Technische Universität München

Lehrstuhl für Informatik mit Schwerpunkt
Wissenschaftliches Rechnen

Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques

Benjamin Peherstorfer

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 25.06.2013 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 26.08.2013 angenommen.

Abstract

The complexity of today's models and simulations in computational science and engineering has dramatically increased and clearly outpaced the boost in computing power. To cope with this increase in complexity, model order reduction methods construct low-cost surrogates of large-scale simulations by incorporating data or additional knowledge of the current problem at hand. Thus, the problem is not solved in a general, high-dimensional solution space but in a problem-dependent, low-dimensional subspace, which, at least approximately, includes the solution. The challenge is now to efficiently utilize additional knowledge of the problem to find the most appropriate subspace.

We propose to consider model reduction as a learning task – from available data or knowledge, we want to learn a good subspace for the current problem at hand. Therefore, we first introduce a new hierarchy or classification of model reduction techniques which reflects this strong relationship with learning. Second, we tackle the learning problems with new sparse grid techniques. For example, we introduce a novel sparse grid multigrid method, we present the first clustering method based on sparse grids, and we discuss an Offline/Online strategy to solve the classification problem up to 300 times faster than before. Third, because we consider model reduction as a learning task, we can now employ learning methods such as our sparse grid techniques in the context of model reduction. This gives us a variety of new methods in each class of our model reduction hierarchy.

We demonstrate the advantages of the new model reduction methods on the basis of various engineering applications reaching from thermal conduction via car crash to chemical reaction simulations, and we show that our sparse grid learning techniques enable us to tackle numerous classical learning problems such as image segmentation and the analysis of flow simulation data which could not be treated with sparse grids before.

Acknowledgments

First and foremost I would like to thank Hans-Joachim Bungartz for allowing me to work in his research group. Only because of Hans' anarchic style of leading the group, I was able to freely pursue thoughts, ideas, and projects. Besides research, I profited so much from the countless things I learned while working with him on a day-to-day basis. I'm particularly indebted to him for his continuous stream of opportunities (SPPEXA, in.tum.quarterly, to name just a few).

One of these opportunities allowed me to closely collaborate with Karen Willcox who turned out to fundamentally shape my research. I still can hardly believe how lucky I am to work with her on some of the most exciting topics of model order reduction and machine learning. And I cannot express how honored I felt as she accepted to review my thesis; thank you so much!

Next, I have to thank Prof Zenger for the many vivid and motivating discussions on sparse grids and multigrid methods. I'm almost convinced now that binary computers were a bad decision and that ternary logic would have made many algorithms easier to implement (at least the Peano curve). Unfortunately, Stefan Zimmer left TUM the day I started my PhD (a coincidence?); however, the scientific (and hiking) collaboration continued through regular visits from which I profited a lot. The same holds for Dirk Pflüger, who developed not only the best sparse grid library but who is also the best proof reader I have ever met; every typo you do *not* find in this thesis is due to Dirk.

I also would like to thank my colleagues (in particular Christoph Riesinger, Martin Schreiber, and Daniel Butnaru) for creating such a friendly, open, and collaborative environment as we have in our research group. Finally, I have to thank my students who contributed a lot to this thesis. I do not attempt to list all of them here, but I certainly have to name Emily Mo-Hellenbrand, Julius Adorf, Fabian Franzelin, Zhongwen Song, and Florian Zacherl.

Contents

1. Introduction

Today, numerical simulations have become at least as important as experiments. Such computer simulations allow researchers to investigate problems that are otherwise either too costly or simply impractical to address. This new discipline called computational science and engineering (CSE) can be seen as the third pillar of 21st century science, besides theoretical analysis and experimentation. It is the combination of these three approaches that led to the great leaps of science in the past decades [70].

Performing an experiment or computer simulation does not directly lead to scientific discovery. At first, the result is only data. Answers to posed scientific questions are found by searching for patterns in the produced data. Some of the greatest discoveries in history are based on this fundamental interaction between collecting and analyzing data. Examples are the empirical laws of planetary motion, or the development and verification of quantum physics [28]. Nowadays, experiments — independent whether they are performed in our physical world or through a computer simulation — produce huge amounts of data that cannot be analyzed anymore by humans alone. With machine learning methods, it is possible to crunch such amounts of data to detect similarities, extreme outliers, and other significant patterns. For example, in astrophysics, computer-based data analysis is starting to play a major role [184]. However, the data produced by computational experiments cannot only be used for scientific discovery but it can also be used to derive more efficient computer models and numerical methods. In this thesis, we employ machine learning methods to analyze simulation data but also to derive low-cost surrogates of large-scale simulations.

A common task in science and engineering is to evaluate input-output problems where the output of interest is computed from a solution of an input-parametrized partial differential equation (PDE). These problems can be solved with classical numerical techniques such as the finite element method, the finite difference method, or the finite volume method. However, this quickly exhausts today's computers because either very high accuracies are required or, even more common, the input-output problem has to be solved many times in a row as, for example, in the optimization, uncertainty quantification, or statistical inverse contexts. Therefore, the aim is to keep the computational effort to solve the input-output problem low:

1. A first step towards reducing the computational costs is to discretize the PDE not on an equidistant grid with equal mesh width in all directions, but to employ *a priori* grid optimization to derive grids and spaces that are specially constructed for certain problem classes [171, 195]. If the current problem at hand belongs to such a problem class where an *a priori* optimized grid is available and the underlying PDE is discretized on the optimized grid, then fewer grid points (degrees of freedom) are sufficient to obtain a solution with similar accuracies as the classical solution. This leads to a reduction of the complexity of the solution process. That is what we call *a priori* model order reduction.

2. The natural next step is to derive solution spaces not for whole problem classes but only for the particular problem of interest [170, 139]. To derive a space that is

tailored to an input-output problem, the characteristic relationship between inputs and outputs is learned from data. This data is obtained in a pre-processing step by solving the problem multiple times for different inputs in the classical way. We call this *a posteriori* model order reduction because first the data must be produced from which then the problem-dependent space is derived.

3. To continue this process, we do not only construct one problem-dependent space but multiple local spaces, each of them tailored to a specific system behavior. This can be achieved by analyzing the solution data to discover distinct system behaviors and to learn from it to construct these multiple local solution spaces. From these local spaces, an appropriate one is selected for the final approximation. We call this *post analysis* model order reduction.

These three model reduction types — *a priori*, *a posteriori*, and *post analysis* — form our reduced-order model hierarchy. We refer to, e.g., [63, 73] for a more common hierarchy consisting of projection-based, data-fit, and physics-based reduced models.

Our reduced-order model hierarchy reflects the close relationship between model order reduction and machine learning. In fact, model order reduction is nothing else than a learning problem — from the knowledge about the problem, a specific solution space is derived leading to a reduction of the solution complexity for the current problem or problem class. In case of *a priori* model reduction, this relationship is not so clear because no data is available. Still, from the knowledge of the problem class, the *a priori* grid optimization is performed. The other two model reduction types, i.e., *a posteriori* and *post analysis* model order reduction, heavily rely on data to gather information about the problem and thus are learning problems in the classical sense. *A posteriori* model order reduction can be stated as a supervised learning where the relationship between data points (parameters) and target values (outputs of interest) is learned. *Post analysis* model reduction is not only related to supervised learning but also to unsupervised learning where only the data points (solutions) are given and the hidden structure (system behavior) has to be discovered to construct multiple local reduced models.

In this thesis, we focus on grid-based methods to tackle supervised and unsupervised learning problems. Conventional learning methods represent functions as linear combinations of basis functions centered at the given data points. This automatically adapts the function representation to the data. However, the computational complexity of such a function representation grows if the number of data points is increased. Since these data-based learning methods typically scale quadratically or even worse with the number of data points, additional approximations are required to tackle large data sets. By discretizing functions as linear combination of basis functions centered at grid points rather than at the data points, we obtain learning algorithms that scale only linearly in the number of data points. To cope with the curse of dimensionality, i.e., the exponential increase of the number of grid points with the dimension of the data points, we employ sparse grids [79, 153].

The objective of this thesis is to develop novel sparse-grid-based learning techniques to tackle supervised and unsupervised learning problems as well as to apply these sparse grid learning techniques and other learning methods to *a priori*, *a posteriori*, and *post analysis* model order reduction. The structure of this thesis reflects our reduced-order model hierarchy and the thesis' objectives:

- In Part I we consider *a priori* model order reduction where the underlying PDE is discretized on a sparse grid. We present a novel multigrid method to efficiently solve the discretized system and show results for a multi-dimensional convection-diffusion problem.

- In Part II we present sparse grid methods to tackle supervised learning problems. We introduce a classification method based on sparse grid density estimation which allows us to split the learning procedure into a costly Offline (pre-processing) and a rapid Online phase (prediction). In addition, we show how to employ ensembles of sparse grid classifiers and demonstrate that this approach is well-suited for modern hardware architectures. We then employ the sparse grid methods to derive *a posteriori* reduced-order models for thermal conduction problems and the simulation of a heat shield.

- Finally, in Part III we develop two sparse-grid-based clustering (unsupervised learning) methods — to the author's knowledge the first clustering methods based on sparse grids — and employ them to derive *post analysis* reduced-order models for the analysis of car crash data. Furthermore, we consider the simulation of an H_2-Air flame which is a nonlinear problem where *post analysis* model reduction greatly improves the accuracy.

2. Preliminaries

We review several concepts and methods related to data mining, model order reduction, and sparse grids. Of course, this section is not an exhaustive presentation of these topics. Its purpose is rather to provide enough background information for the following sections, to cite literature with more details, and, in particular, to define the notation.

