig. 9. In both figures the wPOD was set up with $\epsilon = 3.6 \times 10^{-5}$ and $J_\mathrm{max} = 5$, such that $S_\mathrm{wPOD} \approx S_\mathrm{rSVD}$.

One important aspect of this comparison has to be highlighted first. The performance of the rSVD and wPOD strongly depends on the data. The wPOD will always benefit from data with localized smooth structures, but it will be less efficient than the rSVD for cases which are distorted by random noise. In our studies the oscillating bump structures are very localized; therefore, the wPOD has an advantage over the rSVD. In fact wPOD needs less memory than the rSVD to achieve the same overall error as shown in the studied parameter range, see Fig. 10.

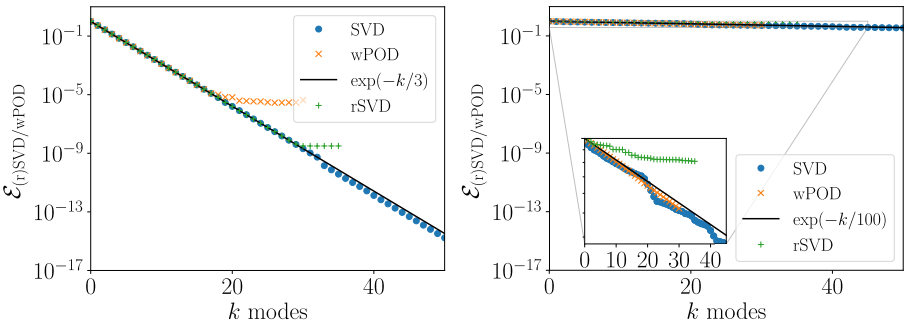


Fig. 9 Comparison of the total approximation errors $\mathcal{E}_{(r)SVD}, \mathcal{E}_{wPOD}$ for $\Delta\lambda = 3$ (left) and $\Delta\lambda = 100$ (right). As reference we plot the exact values $\exp(-k/3)$ and $\exp(-k/100)$ indicated by a black line. The rSVD is computed with 35 random samples of $\mathbf{U}$ and the wPOD set up with $\epsilon = 6 \times 10^{-3}$ and $J_{max} = 5$, to ensure approximately the same memory consumption. The inset shows a zoom

A very interesting case, that cannot be efficiently dealt with by the rSVD, is the case of slowly decaying singular values. Here, random sampling can hardly capture the column rank of $\mathbf{U}$, since all columns are nearly equally important. Note that also the SVD algorithm implemented in python's numpy package (using LAPACK, the implicit zero-shift QR algorithm after reducing $\mathbf{U}$ to bidiagonal form) exhibits numerical instabilities due to round-off errors. These errors occur, when the distance between the singular values gets close to their absolute value, which is especially visible for the case of slowly decaying singular values (see Fig. 9). This is not the case for the wavelet POD, as shown in Fig. 9. However, small deviations from the expected error decay $\sim \exp(-k/100)$ are visible. For this case we have chosen all parameters as before, except that the magnitude of the modes in Eq. (25) decays much slower with $\Delta\lambda = 100$. This result demonstrates that our method can be used in the combination with large model order reduction problems, which suffer under slowly decaying eigenvalues or singular values. We would like to highlight that especially in large transport dominated systems, as they often occur in fluid dynamics,

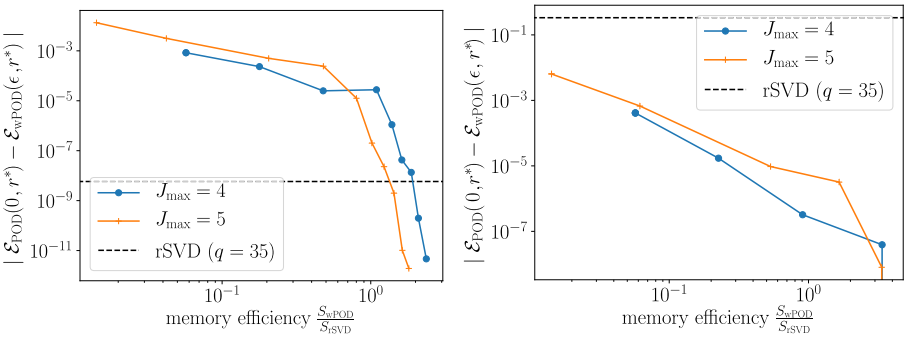


Fig. 10 Memory consumption vs. deviation from the exact POD values. Comparison between rSVD and wPOD. On the left we show the test case with $\Delta\lambda = 3$ and on the right $\Delta\lambda = 100$. The horizontal dashed lines mark the difference between the rSVD and the SVD using $q = 35$ random samples

MOR is negatively impacted by slowly decaying singular values as reported in [40]. Therefore, our method could be very useful in the treatment of such problems. We will come back to this in Section 4.2.4, where we compute a POD of a bumblebee in forward flight.

4.2 Application to 2D and 3D numerical flow data

In this section we compute a sparse POD basis of 2D and 3D flow data, computed with `WABBIT`. In the first study, we use data from a numerical simulation with an equidistant grid of a 2D flow past a cylinder. In the second application, we apply the algorithm to highly resolved 3D data of a flapping flight simulation of a bumblebee [19].

4.2.1 2D case — Von Kármán vortex street

In this first example, our dataset $\mathcal{U}$ results from a 2D simulation of an incompressible flow past a cylinder using the artificial compressibility method computed with `WABBIT`. Details about the software can be found in [19]. For this study the block structured grid of `WABBIT` is fixed at the resolution $J_{\max} = 6$, with $(B_1, B_2) = (65, 17)$ and domain size $\mathbb{D} := [0, 64] \times [0, 16]$. This is equivalent to an equidistant grid of 4096×1024 grid points. Each snapshot has three components $\mathbf{u} = (\mathbf{v}, p)$, where $\mathbf{v}$ denotes the velocity and p the pressure. For this case study we chose the vortex street for a Reynolds number $\mathrm{Re} = 200$, because it is known to have fast decaying POD truncation errors. The Reynolds number $\mathrm{Re} = 2v_\infty R/\nu$ is based on the cylinder radius $R = 1$, freestream velocity $v_\infty = 1$ and kinematic viscosity $\nu = 0.01$. The simulation was run until a stable Kármán vortex shedding was achieved. From this point onwards, we sample the solution in the time interval $500 \leq t \leq 612$ with $\Delta t = 0.5$, resulting in $N_s = 225$ snapshots. The relevant parameters for our algorithm are summarized in Table 1. For concise overview we only show the scalar-valued vorticity $\omega = \nabla \times \mathbf{v}$ computed from the velocity components of the state vector in Figs. 3 and 12 and the curl of the two velocity components of the POD basis in Fig. 21 in Appendix C.

Table 1 Parameters of the numerical test cases. Here ϵ^∞ is the wavelet threshold with CDF4,4 wavelets being normalized in L^∞

Von Kármán Vortex Street		**Bumblebee**	
Parameter	Value	Parameter	Value
Number snapshots N_s	225	Number snapshots N_s	41
Resolution $N_1 \times N_2$	4096×1024	Resolution $(J_{\max}, B_\alpha, \epsilon^\infty)$	$(7, 23, 0.01)$
Domain size $L_1 \times L_2$	64×16	Domain size L^3	8^3
Reynolds number Re	200	Reynolds number Re	2000

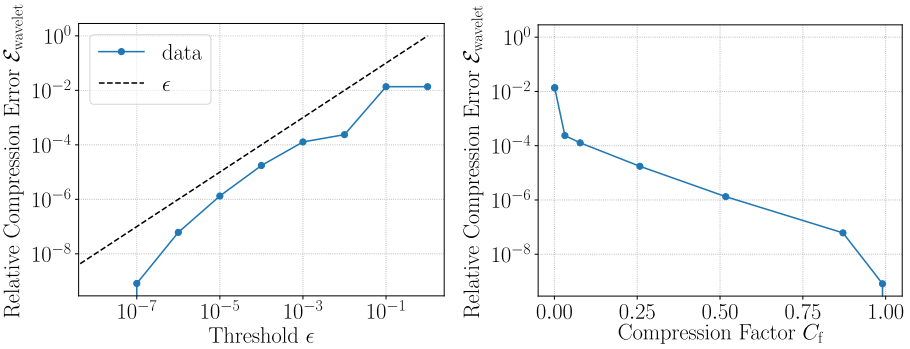


Fig. 11 Compression error and compression factor C_{f} of the vortex street. Left: The compression error in the L^2-norm is bounded by ϵ, drawn with dashed line (- - -). Right: Relative error in the L^2-norm vs. compression factor. For direct comparison the vertical axis limits of both figures are identical

4.2.2 Wavelet compression

As in the synthetic test case in Section 4.1, we first study the compression of the flow data. To this end, we sample $\mathcal{E}_{\text{wavelet}}(\epsilon)$ of one representative snapshot $\mathbf{u}$ at $t = 550$ with different $\epsilon \in \{10^{-8}, 10^{-7}, \ldots, 1\}$. In Fig. 11 we plot the relative compression error $\mathcal{E}_{\text{wavelet}}$ against the wavelet threshold ϵ (left) and against the compression factor C_{f} (right). In contrast to the scalar field in the synthetic example, the thresholded quantity is vector valued. Therefore, we normalize each state vector component before thresholding. All norms in the plots are vector norms. Note, that for the threshold $\epsilon = 0.1$ the grid is on the coarsest resolution ($j = 1$) with 4 blocks only (corresponding blocks are shown in Fig. 3). This explains why the compression errors beyond $\epsilon = 0.1$ do not differ. Similarly $\epsilon = 1 \times 10^{-8}$ corresponds to the finest resolution at grid level $j = J_{\max}$ and the error in Fig. 11 drops to zero. However, as mentioned earlier, for maximal performance the wavelet threshold ϵ should be located at the onset of the plateau, which is reached at $\sim 1 \times 10^{-2}$ (i.e., $C_{\text{f}} \sim 0.035$). Comparing this observation with the block distribution in Fig. 3, one observes that smaller values of ϵ produce denser grids, with only little gain in precision. It is remarkable that according to Fig. 11 less than 3.5% of the actual data is needed to represent the full data with an L^2-error less than 0.5‰. At this compression level we have compressed the full data with 225 snapshots from ~ 24 to ~ 0.84GB, which makes it easily manageable for most laptops.