2.1. General Notation of Vectors, Matrices, Sets, and Spaces

We follow the common convention that vectors are bold lower-case letters (e.g., $\boldsymbol{a}$) and that matrices are bold upper-case letters (e.g., $\boldsymbol{A}$). A matrix denoted as $\boldsymbol{A} = [\boldsymbol{a}_1, \dots, \boldsymbol{a}_M]$ has the column vectors $\boldsymbol{a}_1, \dots, \boldsymbol{a}_M$. The i-th component of a vector $\boldsymbol{a}$ is denoted with a_i, the element in the i-th row and the j-th column of a matrix $\boldsymbol{A}$ is A_{ij}. The j-th component of the i-th column vector in a matrix $\boldsymbol{A} = [\boldsymbol{a}_1, \dots, \boldsymbol{a}_M]$ might be denoted with $a_{i,j}$, but, to avoid confusion, we always indicate explicitly when we use this subtle notation. We denote spaces and sets with calligraphic letters (e.g., $\mathcal{S}, \mathcal{V}, \mathcal{D}$) except for $\mathcal{N} \in \mathbb{R}$ which is a scalar value, and the mixed Sobolev space H^2_{mix} which is not denoted as a calligraphic letter even though it is a space. Further notation is introduced where necessary.

2.2. Learning from Data

Data mining or knowledge discovery is the process of searching for patterns, regularities, and irregularities in data. It is a very broad task involving many steps reaching from the initial data pre-processing to the final interpretation of the results [65]. We are only interested in the learning step where systems are trained on data samples to generalize to new, unseen samples. In this section, we briefly review supervised and unsupervised learning problems, and list a few learning methods which are used in the following sections.

2.2.1. Supervised Learning

Supervised learning constructs a function to map data points onto target values. We define the supervised learning problem, introduce classification and regression as the two major supervised learning tasks, discuss non-probabilistic and probabilistic approaches, and list a few supervised learning methods.

Classification and regression Let $c : \mathbb{R}^d \to \mathbb{R}$ be an unknown function that has been sampled at the data points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\} \subset \mathbb{R}^d$ leading to the target values $\{y_1, \dots, y_M\} \subset \mathbb{R}$. Assume the sampling process was disturbed by noise. The unsupervised learning task is to find a function $\hat{c} : \mathbb{R}^d \to \mathbb{R}$ approximating the unknown function c to make predictions on before unseen samples, i.e., the function $\hat{c}$ should generalize to new data. The function $\hat{c}$ is computed by considering only the training data set

$$\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^M \subset \mathbb{R}^d \times \mathbb{R}\,.$$

We do not want to interpolate the data in $\mathcal{S}$ because we are not interested in exactly matching the training data. It is more important that $\hat{c}$ generalizes well to new data samples. We have a classification problem with k classes if $\{y_1, \dots, y_M\} \subseteq \{1, \dots, k\}$ and a regression problem if $\{y_1, \dots, y_M\} \subset \mathbb{R}$. We distinguish between binary classification problems where $k = 2$ and multi-class classification problems with $k > 2$. In the context of classification problems, we call the function $\hat{c}$ a classifier.

We can think of many real-world supervised learning problems [136]. For example, with document classification, email messages can be filtered for spam and junk emails [102]. Another example is handwriting recognition where small images have to be classified into $\{0, 1, \dots, 9\}$ [136]. But supervised learning problems are also found in science, in particular, in life sciences where, for example, biochemical screenings are replaced with classifiers [176]. We will consider such a problem in Sec. 7.3.

If we have a classification problem, the quality of a classifier $\hat{c}$ can be quantified by the accuracy

$$\frac{1}{M_{\mathcal{T}}} \sum_{i=1}^{M_{\mathcal{T}}} I(\hat{c}(\boldsymbol{x}_i') = y_i') \tag{1}$$

on a test set $\mathcal{T} = \{(\boldsymbol{x}_i', y_i')\}_{i=1}^{M_{\mathcal{T}}}$ where I is one if the classifier $\hat{c}$ at $\boldsymbol{x}_i'$ predicts the correct class y_i' and else zero. In case of a regression problem, we can compute the average L_2 or L_∞ error. Because the classifier should be tested on unseen data, the test set is not used to train the classifier.

Usually, the classifier $\hat{c}$ depends on parameters which have to be tuned to the data at hand with the aim to obtain the best performance on new data. For that purpose, it is common to employ n-fold cross validation where the training data set is split into n equally sized subsets [28]. The classifier is then trained for different parameters on the union of $n-1$ subsets and validated on the remaining set with, for example, the accuracy (1). This process is repeated such that each subset becomes the validation set once. Those parameters are selected for which the classifier performs best on the validation data sets. To finally assess the quality of the classifier $\hat{c}$ on new, unseen data, the accuracy (1) can be computed for a test data set if available for the given problem.

Probabilistic models So far, we considered classification from the classical point of view where the training data set $\mathcal{S}$ is given and the classifier $\hat{c}$ is constructed to predict the label y for a new, unseen data point $\boldsymbol{x} \in \mathbb{R}^d$ by $\hat{c}(\boldsymbol{x}) = y$. Let us now consider classification from the Bayesian point of view [136]. Given are the samples $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\}$ of a random variable X, the labels $\{y_1, \dots, y_M\}$ of a random variable Y, and the joint probability density function p of X and Y. Probabilistic models do not directly determine the label $y \in \{1, \dots, k\}$ of a data point $\boldsymbol{x} \in \mathbb{R}^d$ but first estimate the posterior class probabilities $p(Y = i|X = \boldsymbol{x})$ for each class $i \in \{1, \dots, k\}$ and then decide how to label the data point $\boldsymbol{x}$ [28]. The posterior class probability $p(Y = i|X = \boldsymbol{x})$ is the probability that data point $\boldsymbol{x}$ has label $i \in \{1, \dots, k\}$. There are two approaches to compute these probabilities. Discriminative models directly compute $\hat{p}(Y = i|\boldsymbol{x})$ to estimate the posterior class probability function $p(Y = i|\boldsymbol{x})$ for each class $i \in \{1, \dots, k\}$.

Generative models construct $\hat{p}(\boldsymbol{x}|Y = i)$ to estimate the class-conditional probability functions $p(\boldsymbol{x}|Y = i)$ and then compute for a data point $\boldsymbol{x} \in \mathbb{R}^d$ with Bayes' theorem

$$\hat{p}(Y = i|X = \boldsymbol{x}) = \frac{\hat{p}(X = \boldsymbol{x}|Y = i)\hat{p}(Y = i)}{\hat{p}(X = \boldsymbol{x})} \tag{2}$$

the posterior class probabilities, where $\hat{p}(Y = i)$ is the estimated prior class probability. To simplify the notation, we follow [28] and introduce a random variable Y_i for each class $i \in \{1, \dots, k\}$, and write (2) as

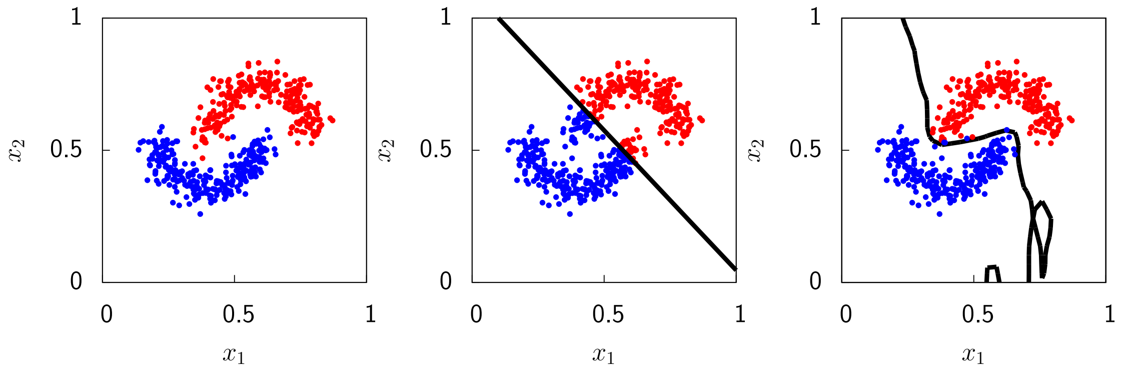
$$\hat{p}(Y_i|\boldsymbol{x}) = \frac{\hat{p}(\boldsymbol{x}|Y_i)\hat{p}(Y_i)}{\hat{p}(\boldsymbol{x})}. \tag{3}$$

Models estimating the class-conditional densities $p(\boldsymbol{x}|Y_i)$ are called generative models because synthetic data can be generated by sampling the estimated probability function $\hat{p}(\boldsymbol{x}|Y_i)$. In the following, we employ probabilistic and non-probabilistic models, as well as generative and discriminative probabilistic models. We refer to [115, 136, 28] for more details on the Bayesian statistics point of view on classification in general and for more details on discriminative and generative models in particular.

Supervised learning methods There is a wide range of supervised learning methods, see, e.g., [102, 28, 165, 136]. Amongst others, we employ in the following nearest neighbor classifiers where a new point is classified according to the majority class membership of its nearest training data points [102]. If the training data set is small, we can simply iterative over all training data points to find the nearest neighbors. However, if the training data becomes large, it is worthwhile to first construct a search structure in a pre-processing step to find nearest neighbors without an exhaustive search of the training data [28].

We also consider support vector machines (SVM) which map the training data into a high-dimensional space where they can be separated with hyperplanes even though the data might not be linearly separable in the original space. Support vector machines rely heavily on the concept of kernels $K : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$. These are functions to which an inner product space $\mathcal{V}$ is associated such that we can write $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (g(\boldsymbol{x}_i), g(\boldsymbol{x}_j))_{\mathcal{V}}$ where $g : \mathcal{S} \to \mathcal{V}$ is a function and $(\cdot, \cdot)_{\mathcal{V}}$ is the inner product of $\mathcal{V}$. This means that if the data points enter the learning algorithm only through the inner product $(\cdot, \cdot)_{\mathcal{V}}$, we can in principle consider them in the possibly very high-dimensional space $\mathcal{V}$ without explicitly constructing the representation in $\mathcal{V}$. The advantage is that in a high-dimensional space $\mathcal{V}$ the data points can be represented such that they become separable by a hyperplane even though this might not be possible in the original space. We refer to [165] for an extensive study of these kernel properties and the support vector machines, and to [110] for a widely-used implementation.

In the following, we also frequently use the sparse-grid-based classification and regression method which will be introduced in Sec. 2.4.2. It discretizes the classifier $\hat{c} : \mathbb{R}^d \to \{1, \dots, k\}$ on a sparse grid and thus achieves a computational procedure which scales only linearly with the number of data points. Thus, it is well-suited for large data sets.



(a) two moons data (b) k-means (c) spectral clustering

Figure 1: The two moons data set consists of two interwoven half moons. The k-means method fails because a straight line cannot separate the two moons, see (b). Spectral clustering in (c) is able to find clusters with non-convex shape and curvilinear boundaries leading to a nearly perfect clustering of the two moons. Note that the cluster boundary in (c) near the right bottom corner of the domain is unspecified because no information is given in this region.

2.2.2. Unsupervised Learning

Training data of unsupervised learning problems contains only data points and no target values. The aim is to find the hidden structures in this data. There are three major unsupervised learning tasks, namely density estimation, clustering, and dimensionality reduction. We briefly discuss each of them.

Unsupervised learning tasks Given is the training data set

$$\mathcal{S} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\} \subset \mathbb{R}^d \,.$$

Note that it does not contain any target values. We distinguish between three unsupervised learning tasks. Density estimation determines the probability density function of the distribution of the data $\mathcal{S}$, clustering divides the data $\mathcal{S}$ into groups of similar data points with respect to a certain similarity measure, and dimensionality reduction finds a low-dimensional embedding of $\mathcal{S}$ which reflects the structure of the training data with respect to a certain criterion [28]. Density estimation is the most fundamental one and it stands out from the other two tasks, as it can be clearly defined without the context of the application. Clustering and dimensionality reduction require specific and problem-dependent criteria, and it is hard to assess the quality of the result without taking the context into account [180].

Convex and non-convex clustering methods Assume we want to cluster the two moons data set as shown in Fig. 1a into two clusters and suppose the similarity between two data points is their Euclidean distance. We then expect a clustering method to separate the two moons.

The widely-used k-means clustering method partitions the set $\mathcal{S}$ into $\mathcal{S}_1, \dots, \mathcal{S}_k$ such that the within-cluster sum of squares is minimized, i.e., the partition $\mathcal{S}_1, \dots, \mathcal{S}_k$ is the result of the optimization problem

$$\operatorname*{arg\,min}_{\mathcal{S}_1,\dots,\mathcal{S}_k} \sum_{i=1}^{k} \sum_{\boldsymbol{x}_j \in \mathcal{S}_i} \|\boldsymbol{x}_j - \boldsymbol{c}_i\|^2 \tag{4}$$

where $\boldsymbol{c}_1, \dots, \boldsymbol{c}_k \in \mathbb{R}^d$ are the means of the points in $\mathcal{S}_1, \dots, \mathcal{S}_k$, respectively. We also call $\boldsymbol{c}_i$ the center of the i-th cluster. The optimization problem (4) is usually solved with Lloyd's algorithm [28]. A fundamental problem of k-means clustering is that it is a convex clustering method and thus can only detect clusters with a convex shape with respect to the similarity measure. This means, the two moons cannot be separated with k-means clustering, see Fig. 1b.

So-called non-convex (or nonlinear) clustering methods are required to separate the two moons. Non-convex methods, e.g., spectral clustering and density-based clustering, are able to find clusters of arbitrary shape and thus can separate the two moons, see Fig. 1c. Spectral clustering represents the data as a similarity graph and uses the eigenvectors of a modified adjacency matrix of the graph to derive the clustering [179]. We extensively deal with spectral clustering in Sec. 10.1. Density-based methods define a cluster as a dense region ("where many data points are") surrounded by a region of low-density ("where few data points are") and find these high- and low-density regions by estimating the density function of the data [64]. We discuss a density-based method with sparse grids in Sec. 10.2.

All these methods have in common that they assign exactly one cluster label to each data point in $\mathcal{S}$. Even though not of relevance for us, we want to mention that clustering can also be put into a probabilistic setting where the data points have multiple labels each associated with a certain probability [28].

Clustering validation Having obtained a clustering of the data points in $\mathcal{S}$ with a clustering method, we want to validate the quality of the cluster assignment. If the true clustering is known, we can employ so-called external measures to compare the two cluster assignments. If external information is not available, structural properties such as compactness and separation of the clusters are determined with so-called internal measures [127].

Let us first consider an external measure. The adjusted rand index (ARI) is a measure of the similarity between two cluster assignments [112]. We cannot simply compute the accuracy (1) as in supervised learning, because we do not know how to match the cluster labels from the true clustering with the cluster labels obtained by the clustering method. Hence, the adjusted rand index does not consider each element on its own but counts correctly clustered pairs of elements. Its range is between -1 and 1 where 1 corresponds to perfect agreement with the true clustering.

Internal measures do not require external information. They assess the quality of a clustering based on certain structural properties, see, e.g., [127]. For example, the score of the so-called Silhouette index [164] is higher when clusters are dense and well separated. Similar properties are measured by the Dunn and Davies-Bouldin index [34, 127]. Unfortunately, all these indices take the distance and cluster shape into account and thus do not perform well on clusters with non-convex shape. That is why in [173] a new internal measure called expected density measure is introduced. It also measures how dense the clusters are but to cope with arbitrary cluster shapes it employs geodesic distances derived from a similarity graph instead of the Euclidean distances.

Linear and nonlinear dimensionality reduction Let $f : \mathbb{R}^{d'} \to \mathbb{R}^d$ be an unknown function with $d' < d$ and assume the data in $\mathcal{S} \subset \mathbb{R}^d$ has been computed by sampling the function f at the low-dimensional data points $\mathcal{S}' = \{\boldsymbol{x}'_1, \dots, \boldsymbol{x}'_M\} \subset \mathbb{R}^{d'}$. The aim of dimensionality reduction is to recover the low-dimensional data $\mathcal{S}'$ from the high-dimensional data $\mathcal{S}$ without the function f. This fits to the more general definition of dimensionality reduction we have given above where we said that the low-dimensional representation $\mathcal{S}'$ has to contain the relevant information of $\mathcal{S}$. Thus, the function f just adds unnecessary information which is not important for our purposes. Again, it is hard to validate the embedding if the context of the application is not known.

With the principal component analysis (PCA) the data in $\mathcal{S}$ is transformed into a space where the first dimension of the representation has the largest possible variance. Thus, it carries as much of the variability of the data as possible [102, 28]. By considering only the first $d' < d$ dimensions of the representation, the number of dimensions can be reduced but the dimensions with high variability remain. This is a linear approach because the embedding into the low-dimensional space is a linear transformation. Thus, the function f has to be (nearly) linear to obtain good results. We defer a detailed discussion to Sec. 2.3.2. Recently, nonlinear dimensionality reduction methods have been developed which can cope with nonlinear functions f as well. Most of them are based on similarity graphs to approximate the low-dimensional, nonlinear manifold induced by the data in the high-dimensional space $\mathbb{R}^d$. We refer to [125] for details and to Sec. 10.1.3 for examples.

2.3. Model Order Reduction of Parametrized Systems

Many simulations in computational science and engineering are input-output problems formulated with a state space. For example, consider a thermal conductivity problem. The inputs are the parameters, e.g., the thermal conductivity coefficient and the thickness of the material, the state is the temperature field, and the output of interest is the average temperature. Model order reduction approximates such simulations with low-cost surrogates by solving the problem not in a general, high-dimensional solution space but in a problem-dependent, low-dimensional subspace. In particular when the simulation has to be repeated many times, e.g., in optimization, uncertainty quantification, or statistical inverse problems, model reduction can lead to tremendous savings in computational complexity and runtime.

In this section, we formulate these input-output problems and their reduced-order models, discuss why model order reduction works, and briefly introduce two common model order reduction methods.

2.3.1. Reduced-Order Models of Parametrized Systems

We introduce input-output problems in a very general setting and define the notation for the full-order and reduced-order model.

Input-output problems Consider the input-output problem $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$ where $\mathcal{D}$ is the parameter domain, $\mathcal{U}$ the state space, and $\mathcal{Y}$ the output space. In most cases, we are only interested in the output $\boldsymbol{y} \in \mathcal{Y}$, and therefore the state $u(\boldsymbol{\mu}) \in \mathcal{U}$ is just a means to compute the output $\boldsymbol{y}$. The state $u(\boldsymbol{\mu}) : \Omega \to \mathbb{R}$ is the solution of a partial differential equation (PDE) with parameter $\boldsymbol{\mu} \in \mathcal{D}$ and spatial domain Ω. To make the dependence on the parameter and the spatial domain explicit, we sometimes write $u : \Omega \times \mathcal{D} \to \mathbb{R}$. We might add the superscript e to clearly distinguish the "exact" problem $s^e : \mathcal{D} \to \mathcal{U}^e \to \mathcal{Y}$ from its discretized counterpart $s^{\mathcal{N}} : \mathcal{D} \to \mathcal{U}^{\mathcal{N}} \to \mathcal{Y}$ where $\mathcal{N}$ is the number of degrees of freedom. The discretized PDE underlying $s^{\mathcal{N}}$ leads to a parametrized system of equations

$$\boldsymbol{A}\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}) + \boldsymbol{F}(\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu})) = 0 \tag{5}$$

with $\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ and where the matrix $\boldsymbol{A} \in \mathbb{R}^{\mathcal{N} \times \mathcal{N}}$ corresponds to the linear operators of the PDE, and the function $\boldsymbol{F} : \mathbb{R}^{\mathcal{N}} \to \mathbb{R}^{\mathcal{N}}$ to the nonlinear operators. Note that (5) becomes a system of ordinary differential equations (ODE) if we have a time-dependent PDE. To evaluate the output function $s^{\mathcal{N}}(\boldsymbol{\mu}) = \boldsymbol{y}$, the system (5) is solved for the parameter $\boldsymbol{\mu} \in \mathcal{D}$ to obtain the state vector $\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ corresponding to the solution function $u^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathcal{U}^{\mathcal{N}}$ from which then the output $\boldsymbol{y} \in \mathcal{Y}$ is determined. We call the input-output problem $s^{\mathcal{N}} : \mathcal{D} \to \mathcal{U}^{\mathcal{N}} \to \mathcal{Y}$ the high-fidelity or the full model, and $\mathcal{N}$ the dimension of the model.