4.2.3 POD truncation

We will now study the error behavior of the wPOD numerically for fixed threshold ϵ. To this end, we compute $\mathcal{E}_{\text{POD}}(\epsilon, r), \mathcal{E}_{\text{wPOD}}(\epsilon, r)$ for $\epsilon \in \{10^{-8}, 10^{-7}, \ldots, 1\}$. Three calculated modes and their temporal coefficients are visualized with the corresponding block structure in Appendix C, Fig. 21. Furthermore, we compare the reconstruction $\tilde{\mathbf{u}}^{\epsilon}$ in Fig. 12 for the dense case with $\epsilon = 0$ (right column) and one adaptive case $\epsilon = 10^{-2}$ (left column) using $r = 2, 6, 10$ modes. The comparison

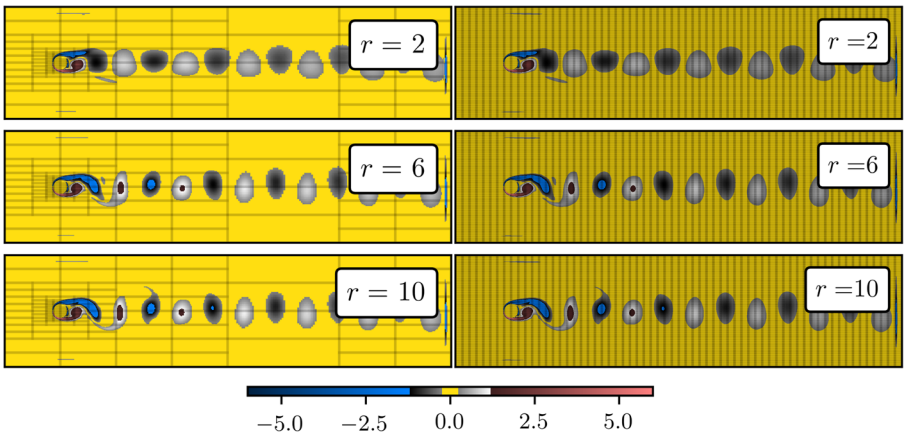


Fig. 12 Direct comparison between sparse reconstruction $\tilde{\mathbf{u}}^\epsilon$ with $\epsilon = 10^{-2}$ (left) and dense reconstruction with $\epsilon = 0.0$ (right) using $r = 2, 6, 10$ modes (from top to bottom). In both columns we display the vorticity $\tilde{\omega}^\epsilon = \partial_x \tilde{v}_y^\epsilon - \partial_y \tilde{v}_x^\epsilon$ of the reconstructed velocity components $(\tilde{v}_x^\epsilon, \tilde{v}_y^\epsilon)$

shows that with increasing number of modes the typical vortex structure is recovered. No qualitative differences, except local changes in resolution, can be seen, when comparing the adaptive and non-adaptive results.

For a quantitative analysis we have plotted the total L^2-error and the truncation error in Fig. 13. In both plots the impact of the wavelet adaption (corresponding to $\epsilon > 0$, blue lines) is visualized and compared to the results of the classical POD procedure (corresponding to $\epsilon = 0$, black line). The numerical data show the behavior stated in Section 3.4: With the wavelet adaption of the snapshots, errors are introduced, which lead to distortion of the POD eigenvalue problem and compression errors in the POD basis.

The perturbation l_k of the eigenvalues is especially visible in $\mathcal{E}_{\text{POD}}$ (lower left plot in Fig. 13) for the low energy modes ($r > 30$), corresponding to small eigenvalues. In this regime the distorted eigenvalues $\lambda_r^\epsilon = \lambda_r + l_r \epsilon$ fluctuate around the exact value λ_r (black markers). For the smallest eigenvalues the relative fluctuation can lead to a total failure of the algorithm, with even negative eigenvalues. This regime, however, can be ignored, since the total error $\mathcal{E}_{\text{wPOD}}$ will be dominated by the compression effects for large r.

Figure 13 (upper left) shows that the error behavior of $\mathcal{E}_{\text{wPOD}}$ is dominated by the truncation error $\mathcal{E}_{\text{POD}}$ for small number of modes r. With increasing r the truncation error decreases, while the error introduced by the compression remains constant. This leads to a saturation of the total error, as soon as the truncation error is smaller than the error due to compression. This saturation effect is not a special case of the chosen wavelet compression scheme, as it also appears in finite element schemes as well, see for instance [24, 51]. However, the wavelet basis has a major advantage over finite element schemes in this setting, since the grids are hierarchically structured, which is easier to handle and computationally efficient. In fact, no additional computations for "(i) collision detection, (ii) mesh intersection (detect intersection interface) and (iii)

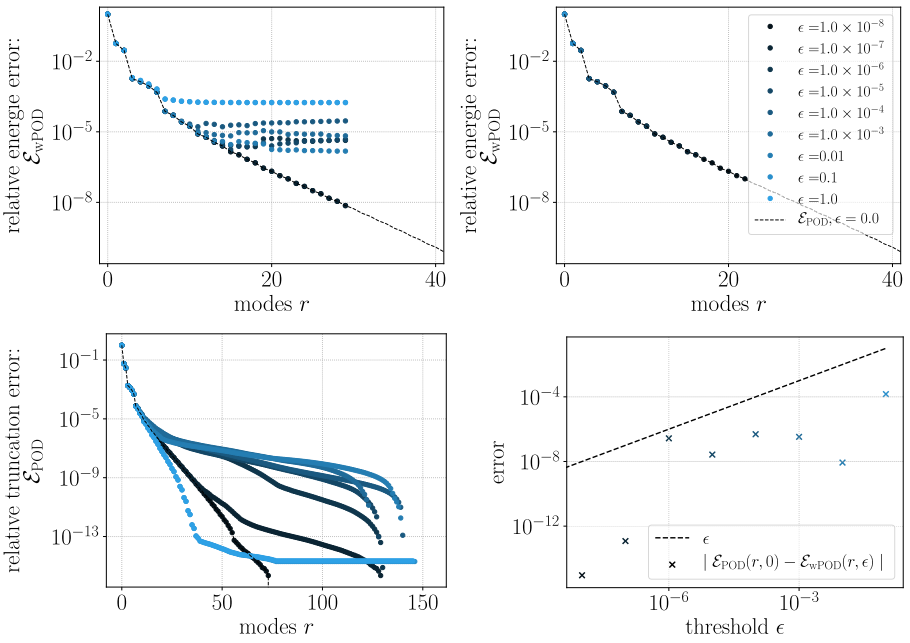


Fig. 13 Relative errors $\mathcal{E}_{\text{wPOD}}$ for the vortex street without error control (upper left) and with conservative error control (upper right). In the lower left the relative errors of the POD ($\mathcal{E}_{\text{POD}}$) and the absolute difference between the POD results and the wPOD with conservative error control (lower right): $\Delta\mathcal{E}(r,\epsilon) = |\mathcal{E}_{\text{POD}}(r,0) - \mathcal{E}_{\text{wPOD}}(r,\epsilon)|$. Note, that the difference remains below ϵ in the conservative error setting. In all plots we vary the colors from bright blue at $\epsilon = 1$ to black at $\epsilon = 0$

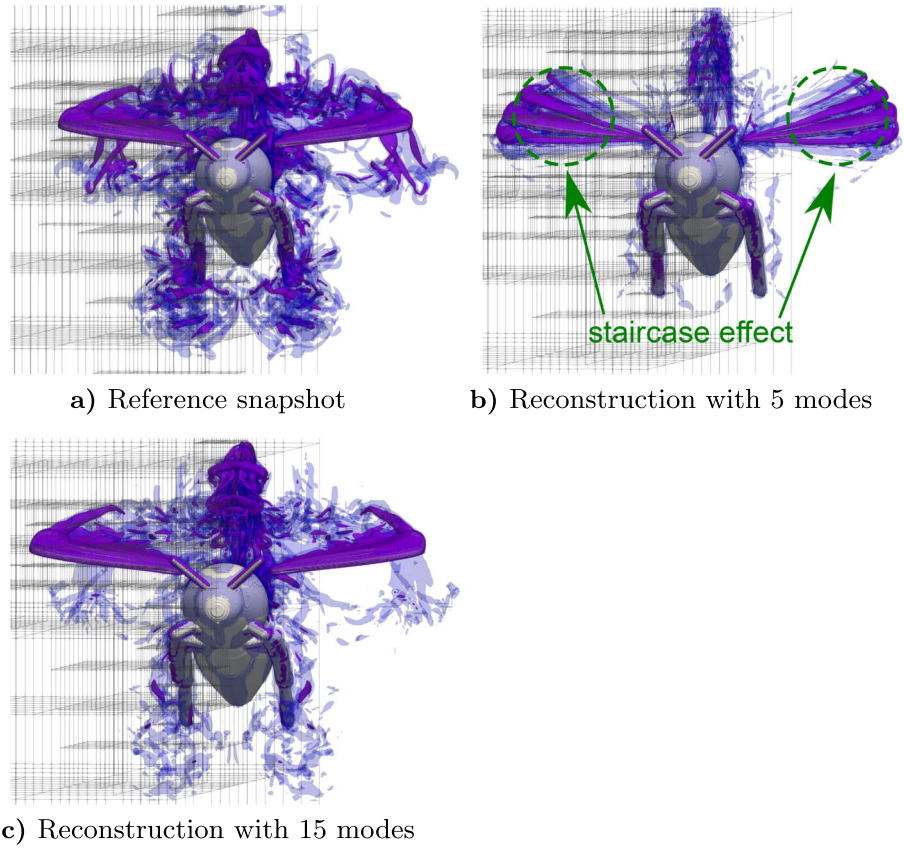
integration of complex polyhedra" [24, p. 9] or special vertex bisection triangulation, as discussed in [51], are needed.

If we choose the conservative error setting $\epsilon^* < \mathcal{E}^*$, recommended in Section 3.4, with for example $\epsilon^* = 0.1\mathcal{E}^*$ and truncate all modes for which $\mathcal{E}_{\text{POD}}(\epsilon^*, r) \leq \mathcal{E}^*$ we obtain the results shown in the upper right of Fig. 13. This is essentially the same figure as shown in the left, but all points are excluded which do not fulfill the conservative error criterion. With the help of the conservative error setting, we are able to control the errors introduced by the perturbation of the eigenvalues. Therefore, the difference $|\mathcal{E}_{\text{wPOD}}(\epsilon, r) - \mathcal{E}_{\text{POD}}(0, r)|$, shown in the lower right of Fig. 13, stays below ϵ.

4.2.4 3D case — insect flight

The data come from a three-dimensional, highly resolved block-based adaptive simulation of a bumblebee in forward flight using `WABBIT` [49]. A summary of relevant parameters is given in Table 1. Additional details of the adaptive flight simulation can be found in [19].

One representative snapshot is shown in Fig. 14a together with the reconstruction of our algorithm using either 5 (Fig. 14b) or 15 modes (Fig. 14c). Additionally we plot some selected modes in Fig. 15. The wPOD algorithm is applied to the vorticity



a) Reference snapshot

b) Reconstruction with 5 modes

c) Reconstruction with 15 modes

Fig. 14 Comparison between a bumblebee snapshot at time $t_i, i = 13$ a) and its POD reconstruction $\tilde{\mathbf{u}}_i^\epsilon$ using 5 modes b) and 15 modes c) with $\epsilon = 0.01$. Shown are two isosurfaces of the magnitude of vorticity, i.e., $\|\operatorname{curl}(\mathbf{v})\|_2 = 50$ and 100. The rigid body of the bumblebee is displayed in gray. The grid structure is indicated behind. Staircase effects at the wings are indicated in Fig. 14b. These artifacts appear when POD is applied to sharp structures or discontinuities which move

vector $\mathbf{w} = \operatorname{curl}(\mathbf{v})$, which is computed from the velocity $\mathbf{v}$. Two isosurfaces of the magnitude of the vorticity, 50 and 100, are shown in Figs. 14 and 15.

The moving wing geometry causes large gradients of the flow field at the interfaces of the object. This large gradients move with the flapping wing and cause major problems to the POD, like staircase effects (see Fig. 14b) of the reconstructed field with slowly decaying energy error (see Fig. 16). This drawback of the POD is known for transport dominated fields with large gradients and is theoretically studied in [26, 40] with help of the Kolmogorov n-width. Although methods like [32] are available for parametric moving discontinuities, we obviate from using them here, since it is not within the scope of this work. However, wavelet adaptation reduces the amount of computational resources needed in favor of additional accuracy, like increased number of modes. In fact, for the data presented here ($J_{\max} = 7, B = 23, \epsilon^\infty = 0.01, N_{\text{blocks}} \leq 8000$) the factor in memory savings in comparison to the dense grid

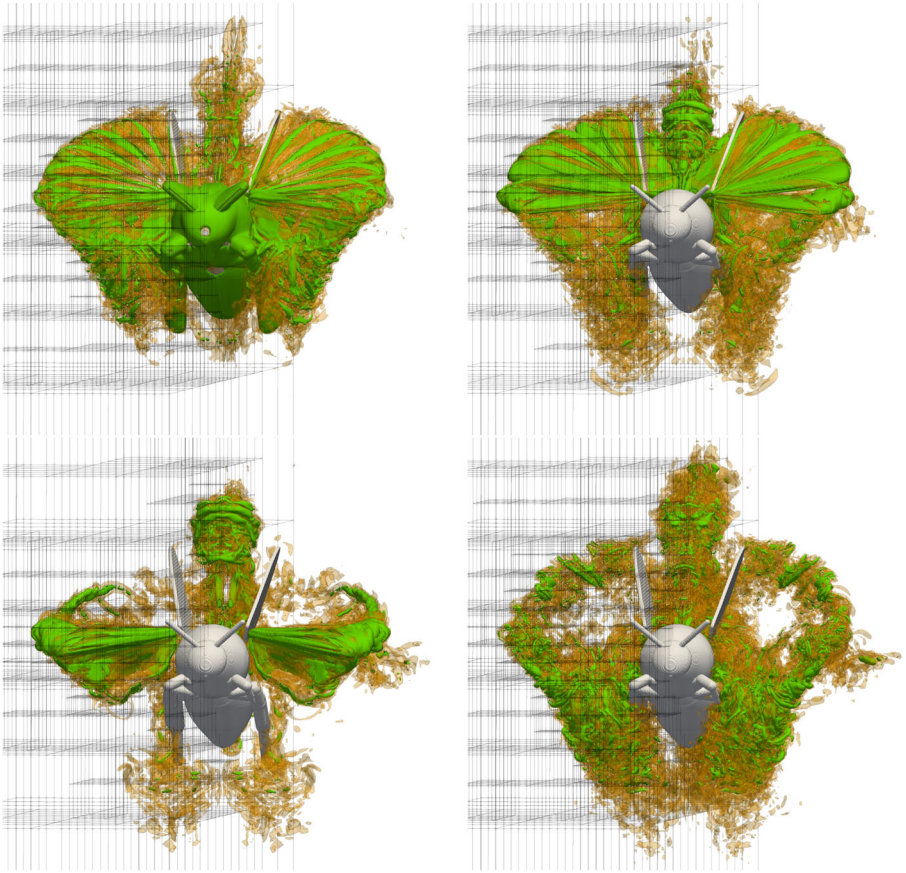


Fig. 15 Bumblebee modes $\boldsymbol{\Psi}_i^\epsilon$, with $i = 1, 9, 18, 27$ and $\epsilon = 1 \times 10^{-4}$ in row major order. The modes are visualized as isosurfaces of the magnitude $\|\boldsymbol{\Psi}_i^\epsilon\|_2 = 10, 20$ with colors green and orange, respectively. For reference, the rigid body of the insect is shown in gray. The adaptive grid is only indicated on the left of each figure

($N_{\text{blocks}} = 2^{3J_{\max}} = 2097152$) is larger than 260. This factor can be further increased, when increasing ϵ. A full POD would be prohibitive because of its tremendous memory demand of approximately 31 TB ($N_{\text{b}} = 2^{3J_{\max}} N_{\text{s}}$ Blocks with 0.4 MB each). It should be further noted that, to the best of our knowledge, all previous results in the literature including [9, 20, 24, 25, 32, 51] have been only applied to 1D or 2D cases.

The statements about the error made in Section 3.4 also hold for the adaptive data: Here the slowly decaying eigenvalues are rather large compared to their perturbation. Hence, $\mathcal{M}_r$ is negligible and the total error behavior is mainly dominated by $\mathcal{E}_{\text{POD}}(r, 0)$ and $\mathcal{E}_{\text{wavelet}}(\epsilon)$ (see Fig. 16, left). For the case of $\epsilon = 1.0$, it can be nicely seen that the truncation error dominates the total error, after it falls below the compression error plateau $\mathcal{E}_{\text{wavelet}}(\epsilon)^2$ at $r \geq 20$. In Fig. 16 (right) the difference between the total error $\mathcal{E}_{\text{wPOD}}$ and the POD truncation error $\mathcal{E}_{\text{POD}}$ is shown. Note that for adaptive input data of the bumblebee $\mathcal{E}_{\text{POD}}(0, r)$ can be only assessed approximately, since

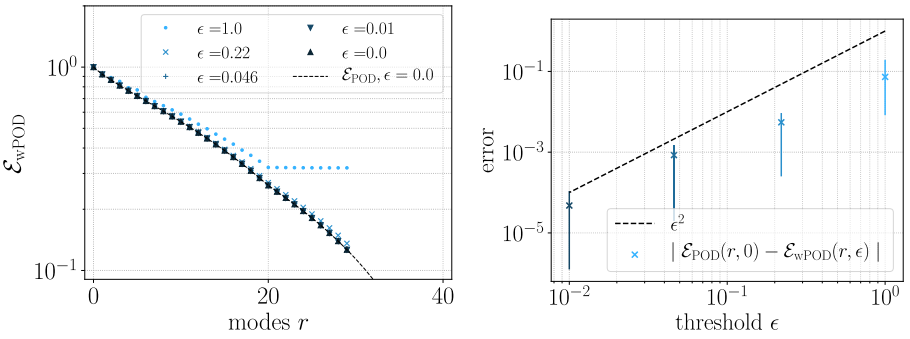


Fig. 16 Total relative error, $\mathcal{E}_{\mathrm{wPOD}}$, of the wPOD (left) and difference of the wPOD and POD error, $\Delta\mathcal{E}(r,\epsilon) = |\mathcal{E}_{\mathrm{POD}}(r,0) - \mathcal{E}_{\mathrm{wPOD}}(r,\epsilon)|$ (right). The error bars in the right figure indicate the minimal and maximal value of $\Delta\mathcal{E}$ for all ranks $1 \leq r \leq 30$ and the markers the mean $\overline{\Delta\mathcal{E}}(\epsilon) = 1/30 \sum_{r=1}^{30} \Delta\mathcal{E}(r,\epsilon)$

wavelet details have been already discarded during the generation of the data. Hence for the adaptive case $\mathcal{E}_{\mathrm{POD}}(0,r)$ means that no additional compression errors were introduced during the wPOD algorithm. The difference scales quadratically with the compression error $\mathcal{E}_{\mathrm{wavelet}}(\epsilon)^2 \sim \epsilon^2$, drawn as dashed line in Fig. 16 (right).

From these results we thus can conclude that our algorithm is able to reproduce the POD eigenvalue spectra of the original data, even for larger thresholds $\epsilon > 0.01$ within a predefined precision given by the squared wavelet compression error. As shown in our synthetic test case, this would be a challenge for the randomized SVD, since the eigenvalues decay slowly.