Reduced-order models Let us consider our input-output problem $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$ in the many-query context where we have to evaluate s for many parameters. For each parameter we have to solve the system (5) to obtain the output. This becomes computationally infeasible very soon if the dimension $\mathcal{N}$ of the high-fidelity model is large. We have a similar situation in the real-time context where we need a rapid respond $s(\boldsymbol{\mu})$ to a parameter $\boldsymbol{\mu}$. Again, if $\mathcal{N}$ is large, the real-time requirements might not be satisfied.

To reduce the computational complexity and thus the runtime of solving our input-output problem, we approximate the high-fidelity model $s^{\mathcal{N}} : \mathcal{D} \to \mathcal{U}^{\mathcal{N}} \to \mathcal{Y}$ with a reduced-order model $s_n : \mathcal{D} \to \mathcal{U}_n \to \mathcal{Y}$ with only $n \ll \mathcal{N}$ degrees of freedom. The reduced space $\mathcal{U}_n$ is spanned by the so-called reduced basis. In principle, there are two points of view why model order reduction is able to construct a reduced model with only n degrees of freedom leading to a similar input-output response characteristic as the full model with $\mathcal{N}$ degrees of freedom. First, the state solutions are usually not scattered all over the space $\mathcal{U}^{\mathcal{N}}$ but form a low-dimensional and smooth manifold

$$\mathcal{M} = \{u(\boldsymbol{\mu}) \,|\, \boldsymbol{\mu} \in \mathcal{D}\}.$$

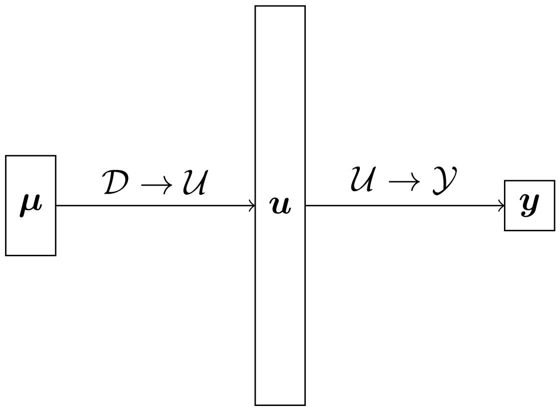


Figure 2: In many cases, the dimension $\mathcal{N}$ of the state space $\mathcal{U}$ can be reduced because a few input parameters $\boldsymbol{\mu}$ lead to a very large and costly state vector $\boldsymbol{u}$ which is in the end reduced to a small number of outputs $\boldsymbol{y}$. This suggests that the state vector contains much more information than is required to compute the output of interest.

Therefore, it is distinctly cheaper to represent a solution $u(\boldsymbol{\mu})$ as an element of span$\{\mathcal{M}\}$ than as an element of $\mathcal{U}^{\mathcal{N}}$. Thus, the reduced space $\mathcal{U}_n$ should be an approximation of span$\{\mathcal{M}\}$. This is the motivation given in, e.g., [142]. The second point of view directly considers the input-output map $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$ and argues that the map $\mathcal{D} \to \mathcal{Y}$ is often much simpler than the state vector $\boldsymbol{u}$ suggests. This is illustrated in Fig. 2. The few input parameters lead to a very large and costly state vector which is in the end reduced to only a few outputs. This suggests that the state vector contains much more information than is required to compute the output of interest [126].

Usually, model order reduction is split into an Offline and an Online phase. In the Offline phase the reduced-order model is constructed. This might require to solve the high-fidelity model for several parameters which makes the Offline phase computationally expensive. In the Online phase, a rapid response $s(\boldsymbol{\mu})$ to a parameter $\boldsymbol{\mu}$ can be provided by evaluating the reduced-order model instead of the high-fidelity model. It is commonly assumed that the Online phase is repeated many times to compensate for the costly Offline phase. In Part I we discuss *a priori* model order reduction which does not require this Offline/Online splitting.

2.3.2. Selected Model Reduction Techniques

Proper orthogonal decomposition (POD) and the reduced basis method (RBM) are two widely-used model reduction methods. We briefly introduce them and discuss their basic properties.

Model reduction methods In the following, we employ projection-based and data-fit reduced-order models and methods. Data-fit models are generated by directly learning the input-output relationship described by s from data with classical learning methods. For example, the input-output problem from the parameters in $\mathcal{D}$ to the outputs in

$\mathcal{Y}$ can be learned with supervised learning methods, cf. Sec. 2.2.1 and [118, 17, 16]. Projection-based methods explicitly construct the reduced space $\mathcal{U}_n$ and project onto $\mathcal{U}_n$ rather than onto $\mathcal{U}^{\mathcal{N}}$. We discuss two projection-based methods below.

We can further differentiate between intrusive and non-intrusive methods. Intrusive methods require knowledge about the underlying governing equations to build a solver suited for the reduced-order model. This is a severe drawback of intrusive methods because for many applications the implementation of the solver for the high-fidelity model is already a challenging task. It cannot be expected to spend the same effort on building a solver for the reduced-order model. In contrast, non-intrusive methods only require data which can be obtained from the high-fidelity model solver. But it also has to be noted that in many practical cases non-intrusive methods achieve lower accuracies than intrusive methods which is not surprising because less information from the problem is used.

There are many model reduction methods for parametrized systems, e.g., [142, 26, 170, 83, 66, 135]. For us, proper orthogonal decomposition and the reduced basis method are the two most important ones. We discuss them in the following two paragraphs.

Proper orthogonal decomposition Proper orthogonal decomposition (POD) is a method to construct from a set of snapshots $\mathcal{S} = \{\boldsymbol{u}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}(\boldsymbol{\mu}_M)\} \subset \mathbb{R}^{\mathcal{N}}$ a reduced basis $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\} \subset \mathbb{R}^{\mathcal{N}}$ to span a reduced basis space, see, e.g., [160, 26, 170]. The quality of the reduced space depends on the set of snapshots $\mathcal{S}$. We do not address this issue of sampling the high-fidelity model here, but refer to, e.g., [149, 97, 126, 36].

Let r be the dimension of the space $\operatorname{span}\{\mathcal{S}\}$ spanned by the snapshots in $\mathcal{S}$. The space $\operatorname{span}\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ of dimension $n < r$ spanned by the POD basis vectors $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\} \subset \mathbb{R}^{\mathcal{N}}$ best approximates the space $\operatorname{span}\{\mathcal{S}\}$, i.e., the POD basis solves the minimization problem

$$\min_{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n} \sum_{i=1}^{M} \|\boldsymbol{u}(\boldsymbol{\mu}_i) - \sum_{j=1}^{n} (\boldsymbol{u}(\boldsymbol{\mu}_i)^T \boldsymbol{v}_j)\boldsymbol{v}_j\|_2^2 \,.$$

Note that the POD basis vectors $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ are orthonormal. The basis vectors $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$ are the left singular vectors of the snapshot matrix $\boldsymbol{U} = [\boldsymbol{u}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}(\boldsymbol{\mu}_M)] \in \mathbb{R}^{\mathcal{N} \times M}$ corresponding to the largest singular values. If $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ are the singular values of $\boldsymbol{U}$, the approximation error of the POD basis representation of the snapshots is given by

$$\sum_{j=1}^{M} \|\boldsymbol{u}(\boldsymbol{\mu}_j) - \sum_{i=1}^{n} (\boldsymbol{u}(\boldsymbol{\mu}_j)^T \boldsymbol{v}_i)\boldsymbol{v}_i\|_2^2 = \sum_{i=n+1}^{r} \sigma_i^2 \,. \tag{6}$$

Even tough (6) only holds for the snapshots in $\mathcal{S}$, it is usually a good indicator for the error when representing a state vector that is not in $\mathcal{S}$, see, e.g., [123, 124, 117] and the references therein. POD is the same as PCA, see Sec. 2.2.2, but the name POD is usually used in the context of model reduction whereas PCA is used in the context of machine learning and statistics. We follow this convention here as well.

The POD method provides us with the basis $\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_n\}$. POD is usually combined with Galerkin projection to obtain a reduced solution $\boldsymbol{u}_n(\boldsymbol{\mu}) \in \mathbb{R}^n$ corresponding to $u_n(\boldsymbol{\mu}) \in \mathcal{U}_n$ for a parameter $\boldsymbol{\mu}$. Thus, the system (5) becomes

$$\boldsymbol{V}^T \boldsymbol{A} \boldsymbol{V} \boldsymbol{u}_n(\boldsymbol{\mu}) + \boldsymbol{V}^T \boldsymbol{F}(\boldsymbol{V} \boldsymbol{u}_n(\boldsymbol{\mu})) = 0 \tag{7}$$

where now $\boldsymbol{u}_n(\boldsymbol{\mu}) \in \mathbb{R}^n$ has only n components. The columns of the matrix $\boldsymbol{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_n] \in \mathbb{R}^{\mathcal{N} \times n}$ are the POD basis vectors. If $n \ll \mathcal{N}$ holds, then the system (7) is distinctly faster to solve than the original system (5), even though the nonlinear term $\boldsymbol{F}$ might become problematic, see Sec. 12. The Galerkin projection renders the POD approach into an intrusive method. Note, however, that the reduced solution $\boldsymbol{u}_n(\boldsymbol{\mu}) = \sum_i \beta_i(\boldsymbol{\mu}) \boldsymbol{v}_i$ can also be obtained by computing the coefficients $\boldsymbol{\beta}(\boldsymbol{\mu}) = [\beta_1(\boldsymbol{\mu}), \dots, \beta_n(\boldsymbol{\mu})]^T \in \mathbb{R}^n$ with supervised learning methods leading to a non-intrusive method, see Sec. 8.2.

Reduced Basis Method The drawback of POD is that M solutions of the high-fidelity model have to be computed. Furthermore, the SVD of the $\mathcal{N} \times M$ snapshot matrix might become computationally expensive if $\mathcal{N}$ or M is large. The reduced basis method (RBM) follows a different approach and constructs the basis of the reduced space $\mathcal{U}_n$ directly from the solutions of the high-fidelity model [142]. This means, the state vector $\boldsymbol{u}_n(\boldsymbol{\mu}) \in \mathbb{R}^n$ corresponding to the reduced solution $u_n(\boldsymbol{\mu}) \in \mathcal{U}_n$ is a linear combination

$$\boldsymbol{u}_n(\boldsymbol{\mu}) = \sum_{i=1}^{n} \beta_i \boldsymbol{\zeta}_i \tag{8}$$

of the orthogonal basis $\{\boldsymbol{\zeta}_1, \dots, \boldsymbol{\zeta}_n\}$ which is obtained by orthogonalizing the solutions $\{\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}_n)\}$ corresponding to specific parameters $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_n\}$, see [142, 83] and the references therein.

The linear combination (8) gives a very general idea of RBM. To obtain an efficient and practical method, we additional assume in the following that a rigorous, sharp, and efficient *a posteriori* error estimator $\xi_n : \mathcal{D} \to [0, \infty)$ for a reduced basis space $\mathcal{U}_n$ exists [142]. The estimator $\xi_n(\boldsymbol{\mu})$ gives an upper bound for the error $\|u^{\mathcal{N}}(\boldsymbol{\mu}) - u_n(\boldsymbol{\mu})\|$ of the reduced solution $u_n(\boldsymbol{\mu}) \in \mathcal{U}_n$ with respect to the high-fidelity solution $u^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathcal{U}^{\mathcal{N}}$ in a certain norm $\|\cdot\|$. An error estimator is rigorous if it is valid for all parameters and all dimensions $n \in \mathbb{N}$. Furthermore, it has to yield an error bound and not just an indicator. We call an error estimator sharp if it does not yield a completely over estimated bound but matches the error closely. And, finally, an error estimator is efficient in the context of the reduced basis method if it can be evaluated with costs independent from the dimension $\mathcal{N}$ of the full model. One goal of the reduced basis community is to derive such error estimators for all kind of problems. Still, they are available only for certain problem classes, see [142, 84, 83, 23, 138, 122, 178, 158] and the references therein.

Based on these error estimators, the reduced basis method constructs the basis vectors spanning the reduced space with a greedy strategy. The method starts with a set of parameters $\mathcal{P} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_M\} \subset \mathcal{D}$ covering the parameter domain $\mathcal{D}$, and a one-dimensional reduced space $\mathcal{U}_1$ which is the span of $u^{\mathcal{N}}(\overline{\boldsymbol{\mu}})$ where $\overline{\boldsymbol{\mu}} \in \mathcal{D}$ is in most cases the center of the domain $\mathcal{D}$. For each parameter in $\boldsymbol{\mu} \in \mathcal{P}$ the estimated error $\xi_1(\boldsymbol{\mu})$ is computed. Recall that the evaluation costs of ξ_1 are independent from the dimension $\mathcal{N}$ of the space $\mathcal{U}^{\mathcal{N}}$. The solution $u^{\mathcal{N}}(\boldsymbol{\mu}^*)$ for the parameter $\boldsymbol{\mu}^*$ where the error estimator yields the highest value is computed. The new two-dimensional reduced space $\mathcal{U}_2$ is the span of $\{u^{\mathcal{N}}(\overline{\boldsymbol{\mu}}), u^{\mathcal{N}}(\boldsymbol{\mu}^*)\}$. This process is continued until n basis functions have been found. The state vectors associated to these basis functions give the basis vectors for the linear combination (8). Note that the solutions get orthogonalized to ensure a numerically stable system. Note further that only n solutions of the high-fidelity model are required. This leads to great savings in the Offline phase of RBM compared to a POD-based approach.

Of course, the reduced basis for the linear combination (8) can also be constructed without this greedy approach and without error estimators. For example, we could randomly select n parameters $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_n\} \subset \mathcal{D}$, compute the high-fidelity solutions $\{u^{\mathcal{N}}(\boldsymbol{\mu}_1), \dots, u^{\mathcal{N}}(\boldsymbol{\mu}_n)\} \subset \mathcal{U}^{\mathcal{N}}$, and orthogonalize them to obtain the final basis functions and then the basis vectors $\{\boldsymbol{\zeta}_1, \dots, \boldsymbol{\zeta}_n\}$. However, a poor selection of the parameters might lead to poor approximation results with the corresponding reduced basis. This illustrates that the success of the reduced basis method heavily relies on the rigorous, sharp, and efficient error estimators as well as on the described greedy strategy to select the reduced basis vectors.

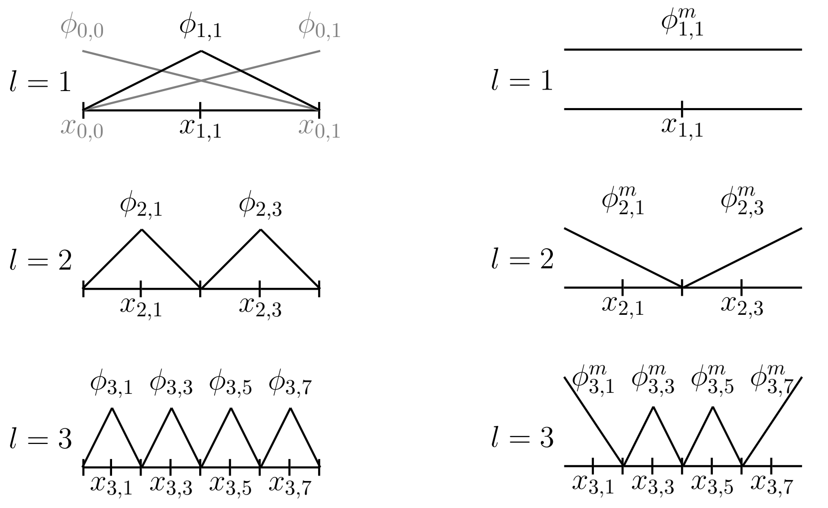
A solution $u_n(\boldsymbol{\mu})$ in the reduced basis space $\mathcal{U}_n$ is computed with Galerkin projection as in the case of the POD basis in (7). In presence of a nonlinear term $\boldsymbol{F}$ the same expensive evaluation as in the case of POD are required, see Sec. 12.

We can summarize that RBM requires only n solutions of the high-fidelity model to construct $\mathcal{U}_n$, that it includes error estimators, but that its scope is limited because those error estimators are only available for certain problem classes at the moment. Due to the Galerkin projection, and because in most cases a sound mathematical understanding and additional solves are required to derive the error estimators, the reduced basis method is an intrusive method.

2.4. The Curse of Dimensionality, Sparse Grids, and Adaptivity

Even though the sparse grid idea has already been presented in a three-page article in [171], the first considerations from a numerical point of view were made in [195] for the solution of elliptic PDEs of second-order. Nowadays, sparse grids are used for a variety of applications including, e.g., quadrature, interpolation, data mining, compression, finance, and uncertainty quantification. We refer to, e.g., [153, 156, 109, 79, 90, 192].

We give a brief overview of sparse grids and in particular define the notation in Sec. 2.4.1 before we discuss regression and classification with sparse grid functions in Sec. 2.4.2. In the following, we employ the sparse grid library SG^{++} developed by Dirk Pflüger [153].



(a) hierarchical basis (b) modified/nonuniform basis

Figure 3: The hierarchical basis functions for level $l = 1$ to $l = 3$ are shown in (a). If boundary points are required, level 0 is added. In (b) the so-called modified or nonuniform basis functions which extrapolate towards the boundary are illustrated [153].

2.4.1. Adaptive Sparse Grids

We introduce the hierarchical basis, hierarchical increments, and sparse grids followed by remarks on adaptivity, nonuniform basis functions, and coarsening by three.