5 Conclusion and outlook

In this paper we presented a novel method to calculate the proper orthogonal decomposition for two or three-dimensional data, typically velocity or vorticity fields, given either on equidistant or block-based adaptive grids, the later obtained with wavelet adaptation. The so-called wPOD algorithm endows the method of snapshots, proposed by Sirovich [47] with nonlinear approximation for computing the POD efficiently. Our method makes use of a multiresolution framework called `WABBIT` to spatially adapt vector fields and thus generating sparse POD modes using wavelet compression on block-based grids.

The introduced compression errors of the POD basis are well controlled using nonlinear approximation by means of wavelet thresholding. While the compression error depends linearly on the chosen threshold ϵ, the total energy error of the wPOD procedure scales with ϵ^2. The wavelet compression and POD truncation errors can thus be balanced, assuming that the perturbation of the eigenvalues remains sufficiently small. In comparison to the classical POD, the compression results in overall savings in memory and computational resources, while obtaining approximately the same error. As a consequence, data from highly resolved 3D direct numerical simulation computed on massively parallel platforms can be processed on much smaller systems.

However, these savings in terms of computational effort are accompanied by additional overhead for handling the tree-like data structure. Hence, the wPOD cannot yet compete with equivalent randomized techniques [27, 56] in terms of CPU time, if the memory is not a limiting factor. Nevertheless, we were able to show that our method can handle data efficiently, when the singular values decay slowly. In this case, the proposed wPOD does indeed outperform the rSVD. Furthermore, our method expresses the correlation matrix in terms of the underlying wavelet basis, similar to what has been done in the context of finite elements in [24]. However, using the scaling relations of wavelets allows to express the inner products (Eq. (13)) effectively and thus avoids many problems associated with finite element schemes, cf. listed caveats in [24, p. 6].

The framework could be extended to non-periodic grids using boundary adapted wavelets (see, e.g., [43] for applications to channel flows) as well as to other algorithms like DMD [44], shifted POD [41] or multiscale POD [38]. Furthermore, our framework provides interesting perspectives for reduced order models introducing adaptive POD-Galerkin simulations of PDEs, a direction we envisage in future studies. It has the potential to become a practical tool for turbulence research when combining it with wavelet denoising for coherent vortex structures [21, 22] as we are able to split the flow into coherent structures and incoherent background noise before computing a POD basis.

Finally, in combination with wavelet denoising, the proposed wPOD technique could be likewise applied to highly resolved flow images obtained experimentally, e.g., using particle image velocimetry or Schlieren imaging.

Appendix A. Block-based wavelet adaptation

A.1 Refinement and coarsening

For the wavelet adaptation scheme, here illustrated for the two-dimensional case, we assume real valued and continuous L^2-functions $u(x, y)$, such as the pressure or velocity component of a flow field. The function is sampled on a block-based multiresolution grid as shown in Fig. 1, with maximum tree level $J_{\max}$, block size $\mathrm{B}_1 \times \mathrm{B}_2$ and a ghost node layer of size g, needed for synchronization. The sampled values on a block $\mathcal{B}_p^j$ are denoted by:

$$x^j_{p,k_1} = x_p + k_1 \Delta x^j \qquad k_1 = -g, \ldots, \mathrm{B}_1 + g - 1 \tag{27}$$

$$y^j_{p,k_2} = y_p + k_2 \Delta y^j \qquad k_2 = -g, \ldots, \mathrm{B}_2 + g - 1 \tag{28}$$

$$\underline{u}^j[p, k_1, k_2] := u\left(x^j_{p,k_1}, y^j_{p,k_2}\right) \tag{29}$$

For a block *refinement* $j \to j+1$ the lattice spacings are halved and dyadic points are added to the block, as shown in Fig. 17.

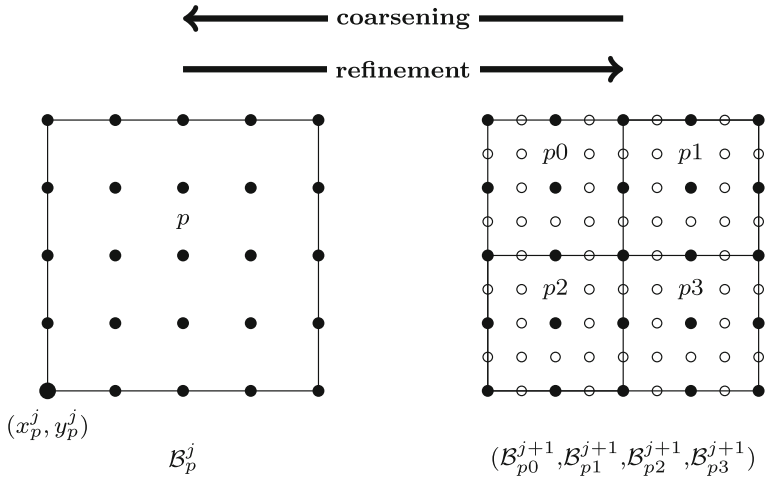


Fig. 17 Dyadic grid refinement and coarsening of a single block. Refinement: First the block $\mathcal{B}_p^j$ is refined by midpoint insertion and then split into four new blocks $(\mathcal{B}_{p0}^j, \mathcal{B}_{p1}^j, \mathcal{B}_{p2}^j, \mathcal{B}_{p3}^j)$. Coarsening: After a low pass filter is applied all midpoints are removed and merged into one block

The values at the refined blocks can be obtained by the refinement relation:

$$\widehat{\underline{\mathrm{u}}}^{j+1}[p, l_1, l_2] = \sum_{k_1=-g}^{\mathrm{B}_1+g-1} \sum_{k_2=-g}^{\mathrm{B}_2+g-1} h_{l_1-2k_1} h_{l_2-2k_2}\, \underline{\mathrm{u}}^j[p, k_1, k_2] \tag{30}$$

where h_k denotes the weights of the one-dimensional interpolation scheme:

$$\varphi(x) = \sum_{k=-N}^{N} h_k \varphi(2x - k). \tag{31}$$

In the following, we will refer to Eq. (30) as the *prediction operation* $P_j^{j+1} : \underline{\mathrm{u}}^j \mapsto \widehat{\underline{\mathrm{u}}}^{j+1}$ as it was introduced for point value multiresolution in [28]. Using Eq. (31) one can show that Eq. (30) is equivalent to the continuous refinement relation:

$$\widehat{u}^{(p)}(x, y) = \sum_{k_1=-g}^{\mathrm{B}_1+g-1} \sum_{k_2=-g}^{\mathrm{B}_2+g-1} \underline{\mathrm{u}}^j[p, k_1, k_2] \varphi_{p,k_1,k_2}^j(x, y)\,, \tag{32}$$

$$\text{where} \quad \varphi_{p,k_1,k_2}^j(x, y) = \varphi\left(\frac{x - x_{p,k_1}^j}{\Delta x^j}\right) \varphi\left(\frac{y - y_{p,k_2}^j}{\Delta y^j}\right) \tag{33}$$

is the two-dimensional tensor product of one-dimensional *interpolating scaling functions* $\varphi(x)$.

Table 2 Filter coefficients h_k and dual filter coefficients $\tilde{h}_k$ of the Cohen-Daubechies-Feauveau (CDF 4,4) wavelets applied in the prediction/restriction operation

continuous	k	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
$\tilde{\varphi}$	$\tilde{h}_k$	$-\frac{1}{256}$	0	$\frac{9}{128}$	$-\frac{1}{16}$	$-\frac{63}{256}$	$\frac{9}{16}$	$\frac{87}{64}$	$\frac{9}{16}$	$-\frac{63}{256}$	$-\frac{1}{16}$	$\frac{9}{128}$	0	$-\frac{1}{256}$
φ	h_k				$-\frac{1}{16}$	0	$\frac{9}{16}$	1	$\frac{9}{16}$	0	$-\frac{1}{16}$			

When a block is *coarsened*, the tree level is decimated by one: $j+1 \to j$ and every second grid point is removed. The values at the coarser level are obtained by the coarsening relation:

$$\underline{\mathrm{u}}^{j-1}[p, l_1, l_2] = \sum_{k_1=-g}^{\mathrm{B}_1+g-1} \sum_{k_2=-g}^{\mathrm{B}_2+g-1} \tilde{h}_{2l_1-k_1} \tilde{h}_{2l_2-k_2}\, \underline{\mathrm{u}}^{j}[p, k_1, k_2] \tag{34}$$

In the notation of Harten [28] coarsening is called *decimation* and Eq. (34) is denoted by $D_j^{j-1}\colon \underline{\mathrm{u}}^j \to \underline{\mathrm{u}}^{j-1}$ in the following. After decimation the block will be merged with its neighboring blocks, as shown in Fig. 17. Similar to the continuous refinement relation Eq. (32) there is a continuous counterpart for coarsening: the *dual scaling function* $\tilde{\varphi}$, which satisfies

$$\tilde{\varphi}(x) = \sum_{k=-N}^{N} \tilde{h}_k \tilde{\varphi}(2x-k). \tag{35}$$

Instead of using *Deslauriers-Dubuc* (DD) wavelets, as in the framework of Harten [28], we use lifted Deslauriers-Dubuc wavelets, i.e., *biorthogonal Cohen-Daubechies-Feauveau wavelets* of fourth order (CDF 4,4) with filter coefficients h_k and dual filter coefficients $\tilde{h}_k$ listed in Table 2. The lifted DD wavelets allow a better scale separation and can be easily implemented replacing the loose downsampling filter by a low pass filter before coarsening the grid.

A.2 Computing wavelet coefficients and adaptation criterion

With the definition in Eqs. (30) and (34) we have introduced a biorthogonal multiresolution basis $\{\tilde{\varphi}^j_{p,k_1,k_2}, \varphi^j_{p,k_1,k_2}\}$, which can approximate any continuous function $u \in L^2(\mathbb{D})$ arbitrary close. This aspect is the main property of a multiresolution analysis and can be used to relate and compare samples at different resolutions, i.e., different scales.

The difference between two consecutive approximations can be represented by a wavelet with its corresponding coefficients

$$d^j[p, k_1, k_2] = \left[u^j - P^j_{j-1} D^{j-1}_j u^j\right]_{p,k_1,k_2} = \underline{\mathrm{u}}^j[p, k_1, k_2] - \widehat{\underline{\mathrm{u}}}^j[p, k_1, k_2] \tag{36}$$

known as *wavelet details*.

Algorithm 1 Wavelet adaptation

Require: $\mathbf{u}\colon \Omega \to \mathbb{R}^K$ where $\Omega \subset \mathbb{R}^d$ defined in Eq. 3 and $d \in \{2, 3\},\ K \in \mathbb{N}$
Require: threshold $\epsilon \geq 0$ and minimal and maximum tree level $J_{\min}, \mathrm{J}_{\max}$
function ADAPT($\mathbf{u}, \epsilon, J_{\min}, \mathrm{J}_{\max}$)
  set $N_{\text{blocks}} = 0$
  while $\left|\bigcup_j \Lambda^j\right| \neq N_{\text{blocks}}$ **do** ▷ *coarsening is stopped if number of blocks has not changed*
    Synchronize all ghost layers of the blocks
    Count the number of active blocks N_{blocks}
    for all $p \in \bigcup_j \Lambda^j$ **do** ▷ *loop over all active treecodes*
      Compute the refinement indicator of the block:
      1) normalize every state vector component:
      $(\mathbf{u}^{(p)})_i \leftarrow (\mathbf{u}^{(p)})_i / \|(\mathbf{u}^{(p)})_i\|, i = 1, \ldots, K$
      2) compute $\widehat{\mathbf{u}}_p^j$ using Eqs. 30 and 34 in a restriction and prediction step
      3) compute the detail coefficients of all state vector components with Eq. 37
      if $i_\epsilon(\mathcal{B}_p) = 1$ and $J_{\min} \leq j - 1$ **then**
        tag the block $\mathcal{B}_p$ for coarsening
      end if
    end for
    Remove coarsening tag from blocks, if coarsening would result in a non graded grid.
    Coarse all blocks taken for coarsening
  end while
  Balance the load between processors
  return $\mathbf{u}^\epsilon$ ▷ *wavelet filtered field*
end function

As shown in one space dimension by Unser in [52] for biorthogonal wavelets the difference d^j between two consecutive approximations is bounded for sufficiently smooth L^2 functions u. The bound depends on the local regularity of u and the order N (here $N = 4$) of the scaling function:

$$\|d^j\| = \|u - \widehat{u}\| \leq C_{\varphi,\tilde{\varphi}} (\Delta x^j)^N \|u^{(N)}\| \tag{37}$$

where $\Delta x^j \sim 2^j$ is the step size, $C_{\varphi,\tilde{\varphi}}$ is a constant independent of u. This result can be understood as an interpolation error, since the CDF4,4 filter can be viewed as an interpolation of the averaged data $D_j^{j-1}\underline{u}^j$. From Eq. (37) we can thus conclude: since the approximation error of the interpolation scheme depends on the smoothness of the sampled function and the lattice spacing on the block, we can increase the local lattice spacing of the block for blocks where the function is smooth and keep the fine scales only for blocks where u_p is not smooth. This is achieved by coarsening the block, as decreasing the tree level j increases the lattice spacing. For an intended

approximation error we therefore define the *wavelet threshold* ϵ together with the *coarsening indicator* $i_\epsilon(\mathcal{B}_p^j)$:

$$i_\epsilon(\mathcal{B}_p^j) := \begin{cases} 1 & \text{if } \max_{\substack{k_1=0,\dots,\mathrm{B}_1-1 \\ k_2=0,\dots,\mathrm{B}_2-1}} |\, d^j[p,k_1,k_2] \,| < \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{38}$$

which marks the block for coarsening. For vector-valued quantities $\mathbf{u} = (u_1, u_2, \dots, u_K)$, a block will be coarsened only if all components indicate coarsening. The pseudocode in Algorithm 1 outlines the wavelet adaptation algorithm for vector-valued quantities. For the sake of completeness we will put this algorithm in relation to the underlying wavelet representation.

A.3 Wavelet representation in the continuous setting

For completeness of this manuscript we give a detailed description of the underlying wavelet representation in a concise way. For the interested reader we recommend Ref. [50] for a succinct introduction to biorthogonal wavelets and to Refs. [11, 12, 15, 35] for a more detailed description.

Assuming that we have sampled a continuous function $u \in L^2(\mathbb{D})$ inside a domain $\mathbb{D} \subset \mathbb{R}^2$ on an equidistant grid corresponding to refinement level $J_{\max}$, we can block-decompose it in terms of Eq. (29). By choosing $j = J_{\max}$ in Eq. (32) and summing over all blocks p, we can thus represent u in L^2 using a basis of dilated and translated scaling functions $\{\varphi^j_{p,k_1,k_2}\}$:

$$u = \sum_{p \in \Lambda^{J_{\max}}} u^{(p)} = \sum_{\lambda \in \overline{\Lambda}^{J_{\max}}} c_\lambda^{J_{\max}} \varphi_\lambda^{J_{\max}} . \tag{39}$$

Here we have introduced a multi-index $\lambda = (p, k_1, k_2) \in \overline{\Lambda}^j := \Lambda^j \times \{0, \dots, \mathrm{B}_1-1\} \times \{0, \dots, \mathrm{B}_2-1\}$ for ease of notation. With this notation we can rewrite Eq. (39) in a wavelet series

$$u = \sum_{\lambda \in \overline{\Lambda}^{J_{\min}}} c_\lambda^{J_{\min}} \varphi_\lambda^{J_{\min}} + \sum_{j=J_{\min}}^{J_{\max}-1} \sum_{\lambda \in \overline{\Lambda}^j} \sum_{\mu=1}^{3} d_{\mu\lambda}^j \psi_{\mu,\lambda}^j, \tag{40}$$

where the interpolating scaling basis $\{\varphi_\lambda^{J_{\min}}\}_{\lambda \in \overline{\Lambda}^{J_{\min}}}$ approximates u at the coarsest scale $J_{\min}$ and the wavelet basis $\{\psi_{\mu,\lambda}^j\}_{\mu=1,2,3,\lambda \in \overline{\Lambda}^{j \geq J_{\min}}}$ contains all the additional information necessary to construct u.

In the CDF4,4 setting we have biorthogonal scaling functions:

$$\varphi_\lambda^j(x,y) = \varphi\left(\frac{x-x_\lambda^j}{\Delta x_p^j}\right)\varphi\left(\frac{y-y_\lambda^j}{\Delta y_p^j}\right),\ \tilde{\varphi}_\lambda^j(x,y) = \tilde{\varphi}\left(\frac{x-x_\lambda^j}{\Delta x_p^j}\right)\tilde{\varphi}\left(\frac{y-y_\lambda^j}{\Delta y_p^j}\right)$$
$$\text{with} \quad \langle \varphi_{\lambda_1}^j, \tilde{\varphi}_{\lambda_2}^j \rangle = \delta_{\lambda_1,\lambda_2}$$

and the associated three biorthogonal wavelets (in the d-dimensional case we have $2^d - 1$ wavelets see [39, p. 42])

$$\psi^j_{1,\lambda}(x,y) = \psi\left(\frac{x-x^j_\lambda}{\Delta x^j_p}\right)\varphi\left(\frac{y-y^j_\lambda}{\Delta y^j_p}\right),\ \tilde{\psi}^j_{1,\lambda}(x,y) = \tilde{\psi}\left(\frac{x-x^j_\lambda}{\Delta x^j_p}\right)\tilde{\varphi}\left(\frac{y-y^j_\lambda}{\Delta y^j_p}\right)$$
$$\psi^j_{2,\lambda}(x,y) = \varphi\left(\frac{x-x^j_\lambda}{\Delta x^j_p}\right)\psi\left(\frac{y-y^j_\lambda}{\Delta y^j_p}\right),\ \tilde{\psi}^j_{1,\lambda}(x,y) = \tilde{\varphi}\left(\frac{x-x^j_\lambda}{\Delta x^j_p}\right)\tilde{\psi}\left(\frac{y-y^j_\lambda}{\Delta y^j_p}\right)$$
$$\psi^j_{3,\lambda}(x,y) = \psi\left(\frac{x-x^j_\lambda}{\Delta x^j_p}\right)\psi\left(\frac{y-y^j_\lambda}{\Delta y^j_p}\right),\ \tilde{\psi}^j_{1,\lambda}(x,y) = \tilde{\psi}\left(\frac{x-x^j_\lambda}{\Delta x^j_p}\right)\tilde{\psi}\left(\frac{y-y^j_\lambda}{\Delta y^j_p}\right)$$
$$\text{with}\quad \langle \psi^{j_1}_{\mu_1,\lambda_1}, \tilde{\psi}^{j_2}_{\mu_2,\lambda_2}\rangle = \delta_{\mu_1,\mu_2}\delta_{\lambda_1,\lambda_2}\delta_{j_1,j_2}$$

for the horizontal ($\mu = 1$), vertical ($\mu = 2$) and diagonal ($\mu = 3$) direction. The wavelet and its dual are defined by the same scaling relations as φ and $\tilde{\varphi}$:

$$\psi(x) = \sum_{k=-N}^{N} g_k \psi(2x-k) \quad \text{and} \quad \tilde{\psi}(x) = \sum_{k=-N}^{N} \tilde{g}_k \tilde{\psi}(2x-k). \tag{41}$$

The filter coefficients are given by $g_k = (-1)^{1-n}\tilde{h}_k$ and $\tilde{g}_k = (-1)^{1-n}h_k$ with $h_k, \tilde{h}_k$ listed in Table 2. The components of the scaling and wavelet coefficients (c^j_λ and d^j_λ) are determined via component-wise projection of Eq. (40) onto the dual basis $\{\tilde{\varphi}^j_\lambda, \tilde{\psi}^j_{\mu\lambda}\}$:

$$c^j_\lambda = \langle u, \tilde{\varphi}^j_\lambda\rangle, \qquad d^j_{\mu\lambda} = \langle u, \tilde{\psi}^j_{\mu\lambda}\rangle \qquad \text{where} \quad \lambda = (p, k_1, k_2) \tag{42}$$

assuming that $\langle \varphi^j_{\lambda_1}, \tilde{\psi}^j_{\lambda_2}\rangle = \langle \psi^j_{\lambda_1}, \tilde{\varphi}^j_{\lambda_2}\rangle = 0$ are orthogonal. Note that $\langle a, b\rangle = \int_{\mathbb{D}} a(\mathbf{x})b(\mathbf{x})\mathrm{d}\mathbf{x}$ denotes the L^2-inner product.