Hierarchical basis functions We introduce the sparse grid space $\mathcal{V}_\ell^{(1)}$ of level ℓ corresponding to the domain $\Omega = [0,1]^d \subset \mathbb{R}^d$ by constructing its basis, the so-called hierarchical basis. The building block of the hierarchical basis is the one-dimensional standard hat function $\phi : [-1,1] \to \mathbb{R}$ defined as

$$\phi(x) = \max\left(1 - |x|, 0\right) . \tag{9}$$

The one-dimensional hierarchical hat function $\phi_{l,i}$ centered at the grid point $x_{l,i} = i \cdot 2^{-l}$ is the result of dilation and translation of ϕ,

$$\phi_{l,i}(x) = \phi(2^l x - i) . \tag{10}$$

The hierarchical basis functions for level $l = 1$ to $l = 3$ are shown in Fig. 3a. By using a tensor product approach we can extend the hat function (10) to the d-dimensional case

$$\phi_{\boldsymbol{l},\boldsymbol{i}}(x) = \prod_{j=1}^{d} \phi_{l_j,i_j}(x_j), \tag{11}$$

where $\boldsymbol{l} = (l_1, \dots, l_d)$ and $\boldsymbol{i} = (i_1, \dots, i_d)$ are multi-indices denoting the level and index, respectively. The corresponding grid point $\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}} = [x_{l_1,i_1}, \dots, x_{l_d,i_d}]^T$ is the center of the support of $\phi_{\boldsymbol{l},\boldsymbol{i}}$. The space $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ of piecewise d-linear functions is spanned by the nodal point basis

$$\tilde{\Phi}_{\boldsymbol{k}} = \left\{ \phi_{\boldsymbol{k},\boldsymbol{i}} \,|\, \boldsymbol{i} \in \tilde{\mathcal{I}}_{\boldsymbol{k}} \right\}, \tag{12}$$

with the index set

$$\tilde{\mathcal{I}}_{\boldsymbol{k}} = \left\{ \boldsymbol{i} \in \mathbb{N}^d \,|\, 1 \leq i_j < 2^{k_j},\, 1 \leq j \leq d \right\}. \tag{13}$$

Note that the definition (13) of the index set leads to no points at the boundary of the domain and thus we have homogeneous boundary conditions. Besides the set (13) we also introduce the hierarchical increments $\mathcal{W}_{\boldsymbol{k}}$ spanned by

$$\hat{\Phi}_{\boldsymbol{k}} = \left\{ \phi_{\boldsymbol{k},\boldsymbol{i}} \,|\, \boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{k}} \right\},$$

where

$$\hat{\mathcal{I}}_{\boldsymbol{k}} = \left\{ \boldsymbol{i} \in \mathbb{N}^d \,|\, 1 \leq i_j < 2^{l_j},\, i_j \notin 2\mathbb{N},\, 1 \leq j \leq d \right\}. \tag{14}$$

With the component-wise relational operator

$$\boldsymbol{l} \leq \boldsymbol{k} :\iff \forall 1 \leq j \leq d : l_j \leq k_j,$$

and its negation

$$\boldsymbol{l} \succ \boldsymbol{k} :\iff \exists 1 \leq j \leq d : l_j > k_j,$$

the space $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ can be represented as direct sum of hierarchical increments

$$\tilde{\mathcal{H}}_{\boldsymbol{k}} = \bigoplus_{\boldsymbol{l} \leq \boldsymbol{k}} \mathcal{W}_{\boldsymbol{l}},$$

and has, as well as the hierarchical increments $\mathcal{W}_{\boldsymbol{k}}$, tensor product structure. Thus, we can write them as

$$\mathcal{W}_{\boldsymbol{k}} = \bigotimes_{i=1}^{d} \mathcal{W}_{k_i} \qquad \text{and} \qquad \tilde{\mathcal{H}}_{\boldsymbol{k}} = \bigotimes_{i=1}^{d} \tilde{\mathcal{H}}_{k_i}.$$

Three properties of hierarchical basis functions First, the hierarchical basis functions have tensor product structure because we employed a tensor product to construct the multi-dimensional basis functions. Second, we can represent a basis function $\phi_{l,i}$ from level l as linear combination

$$\phi_{l,i}(x) = \frac{1}{2}\left(\phi_{l+1,2i-1}(x) + \phi_{l+1,2i+1}(x)\right) + \phi_{l+1,2i}(x), \tag{15}$$

see the illustration in Fig. 4. Third, for any $k \geq l$

$$\phi_{l,i}(x) = \sum_{t \in \tilde{\mathcal{I}}_k} \phi_{l,i}(x_{k,t}) \phi_{k,t}(x), \tag{16}$$

as the $\phi_{k,t}$ form a nodal basis of $\tilde{\mathcal{H}}_k$. These are three important properties of hierarchical basis functions which are needed later for the multigrid method in Sec. 4.

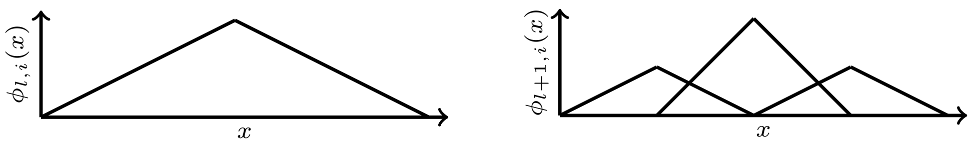


Figure 4: A basis function of level l (left) can be represented as linear combination of basis functions of level $l+1$ (right).

Full grids With the hierarchical increment $\mathcal{W}_{\boldsymbol{l}}$ we may construct further spaces

$$\mathcal{V} = \bigoplus_{\boldsymbol{l} \in \mathcal{L}} \mathcal{W}_{\boldsymbol{l}}$$

by selecting those hierarchical increments $\mathcal{W}_{\boldsymbol{l}}$ with level $\boldsymbol{l}$ in $\mathcal{L} \subset \mathbb{N}^d$. A function $f \in \mathcal{V}_{\mathcal{L}}$ can then be represented as a linear combination

$$f(\boldsymbol{x}) = \sum_{(\boldsymbol{l},\boldsymbol{i}) \in \mathcal{I}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) = \sum_{\boldsymbol{l} \in \mathcal{L}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \tag{17}$$

where the set $\mathcal{I}$ contains all level-index pairs $(\boldsymbol{l}, \boldsymbol{i})$ associated to the levels $\mathcal{L} \subset \mathbb{N}^d$. We sometimes abbreviate (17) with $f = \sum_i \alpha_i \phi_i$ where we assume an arbitrary ordering of the coefficients and basis functions. The coefficients $\alpha_{\boldsymbol{l},\boldsymbol{i}}$ are the so-called hierarchical surpluses. The space of piecewise d-linear functions $\mathcal{V}_\ell$ can then be written as

$$\mathcal{V}_\ell = \bigoplus_{|\boldsymbol{l}|_\infty \leq \ell} \mathcal{W}_{\boldsymbol{l}}$$

where $|\boldsymbol{l}|_\infty$ is $\max_i |l_i|$. Because $\mathcal{V}_\ell = \tilde{\mathcal{H}}_{(\ell,\dots,\ell)}$, the space $\mathcal{V}_\ell$ is not only spanned by $\bigcup_{|\boldsymbol{l}|_\infty \leq \ell} \hat{\Phi}_{\boldsymbol{l}}$ but also by the nodal point basis $\tilde{\Phi}_{(\ell,\dots,\ell)}$.

Sparse grids The selection

$$\mathcal{V}_\ell^{(1)} = \bigoplus_{|\boldsymbol{l}|_1 \leq \ell + d - 1} \mathcal{W}_{\boldsymbol{l}} \tag{18}$$

with $|\boldsymbol{l}|_1 = \sum_i |l_i|$ is called the sparse grid space of level ℓ and dimension d with respect to the L_2- and L_∞-norm, see [39]. Fig. 5 shows the grid points of the two-dimensional sparse grid of level three, the grid points of the corresponding hierarchical increments $\mathcal{W}_{\boldsymbol{l}}$, and the grid points of the hierarchical subspaces $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ with $|\boldsymbol{k}|_1 \leq \ell + d - 1$. The hierarchical basis of $\mathcal{V}_\ell^{(1)}$ is given by $\Phi_\ell = \bigcup_{|\boldsymbol{l}|_1 \leq \ell+d-1} \hat{\Phi}_{\boldsymbol{l}}$. Almost all of the following sparse grid algorithms work without change on any other subspace selection $\mathcal{L}$, provided that with $\boldsymbol{k} \in \mathcal{L}$ also $\boldsymbol{l} \in \mathcal{L}$ holds for all $\boldsymbol{l} \leq \boldsymbol{k}$.

Sparse grid functions $f \in \mathcal{V}_\ell^{(1)}$ can be evaluated in $\mathcal{O}(\ell^d)$ because the space $\mathcal{V}_\ell^{(1)}$ consists of $\mathcal{O}(\ell^d)$ hierarchical increments and all basis functions of a hierarchical increment have disjoint support. Thus, only one basis function per hierarchical increment has to be evaluated. Algorithms for an evaluation in $\mathcal{O}(\ell^d)$ are available, see, e.g., [153]. Recently, it

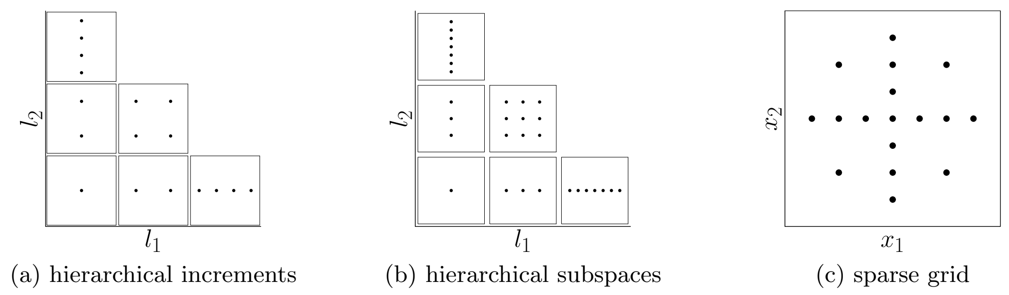


Figure 5: In (a) the grid points corresponding to the hierarchical increments $\mathcal{W}_{\boldsymbol{l}}$ of a sparse grid of level three arranged in the hierarchical scheme are shown. In (b) we plotted the grid points of the hierarchical subspaces $\tilde{\mathcal{H}}_{\boldsymbol{l}}$ and in (c) the corresponding sparse grid.

has been shown, that on modern hardware architectures it is under certain circumstances faster to iterate over all components of the linear combination $f = \sum_i \alpha_i \phi_i$ rather than to execute the asymptotically optimal but highly recursive evaluation algorithm [107].

The transformation from the function values into the hierarchical coefficients of a sparse grid space can be achieved with the so-called hierarchisation procedure. It computes the hierarchical coefficients $\boldsymbol{\alpha}$ in (17), i.e., the sparse grid interpolant, in linear complexity if the function values at the sparse grid points are given, see [39]. Its inverse operation is called dehierarchisation.