In most wavelet adaptation schemes, one truncates Eq. (40) such that only detail coefficients are kept which carry significant information. According to [45] "this can be expressed as a nonlinear filter", which acts as a cutoff wavelet coefficients with small magnitude. The cutoff is given by the threshold parameter $\epsilon > 0$. However, in contrast to these schemes, our block-based adaptation groups the detail coefficients in blocks, i.e., all details on the block are kept if at least one detail carries important information. This seems to be inefficient at first sight, because unnecessary information is kept, but grouping details in blocks is reasonable, since the block-based adaptation is computationally efficient for MPI distributed architectures. Moreover, groups of significant details are often nearest neighbors, rather than a single significant detail in a block. Therefore, we define the set of blocks

$$I^j_\epsilon := \left\{ p \in \Lambda^j \;|\; \max_{\substack{k_1=0,\dots,\mathrm{B}_1-1\\ k_2=0,\dots,\mathrm{B}_2-1}} \left|d^j[p,k_1,k_2]\right| > \epsilon \right\} \tag{43}$$

with significant details in the predefined tree level range $J_{\min} \le j \le J_{\max}$. In the spirit of our previous notation we thus define: $\overline{I}^j_\epsilon := I^j_\epsilon \times \{0,\dots,\mathrm{B}_1-1\}\times$

$\{0, \ldots, B_2-1\}$ for the set of all significant detail indices. The filtered block-based wavelet field in Eq. (40) can now be written as follows:

$$u^\epsilon = \sum_{\lambda \in \overline{\Lambda}^{J_{\min}}} c_\lambda^{J_{\min}} \varphi_\lambda^{J_{\min}} + \sum_{j=J_{\min}}^{J_{\max}-1} \sum_{\lambda \in \overline{I}_\epsilon^j} \sum_{\mu=0}^{3} d_{\mu,\lambda}^j \psi_{\mu,\lambda}^j \tag{44}$$

In the following we will denote all fields with an upper index ϵ, which have been filtered with Algorithm 1 and can be thus expressed as Eq. (44).

For illustration we have computed the vorticity $\omega^\epsilon = \partial_x v_y^\epsilon - \partial_y v_x^\epsilon$ of a thresholded vector field $\mathbf{u}^\epsilon = (v_x^\epsilon, v_y^\epsilon, p^\epsilon)$ in Fig. 3 for various ϵ (more details can be found in Section 4.2.1). Here, $\epsilon > 0$ and $\epsilon = 0$ corresponds to a filtered and unfiltered field, respectively. For increasing ϵ, less detail coefficients will be above the threshold and therefore the number of blocks decreases.

Taking the difference between the thresholded Eq. (44) and the original field Eq. (40) only details below the threshold are left. Hence the total error can be estimated and yields:

$$\|u - u^\epsilon\| \leq \sum_{j=J_{\min}}^{J_{\max}-1} \sum_{\lambda \in \overline{I}_\epsilon^{j\,c}} \sum_{\mu=0}^{3} \left| d_{\mu,\lambda}^j \right| \|\psi_{\mu,\lambda}^j\| . \tag{45}$$

Because $\|\psi_{\mu,\lambda}^j\|_\infty = 1$ we finally get $\|u - u^\epsilon\|_\infty \leq C\epsilon$ for the total error in the L^∞-norm. Similarly one can normalize $\psi_{\mu,\lambda}^j$ in the L^2-norm, which corresponds to re-weighting the thresholding criterion $\left| d_{\mu,\lambda}^j \right| < \epsilon$ with a level (j) and dimension (d) dependent threshold: $\left| d_{\mu,\lambda}^j \right| < \epsilon_0 2^{-d(j-J_{\max})/2} \epsilon$ [42]. The additional constant $\epsilon_0 = 1$ for $d = 3$ and $\epsilon_0 = 0.1$ for $d = 2$ tunes the offset of the compression error. It is chosen such that the relative compression error $\mathcal{E}_{\text{wavelet}}$ is close to, but still below ϵ, i.e., it fulfills Eq. (6) in the L^2-norm.

A.4 L^2 Inner products expressed in the wavelet basis

The L^2 inner product is computed as a weighted sum of two fields $\mathbf{u}$ and $\boldsymbol{v}$. For this we first refine both fields onto a unified grid with identical tree codes Λ^j, as explained in Section 3.2. Then, we are able to compute Eq. (14) as a weighted sum over all blocks:

$$\langle \mathbf{u}, \boldsymbol{v} \rangle = \sum_{j=1}^{J_{\max}} \sum_{p \in \Lambda^j} \langle \mathbf{u}^{(p)}, \boldsymbol{v}^{(p)} \rangle = \sum_{j=1}^{J_{\max}} \sum_{p \in \Lambda^j} \langle \underline{\mathbf{u}}^{(p)}, \underline{\mathbf{v}}^{(p)} \rangle_{\boldsymbol{I}_K \otimes \boldsymbol{W} \otimes \boldsymbol{W}} \Delta x_p \Delta y_p \tag{46}$$

$$\text{with weights: } (\boldsymbol{W})_{lm} = \langle \varphi_l^j, \varphi_m^j \rangle, \tag{47}$$

Note that this quadrature rule is exact for $\epsilon = 0$. We denote by $\boldsymbol{I}_K \otimes \boldsymbol{W} \otimes \boldsymbol{W}$ the Kronecker product between the weight matrix $\boldsymbol{W}$ and the identity matrix $\boldsymbol{I}_K \in \mathbb{R}^{K,K}$. The weight matrix is pre-computed by Eq. (47) and its non-vanishing values $(\boldsymbol{W})_{ik} = w_{i-k}$ are shown in Table 3. The listed matrix elements are the discrete values of the autocorrelation function between two compactly supported scaling functions φ, see

Table 3 Values of the autocorrelation function $w_k = \int \varphi(x-k)\varphi(x)\mathrm{d}x$ of Deslauriers-Dubuc interpolating functions of order $N = 2$ and $N = 4$ in the interior of the block. The values for DD4 are rounded to the 4th digit

$\lvert k\rvert$ (Order)	0	1	2	3	4	5
w_k $(N=2)$	2/3	1/6				
w_k $(N=4)$	0.8001,	0.1370,	−0.0402,	0.0028,	-7.5925×10^{-5},	-1.4829×10^{-7}

Fig. 18. Therefore, $\boldsymbol{W}$ is sparse, symmetric and circulant. Since $\boldsymbol{W}$ is also strictly diagonal dominant and all diagonal entries are positive, $\boldsymbol{W}$ and the Kronecker product of such matrices is also positive definite.

Appendix B. Derivation of the error estimation given in Eq. (23)

Using Eq. (18) the total error in Eq. (16) becomes

$$\mathcal{E}_{\mathrm{wPOD}} \leq \frac{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i - \mathbf{u}_i^{\epsilon}\|^2}{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i\|^2} + \frac{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i^{\epsilon} - \tilde{\mathbf{u}}_i^{\epsilon}\|^2}{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i\|^2}. \tag{48}$$

Furthermore we can simplify the first term with Eq. (19) inserting $\|\mathbf{u}_i - \mathbf{u}_i^{\epsilon}\| \leq \epsilon\|\mathbf{u}_i\|$ into the nominator:

$$\frac{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i - \mathbf{u}_i^{\epsilon}\|^2}{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i\|^2} = \frac{\sum_{i=1}^{\mathrm{N_s}} (\epsilon\|\mathbf{u}_i\|)^2}{\sum_{i=1}^{\mathrm{N_s}} \|\mathbf{u}_i\|^2} = \epsilon^2 .$$

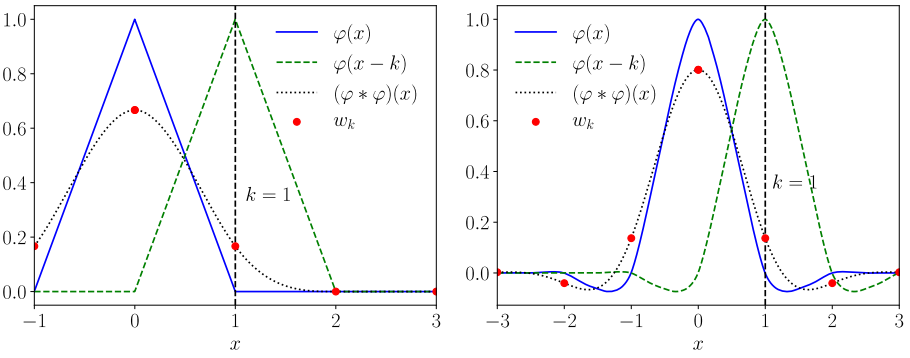


Fig. 18 Autocorrelation functions of Deslauriers interpolating scaling functions φ of order two (left) and order four (right)

The second term in Eq. (48) can be expressed with the help of the eigenvalues of the correlation matrix. We use the identities: $\sum_{i=1}^{N_s} \|\mathbf{u}_i^\epsilon - \tilde{\mathbf{u}}_i^\epsilon\|^2 = \sum_{k=r+1}^{N_s} \lambda_k^\epsilon$ for perturbed eigenvalues $\lambda_k^\epsilon = \lambda_k + l_k\epsilon$ and $\sum_{i=1}^{N_s} \|\mathbf{u}_i\|^2 = \sum_{k=1}^{N_s} \lambda_k$, yielding

$$\frac{\sum_{i=1}^{N_s} \|\mathbf{u}_i^\epsilon - \tilde{\mathbf{u}}_i^\epsilon\|^2}{\sum_{i=1}^{N_s} \|\mathbf{u}_i\|^2} = \frac{\sum_{k=r+1}^{N_s} (\lambda_k + \epsilon l_k)}{\sum_{k=1}^{N_s} \lambda_k} = \mathcal{E}_{\text{POD}}(r,0) + \mathcal{M}_r \epsilon. \tag{49}$$

Here $\mathcal{M}_r = \sum_{k=r+1}^{N_s} l_k / \sum_{k=1}^{N_s} \lambda_k$ is the *perturbation coefficient* of the total error. Note that the perturbations l_k are caused by the non-vanishing mixed terms $\langle \varphi_\lambda^j \rangle \psi_{\mu,\lambda}^j$ (see also [9]), when computing the correlation matrix from thresholded snapshots u_i^ϵ. Note that for orthogonal wavelets the first-order perturbations would vanish. For slowly decaying eigenvalues λ_k the perturbation coefficient $\mathcal{M}_r$ is typically very small, since the sum of perturbations l_k is small compared to the total energy. In this case it is reasonable to neglect the second term in Eq. (49):

$$\mathcal{E}_{\text{wPOD}} \lesssim \mathcal{E}_{\text{POD}}(r,0) + \epsilon^2 . \tag{50}$$

However, in general $\mathcal{M}_r$ does not vanish and we only have linear convergence in ϵ:

$$\mathcal{E}_{\text{wPOD}} \leq \mathcal{E}_{\text{POD}}(r,0) + \mathcal{M}_r \epsilon + \mathcal{O}(\epsilon^2) . \tag{51}$$

Note that $\mathcal{M}_r$ does not depend on ϵ, as all epsilon dependence has been removed. So it is asymptotically a first-order scheme in ϵ only. For a certain range we can observe second order, if $\mathcal{M}_r$ is sufficiently small. Eventually for ϵ sufficiently small the first-order term will dominate.

Appendix C. Technical details and supplementary material

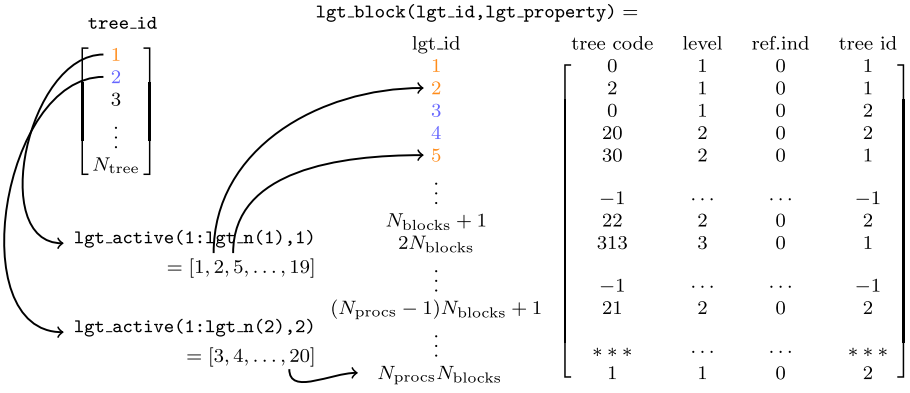


Fig. 19 Example of the light data structure in `WABBIT`, to be compared with Fig. 2. For each tree `lgt_active` stores a light-ID list of all active blocks. With the blocks light-ID (`lgt_id`) all parameters in the forest (tree code, tree ID, tree level, refinement status) can be accessed, from the corresponding row in `lgt_block`. Note that the order of the light-ID depends on the process rank

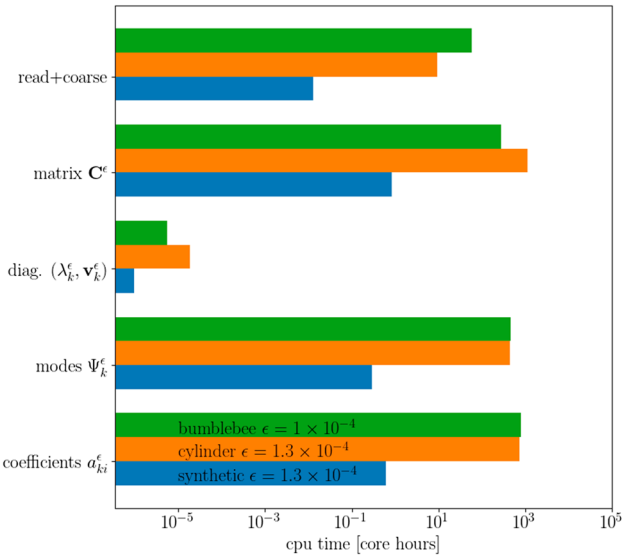


Fig. 20 CPU time required for the individual steps of the wPOD algorithm described in Section 3.3. The CPU time is shown for three different cases: 1.) the synthetic test case from Section 4.1 computed with $J_{\max} = 5$ and $\epsilon = 1.3 \times 10^{-4}$ on a Intel Core i5-7200U cpu (blue), 2.) the flow past a cylinder data Section 4.2.1 computed with $\epsilon = 1.3 \times 10^{-4}$ on Intel Xeon E5645 cpus (orange) and 3.) the bumblebee data of Section 4.2.4 with $\epsilon = 1.3 \times 10^{-4}$ using Intel Xeon Gold 6142 cpus (green). The comparison of the individual cases should be done with care, since they have been computed on different hardware and the data-structure (number of snaphots, blocks, block size, spatial dimension, etc.) is different. Therefore, the computational costs of MPI communication overhead, block administration may vary. However, the overall proportions between the individual steps of the algorithm are comparable among the test cases

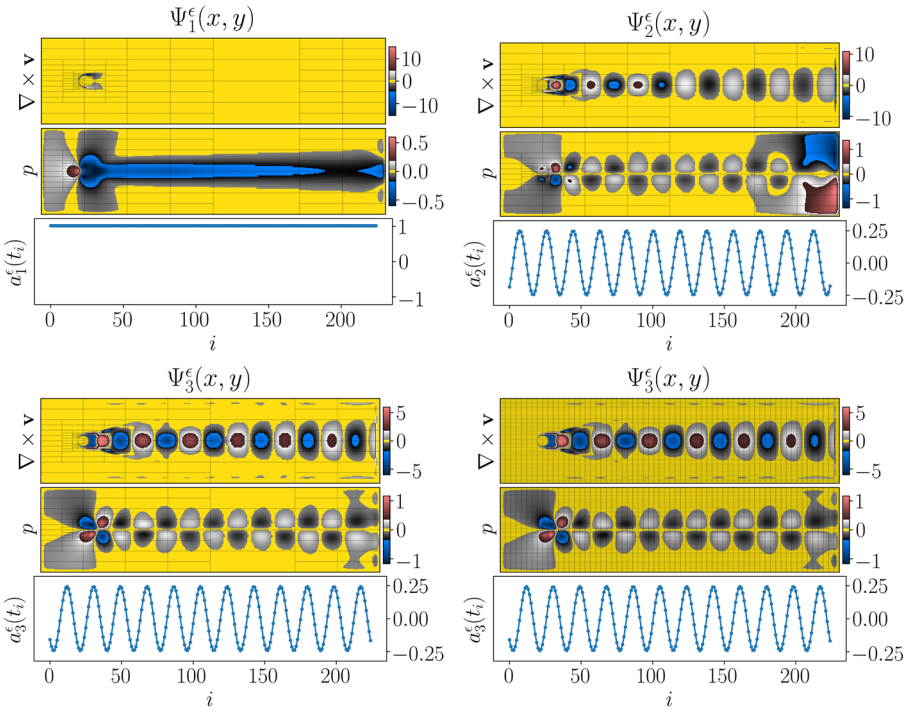


Fig. 21 First three sparse modes $\boldsymbol{\Psi}_k^\epsilon$ ($k = 1$ upper left, $k = 2$ upper right, $k = 3$ bottom left) with their corresponding amplitudes $a_k^\epsilon(t_i)$ computed with $\epsilon = 1.3 \times 10^{-2}$ and one dense ($\epsilon = 0$) mode $\boldsymbol{\Psi}_3^\epsilon$ for comparison (bottom, right). Each figure shows the modes "vorticity" (labeled by $\nabla \times \mathbf{v}$), computed from the two velocity components of the modes, and the pressure component (labeled by p). Note that the first mode represents the base flow, which is non oscillating, whereas the other modes always have an oscillating structure, with a frequency increasing with the mode number. When comparing the dense mode $\boldsymbol{\Psi}_3^\epsilon$ of the non-adaptive case in the lower right of Fig. 21 with the adapted modes on the lower left, no qualitative differences can be observed, except the local changes in the resolution

Acknowledgements The authors gratefully acknowledge the support of the Deutsche Forschungsgemeinschaft (DFG) as part of GRK2433 DAEDALUS. The authors were granted access to the HPC resources of IDRIS under the Allocation No. 2018-91664 attributed by GENCI (Grand Équipement National de Calcul Intensif). Centre de Calcul Intensif d'Aix-Marseille Université is acknowledged for granting access to its high performance computing resources.

Funding Open Access funding enabled and organized by Projekt DEAL.

Code and data availability Software can be found under [17, 49]. All scripts to reproduce the results are available at [18].