Properties of sparse grids With

$$D^{\boldsymbol{l}} f = \frac{\partial^{|\boldsymbol{l}|_1}}{\partial x_1^{l_1} \dots \partial x_d^{l_d}} f$$

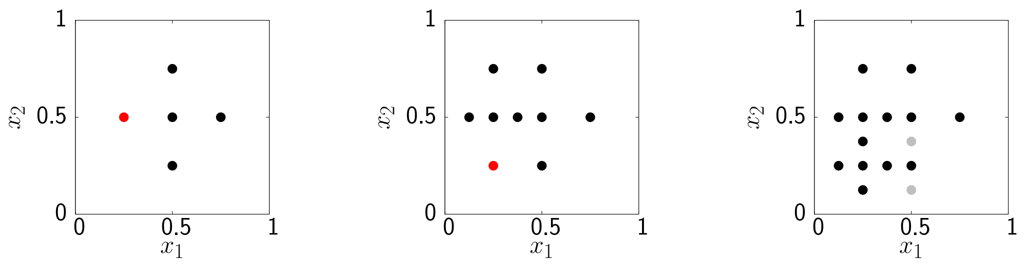
we define the space

$$H^2_{\text{mix}} = \left\{ f : \Omega \to \mathbb{R} \,|\, D^{\boldsymbol{l}} f \in L_2(\Omega) \,, |\boldsymbol{l}|_\infty \leq 2 \,, f|_{\partial\Omega} = 0 \right\}$$

of functions with bounded weak mixed derivatives up to order two. For a function $f \in H^2_{\text{mix}}$ and its interpolant $\tilde{f} \in \mathcal{V}_\ell$ in the space of piecewise d-linear functions, we obtain the asymptotic error

$$\|f(\boldsymbol{x}) - \tilde{f}(\boldsymbol{x})\|_{L_2} \in \mathcal{O}(h_\ell^2) \tag{19}$$

where $h_\ell = 2^{-\ell}$ is the mesh width corresponding to the space $\mathcal{V}_\ell$. The number of grid points corresponding to the full grid space $\mathcal{V}_\ell$ is in $\mathcal{O}(2^{\ell d})$ and thus affected by the curse of dimensionality. Now, let us assume we have the sparse grid interpolant $f_N \in \mathcal{V}_\ell^{(1)}$. Compared to the error decay (19) of the full grid space interpolant, the decay of the asymptotic error of the sparse grid interpolant f_N is only slightly deteriorated to $\mathcal{O}(h_\ell^2 \ell^{d-1})$. However, the number of sparse grid points, and thus the computational costs, have been significantly reduced to $\mathcal{O}(2^\ell \ell^{d-1})$. It can be shown that the sparse grid



(a) regular grid (b) after step one (c) after step two

Figure 6: The regular grid of level two in (a) is refined to obtain the sparse grid in (b). After another refinement step, the hierarchical ancestors (gray points) are created in (c) because most sparse grid algorithms require that the hierarchical ancestors of each grid point exist [153].

space $\mathcal{V}_\ell^{(1)}$ is the optimal approximation space for functions in H^2_{mix} with respect to the interpolation error in the L_2- and L_∞-norm. We refer to [39] for a detailed derivation of these error bounds.

Adaptivity If a function f should be interpolated and it is only known that $f \in H^2_{\text{mix}}$, i.e., that f fulfills the smoothness conditions discussed in the previous paragraph, and no further information about f is available, then the sparse grid interpolant is the optimal interpolant with respect to the L_2- and L_∞-norm. However, in many cases, some more characteristics about f are known. With adaptivity, this knowledge can be used to influence the structure of the sparse grid to adapt it to the special requirements of a particular function [39, 153].

Let $f_N = \sum_i \alpha_i \phi_i \in \mathcal{V}_\ell^{(1)}$ be the sparse grid interpolant of f, and assume we want to refine the sparse grid to get a better approximation of the function f, see Fig. 6. An adaptivity criterion is required to decide which sparse grid points should be refined. A simple but very robust and widely-used criterion is based on the hierarchical coefficients $\boldsymbol{\alpha}$ which refines the grid points corresponding to the coefficients with the largest absolute values. More sparse grid refinement criteria are discussed in [153, 154, 156]. Since adaptivity is highly problem-dependent, a good refinement criterion takes the context of the application into account. However, this also means that for each refinement criterion a problem can be constructed where it fails. Note that most sparse grid algorithms require that for each grid point all hierarchical ancestors exist, see Fig. 6.

Boundary treatment and nonuniform basis functions So far we only considered sparse grids without points at the boundary of the domain $[0, 1]^d$. This is enough for most applications considered here. However, sparse grids can be easily extended to have grid points at the boundary by adding a level 0, see Fig. 3a. We refer to [39] and [68] for more details.

Even though the number of grid points of a sparse grid with boundary points is still asymptotically in $\mathcal{O}(2^\ell \ell^{d-1})$, the constants in the estimations become huge and thus working with these grids becomes soon computationally infeasible in high-dimensional settings. That is why so-called modified or nonuniform basis functions have been developed [153]. These basis functions extrapolate towards the boundary and so do not require boundary points to approximate functions with non-zero values at the boundary. The hierarchical basis function $\phi_{l,i}$ is modified in the following way

$$\phi^m_{l,i}(x) = \begin{cases} 1 & \text{if } l = 1, i = 1, \\ \left.\begin{cases} 2 - 2^l \cdot x & \text{if } x \in [0, 2^{-(l-1)}]] \\ 0 & \text{else} \end{cases}\right\} & \text{if } l > 1, i = 1, \\ \left.\begin{cases} 2^l \cdot x + 1 - i & \text{if } x \in [1 - 2^{-(l-1)}, 1] \\ 0 & \text{else} \end{cases}\right\} & \text{if } l > 1, i = 2^l - 1, \\ \phi(x \cdot 2^l - i) & \text{else}\,. \end{cases} \tag{20}$$

In Fig. 3b we compare the classical (uniform) basis functions with the modified (nonuniform) basis functions. We refer to [153] for a detailed discussion. We will clearly indicate when we use the modified basis functions.

Coarsening by three Usually, sparse grids follow a coarsening by two, i.e., the support of the one-dimensional hierarchical basis function of level l is $2 \cdot 2^{-l}$. However, the multigrid method in Part I will be developed for sparse grids with a coarsening by three where the support of the basis function is $2 \cdot 3^{-l}$. The coarsening by three better fits to the space-filling curve approach introduced in [41]. The sparse grid theory presented in [39] also holds for sparse grids with a coarsening by three, only the constants in the error estimations are changed. If we have a sparse grid with coarsening by three, the hierarchical basis function (10) is defined as

$$\phi_{l,i}(x) = \phi(3^l x - i)\,. \tag{21}$$

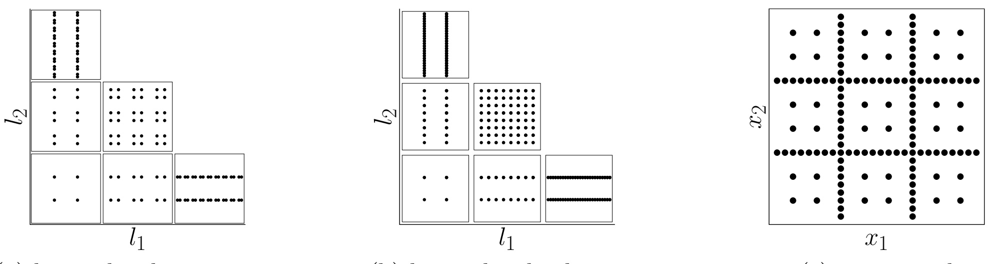
The set of indices (14) for the hierarchical increments is changed to

$$\hat{\mathcal{I}}_{\boldsymbol{k}} = \left\{ \boldsymbol{i} \in \mathbb{N}^d \,|\, 1 \leq i_j < 2^{l_j},\, i_j \notin 3\mathbb{N},\, 1 \leq j \leq d \right\}\,. \tag{22}$$

A two-dimensional sparse grid of level three with coarsening by three is shown in Fig. 7. We emphasize again that we employ coarsening by three only in Part I and that all discussed algorithms are also applicable to sparse grids with coarsening by two.

2.4.2. Regression and Classification with Sparse Grids

We consider supervised learning problems where the function $\hat{c} : \mathbb{R}^d \to \mathbb{R}$, mapping data points to their target values, is a sparse grid function. The corresponding optimization problem is introduced and properties of this sparse-grid-based classification and regression approach are discussed.



(a) hierarchical increments (b) hierarchical subspaces (c) sparse grid

Figure 7: A coarsening by three leads to a slightly different structure of the sparse grid. We plot in (a) the grid points corresponding to the hierarchical increments $\mathcal{W}_{\boldsymbol{l}}$ of a sparse grid of level three with coarsening by three arranged in the hierarchical scheme and in (b) the grid points of the hierarchical subspaces $\widetilde{\mathcal{H}}_{\boldsymbol{l}}$. The corresponding sparse grid is shown in (c).