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Ali, M., Steih, K., Urban, K.: Reduced basis methods with adaptive snapshot computations. Adv. Comput. Math. **43**(2), 257–294 (2017)
2. Ali, M., Urban, K.: Reduced Basis Exact Error Estimates with Wavelets. In: Numerical Mathematics and Advanced Applications ENUMATH 2015, pp. 359–367. Springer (2016)
3. Alla, A., Kutz, J.N.: Randomized model order reduction. Adv. Comput. Math. **45**(3), 1251–1271 (2019)
4. Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N.: The fenics project. https://fenicsproject.org/, Visited, 12, May (2020)
5. Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N.: The fenics project version 1.5 Archive of Numerical Software 3(100) (2015)
6. Anderson, E., Bai, Z., Dongarra, J., Greenbaum, A., McKenney, A., Du Croz, J., Hammarling, S., Demmel, J., Bischof, C., Sorensen, D.: LAPACK: A portable linear algebra library for high-performance computers. In: Proceedings of the 1990 ACM/IEEE Conference on Supercomputing, Supercomputing '90, pp. 2–11. IEEE Computer Society Press, Los Alamitos, CA, USA (1990)
7. Benner, P., Cohen, A., Ohlberger, M., Willcox, K.: Model reduction and approximation: theory and algorithms, vol. 15 SIAM (2017)
8. Benner, P., Ohlberger, M., Patera, A., Rozza, G., Urban, K.: Model reduction of parametrized systems Springer (2017)
9. Castrillon-Candas, J.E., Amaratunga, K.: Fast estimation of continuous karhunen-loeve eigenfunctions using wavelets. IEEE Trans. Signal Process. **50**(1), 78–86 (2002)
10. Cohen, A., Daubechies, I., Feauveau, J.C.: Biorthogonal bases of compactly supported wavelets. Comm. Pure and Appl. Math. **45**, 485–560 (1992)
11. Deslauriers, G., Dubuc, S.: Interpolation dyadique École polytechnique de Montréal (1987)
12. Deslauriers, G., Dubuc, S.A.: Saff DeVore Symmetric Iterative Interpolation Processes. In: Ronald, E.B. (ed.) Constructive Approximation: Special Issue: Fractal Approximation, pp. 49-68. Springer US, Boston, MA (1989)
13. DeVore, R.A.: Nonlinear approximation. Acta Numerica **7**, 51–150 (1998)
14. Domingues, M., Gomes, S., Díaz, L.: Adaptive wavelet representation and differentiation on block-structured grids. Appl. Numer. Math. **47**(3), 421–437 (2003)
15. Donoho, D.L.: Interpolating wavelet transforms. Department of Statistics, Stanford University (1992)
16. Eckart, C., Young, G.: The approximation of one matrix by another of lower rank. Psychometrika **1**(3), 211–218 (1936)
17. Engels, T., Krah, P.: Python tools for the wavelet adaptive block-based solver for interactions in turbulence. https://github.com/adaptive-cfd/python-tools, Visited 9 (2020)
18. Engels, T., Krah, P.: Scripts used for this publication. https://github.com/adaptive-cfd/WABBIT-convergence-test.git. Visited 9 (2020)
19. Engels, T., Schneider, K., Reiss, J., Farge, M.: A wavelet-adaptive method for multiscale simulation of turbulent flows in flying insects. Commun. Comput. Phys. **30**(4), 1118–1149 (2021)
20. Fang, F., Pain, C., Navon, I., Piggott, M., Gorman, G., Allison, P., Goddard, A.: Reduced-order modeling of an adaptive mesh ocean model. International journal for numerical methods in fluids **59**(8), 827–851 (2009)
21. Farge, M., Pellegrino, G., Schneider, K.: Coherent vortex extraction in 3D turbulent flows using orthogonal wavelets. Phys. Rev. Lett. **87**(054), 501 (2001)
22. Farge, M., Schneider, K., Kevlahan, N.: Non-Gaussianity and coherent vortex simulation for two-dimensional turbulence using an adaptive orthogonal wavelet basis. Phys. Fluids **11**(8), 2187–2201 (1999)

23. Futatani, S., Bos, W., del Castillo-Negrete, D., Schneider, K., Benkadda, S., Farge, M.: Coherent vorticity extraction in resistive drift-wave turbulence: Comparison of orthogonal wavelets versus proper orthogonal decomposition. Comptes Rendus Physique **12**(2), 123–131 (2011)
24. Gräßle, C., Hinze, M.: POD Reduced-order modeling for evolution equations utilizing arbitrary finite element discretizations. Adv. Comput. Math. **44**(6), 1941–1978 (2018)
25. Gräßle, C., Hinze, M., Lang, J., Ullmann, S.: POD Model order reduction with space-adapted snapshots for incompressible flows. Adv. Comput. Math. **45**(5-6), 2401–2428 (2019)
26. Greif, C., Urban, K.: Decay of the Kolmogorov n-width for wave problems. Appl. Math. Lett. **96**, 216–222 (2019)
27. Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. **53**(2), 217–288 (2011)
28. Harten, A.: Discrete multi-resolution analysis and generalized wavelets. Applied Numerical Mathematics **12**(1), 153–192 (1993)
29. Harten, A.: Multiresolution representation of data: a general framework. SIAM J. Numer. Anal. **33**(3), 1205–1256 (1996)
30. Harten, A.: Multiresolution Representation and Numerical Algorithms: a Brief Review, pp. 289–322. Springer, Netherlands, Dordrecht (1997)
31. Holmes, P., Lumley, J.L., Berkooz, G., Rowley, C.W.: Turbulence, coherent structures, dynamical systems and symmetry. Cambridge University Press (2012)
32. Karatzas, E.N., Ballarin, F., Rozza, G.: Projection-based reduced order models for a cut finite element method in parametrized domains. Computers & Mathematics with Applications **79**(3), 833–851 (2020)
33. Läuchli, P.: Jordan-Elimination und Ausgleichung nach kleinsten Quadraten. Numer. Math. **3**(1), 226–240 (1961)
34. Mahoney, M.W.: Randomized algorithms for matrices and data. Found. Trends Mach. Learn. **3**(2), 123–224 (2011)
35. Mallat, S.: A Wavelet Tour of Signal Processing, Third Edition Edn. Academic Press, Boston (2009)
36. Mallat, S.G.: Multiresolution approximations and wavelet orthonormal bases of $L^2(\mathbb{R})$. Transactions of the American Mathematical Society **315**(1), 69–87 (1989)
37. Massing, A., Larson, M.G., Logg, A.: Efficient implementation of finite element methods on nonmatching and overlapping meshes in three dimensions. SIAM J. Sci. Comput. **35**(1), C23–C47 (2013)
38. Mendez, M., Balabane, M., Buchlin, J.M.: Multi-scale proper orthogonal decomposition of complex fluid flows. arXiv:1804.09646 (2018)
39. Nguyen van yen, R.: Wavelet-based study of dissipation in plasma and fluid flows. Ph.D. thesis Université Paris-Sud XI (2010)
40. Ohlberger, M., Rave, S.: Reduced basis methods: Success, limitations and future challenges. Proceedings of the Conference Algoritmy, pp 1–12 (2016)
41. Reiss, J., Schulze, P., Sesterhenn, J., Mehrmann, V.: The shifted proper orthogonal decomposition: a mode decomposition for multiple transport phenomena. SIAM J. Sci. Comput. **40**(3), A1322–A1344 (2018)
42. Roussel, O., Schneider, K.: Coherent vortex simulation of weakly compressible turbulent mixing layers using adaptive multiresolution methods. J. Comput. Phys. **229**(6), 2267–2286 (2010)
43. Sakurai, T., Yoshimatsu, K., Schneider, K., Farge, M., Morishita, K., Ishihara, T.: Coherent structure extraction in turbulent channel flow using boundary adapted wavelets. J. Turbul. **18**(4), 352–372 (2017)
44. Schmid, P.J.: Dynamic mode decomposition of numerical and experimental data. J. Fluid Mech. **656**, 5–28 (2010)
45. Schneider, K., Vasilyev, O.V.: Wavelet methods in computational fluid dynamics. Annu. Rev. Fluid Mech. **42**, 473–503 (2010)
46. Sieber, M., Paschereit, C.O., Oberleithner, K.: Spectral proper orthogonal decomposition. J. Fluid Mech. **792**, 798–828 (2016)
47. Sirovich, L.: Turbulence and the dynamics of coherent structures. part i-iii. Q. Appl. Math. **45**(3), 561–571 (1987)
48. Sroka, M., Engels, T., Krah, P., Mutzel, S., Schneider, K., Reiss, J.: An Open and Parallel Multiresolution Framework Using Block-Based Adaptive Grids. In: Active Flow and Combustion Control 2018, pp. 305–319. Springer (2019)

49. Sroka, M., Engels, T., Mutzel, S., Krah, P., Reiss, J.: Wavelet adaptive block-based solver for interactions in turbulence. https://github.com/adaptive-cfd/{WABBIT}. Visited 9 (2020)
50. Sweldens, W., Schröder, P.: Building your own wavelets at home Wavelets in Computer Graphics (1997)
51. Ullmann, S., Rotkvic, M., Lang, J.: POD-Galerkin reduced-order modeling with adaptive finite element snapshots. J. Comput. Phys. **325**, 244–258 (2016)
52. Unser, M.: Approximation power of biorthogonal wavelet expansions. IEEE Trans. Signal Process. **44**(3), 519–527 (1996)
53. Uytterhoeven, G., Roose, D.: Experiments with a wavelet-based approximate proper orthogonal decomposition. Katholieke Universiteit Leuven Departement Computerwetenschappen (1997)
54. Volkwein, S.: Optimal control of a phase-field model using proper orthogonal decomposition. ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik: **81**(2), 83–97 (2001)
55. Volkwein, S.: Model reduction using proper orthogonal decomposition. Lecture Notes, Institute of Mathematics and Scientific Computing. University of Graz. see http://www.uni-graz.at/imawww/volkwein/POD.pdf, p 1025 (2011)
56. Yu, D., Chakravorty, S.: A Randomized Proper Orthogonal Decomposition Technique. In: 2015 American Control Conference (ACC), pp. 1137–1142 (2015)
57. Zumbusch, G.: Parallel multilevel methods: adaptive mesh refinement and loadbalancing Springer Science & Business Media (2012)

Affiliations

Philipp Krah[1] · Thomas Engels[2] · Kai Schneider[3] · Julius Reiss[4]

Thomas Engels
thomas.engels@uni-rostock.de

Kai Schneider
kai.schneider@univ-amu.fr

Julius Reiss
julius.reiss@tnt.tu-berlin.de

1 Technische Universität Berlin, Institute of Mathematics, Straße des 17. Juni 136, 10623 Berlin, Germany

2 Department of Animal Physiology, Institute of Biological Sciences, University of Rostock, Albert-Einstein-Str. 3, Rostock 18059, Germany

3 Aix-Marseille Université, CNRS, Centrale Marseille, Institut de Mathématiques de Marseille (I2M), 39 rue Joliot-Curie, Marseille, 13453 Cedex 13, France

4 Institute of Fluid Mechanics and Engineering Acoustics, Technische Universität Berlin, Müller-Breslau-Str. 15, 10623 Berlin, Germany