Supervised learning and sparse grids Consider the supervised learning problem where we want to construct a function $\hat{c} : \mathbb{R}^d \to \mathbb{R}$ from the training data $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^M \subset \mathbb{R}^d \times \mathbb{R}$ to predict the target values y for new, unseen data points $\boldsymbol{x} \in \mathbb{R}^d$. In [79] this supervised learning problem is formulated with Tikhonov regularization as an optimization problem

$$\hat{c} = \operatorname*{arg\,min}_{f \in \mathcal{V}_\ell^{(1)}} \left(\frac{1}{M} \sum_{i=1}^M (y_i - f(\boldsymbol{x}_i))^2 + \lambda \|\Lambda f\|_{L_2}^2 \right) \tag{23}$$

where the function $\hat{c} \in \mathcal{V}_\ell^{(1)}$ is discretized on a sparse grid. If we explicitly represent the sparse grid function as a linear combination of the hierarchical basis functions in Φ_ℓ and the hierarchical coefficients $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_N]^T$, we can rewrite (23) as

$$\operatorname*{arg\,min}_{\boldsymbol{\alpha}} \left(\frac{1}{M} \sum_{i=1}^M \left(y_i - \sum_{j=1}^N \alpha_j \phi_j(\boldsymbol{x}_i) \right)^2 + \lambda \|\Lambda \sum_{j=1}^N \alpha_j \phi_j\|_{L_2}^2 \right) \tag{24}$$

and make clear that we are looking for the hierarchical coefficients $\boldsymbol{\alpha}$ defining the function $\hat{c} \in \mathcal{V}_\ell^{(1)}$. The first term of (24) ensures closeness of $\hat{c}$ to the training data and the second term imposes a certain smoothness on $\hat{c}$ in order to generalize to new, previously unseen data. The regularization parameter λ can be determined via cross validation and controls the trade-off between fidelity and smoothness. The optimization problem (24) leads to a system of linear equations [79]

$$\left(\frac{1}{M} \boldsymbol{B}\boldsymbol{B}^T + \lambda \boldsymbol{C} \right) \boldsymbol{\alpha} = \frac{1}{M} \boldsymbol{B}\boldsymbol{y}, \tag{25}$$

where $B_{ij} = \phi_i(\boldsymbol{x}_j)$, $C_{ij} = (\Lambda\phi_i, \Lambda\phi_j)_{L_2}$, and $\boldsymbol{y} = [y_1, \dots, y_M]^T$. The system (25) is usually solved with the conjugate gradient (CG) method where only the matrix-vector product with $\boldsymbol{B}$, $\boldsymbol{B}^T$, and $\boldsymbol{C}$ is required. We emphasize that the size of the system matrix $\frac{1}{M}\boldsymbol{B}\boldsymbol{B}^T + \lambda\boldsymbol{C}$ is only $N \times N$ where N is the number of sparse grid points. In particular, the number of unknowns of (25) does not depend on the number of data points M.

Regularization operator The regularization operator Λ is typically ∇, but simpler and computationally more efficient choices are possible [153, 154]. For example, in many situations, the problem (24) can be simplified to

$$\arg\min_{\boldsymbol{\alpha}} \left(\frac{1}{M} \sum_{i=1}^{M} \left(y_i - \sum_{j=1}^{N} \alpha_j \phi_j(\boldsymbol{x}_i) \right)^2 + \lambda \sum_{j=1}^{N} \alpha_j^2 \right)$$

leading to the system of linear equations (25) where the regularization matrix $\boldsymbol{C}$ is the identity matrix $\boldsymbol{I}$. This severely reduces the computational costs of constructing the function $\hat{c}$. With the term $\sum_j \alpha_j^2$ we limit the growth of the hierarchical coefficients. Since the hierarchical coefficients correspond to the second derivative of our sparse grid function $\hat{c} = \sum_j \alpha_j \phi_j$, it imposes smoothness on $\hat{c}$, see [153] for an extensive study.

Classification problem and sparse grids If we have a binary classification problem, i.e., the target values in $\{y_1, \ldots, y_M\}$ are either -1 or 1, the class label y of a new data point $\boldsymbol{x} \in \mathbb{R}^d$ is determined by evaluating $\hat{c}(\boldsymbol{x})$ and setting

$$y = \begin{cases} -1, & \text{if } \hat{c}(\boldsymbol{x}) < 0, \\ 1, & \text{if } \hat{c}(\boldsymbol{x}) \geq 0. \end{cases}$$

The method can also be extended to more than two classes by some workaround which requires to construct multiple classifiers, see, e.g., [153].

Part I.
MOR I: A Priori Model Order Reduction

Model order reduction constructs a low-dimensional, problem-dependent solution space $\mathcal{U}_n$ for input-output problems $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$. To achieve this, it is very common to solve a given problem for selected parameters in a high-dimensional, general space first and then derive a low-dimensional, problem-dependent space from these solutions. We will deal extensively with such approaches in Parts II and III. In this part, however, we do not have a low-dimensional solution space fitted to a particular problem but a space tailored to a whole problem class. This means the construction of the space does not take the solutions or data of a particular problem into account but is built only from theoretic considerations about the problem class. Thus, all information for the construction of the space is derived before a particular problem of the class has to be solved, maybe even before a particular problem is known. This is what we call *a priori* model order reduction.

Gaussian quadrature is related to this concept. It gives the optimal choice of n sampling points and n quadrature weights for polynomial functions of degree $2n - 1$. It is not necessary to know the specific function which should be integrated. Only the knowledge that it is in the function class of polynomial functions is required. Sparse grids follow the same approach. They are an optimal way to discretize functions in H^2_{mix}, i.e., functions with bounded, mixed derivatives up to order two with respect to a certain norm. Thus, if we have an elliptic PDE with solution function $u \in H^2_{\text{mix}}$, it is more efficient with respect to degrees of freedom versus discretization error, to represent it in a low-dimensional sparse grid space $\mathcal{V}^{(1)}_\ell$ corresponding to the function space H^2_{mix} rather than in a general, high-dimensional full grid space $\mathcal{V}_\ell$. Note that it is not necessary to solve the PDE first to determine the sparse grid space $\mathcal{V}^{(1)}_\ell$. Note further that with adaptive sparse grids we can soften this *a priori* approach and incorporate specific knowledge about our current function. We refer to Sec. 2.4 for more details on sparse grids and adaptivity.

The result of a sparse grid discretization and the subsequent Galerkin projection of a linear, elliptic PDE is a system of linear equations. In this part, we introduce a novel multigrid method well-suited for sparse grids to solve such systems of equations.

A finite element discretization with sparse grids for elliptic PDEs and the system of linear equations stemming from the Galerkin projection are introduced in Sec. 3. The systems of linear equations are usually solved either with preconditioned Krylov or with multigrid methods. Both of these approaches are able to solve such systems with a computational complexity scaling only linearly with the number of grid points. Due to the sparse grid structure and the hierarchical basis underlying sparse grids, however, most full grid algorithms are not applicable to sparse grid discretizations and thus special algorithms are required. Preconditioned Krylov methods for sparse grids are briefly discussed in Sec. 3.2. But most preconditioning techniques suitable for sparse grid discretizations introduce additional costs. Multigrid methods exploit the smoothing effect of relaxation methods by correcting the solution with approximations on coarse grids. An overview of multigrid methods for sparse grids is also given in Sec. 3.2 with details in Sec. 3.3. However, these methods either operate on a simplified discretization or introduce an additional approximation step. Therefore, we present a novel multigrid method based on an ANOVA-like (Analysis of Variance) decomposition of the sparse grid solution function in Sec. 4. It can handle the multi-dimensional convection-diffusion equation discretized on spatially adaptive sparse grids without simplifying the corresponding bilinear form and still achieves a linear complexity in the number of sparse grid points. The core of our multigrid method is the dimension-wise decomposition of the sparse grid solution function discussed in Sec. 4.1 and the corresponding algorithm to update the components of the decomposition in Sec. 4.2. We apply the multigrid method to the multi-dimensional convection-diffusion equation in Sec. 5.

In this part, we consider sparse grids with a coarsening by three, see Fig. 7. This has mere technical reasons [41]. All the discussed algorithms are also applicable to classical sparse grids with a coarsening by two.

This multigrid method is the result of a close collaboration with Christoph Zenger. Furthermore, only due to Stefan Zimmer a reader-friendly exposition of the sometimes tricky algorithm and a clear notation for the lengthy formulas were found.

3. Adaptive Sparse Grid Discretization of Elliptic PDEs

We consider elliptic PDEs with spatial domain $\Omega = [0,1]^d$ where we are particularly interested in the case with $d > 3$. It is not feasible anymore to discretize these equations with $d > 3$ on full grids due to the curse of dimensionality. Therefore, we discretize them on sparse grids. The Galerkin projection onto a sparse grid space and the corresponding system of linear equations are derived in Sec. 3.1. We briefly discuss related sparse grid solvers in Sec. 3.2, and develop general concepts of multigrid methods for sparse grids in Sec. 3.3. We show that the algorithmic challenge is to compute the right-hand sides of the systems of linear equations when we switch in the grid hierarchy from one grid to another.

3.1. Elliptic Partial Differential Equations and Sparse Grids

Let $\Omega \subset \mathbb{R}^d$ be a domain, $\mathcal{V}$ a Hilbert space, $\mathcal{V}^*$ its dual, $a_{\mathrm{A}} : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ a bilinear form, and $b : \mathcal{V} \to \mathbb{R}$ a linear form corresponding to an elliptic second-order PDE with the linear operator $\mathrm{A} : \mathcal{V} \to \mathcal{V}^*$. We may also simply write a if the specific operator A is not important.

We assume that the operator A can be decomposed into a product of one-dimensional operators

$$\mathrm{A} = \bigotimes_{i=1}^{d} \mathrm{A}^{(i)}.$$

For example, if we consider Laplace's equation $\Delta u = 0$ in $\Omega \subset \mathbb{R}^3$, we can split the operator Δ as follows

$$\Delta = \partial_{x_1}^2 \otimes \mathrm{I}_{x_2} \otimes \mathrm{I}_{x_3} + \mathrm{I}_{x_1} \otimes \partial_{x_2}^2 \otimes \mathrm{I}_{x_3} + \mathrm{I}_{x_1} \otimes \mathrm{I}_{x_2} \otimes \partial_{x_3}^2,$$

where $\partial_{x_i}^2$ denotes the second derivative operator in direction i, and I_{x_i} the identity operator in direction i. This means the operator Δ is split into three summands $\mathrm{A}_1, \mathrm{A}_2$, and A_3 which themselves are tensor products of one-dimensional operators $\mathrm{A}_1^{(1)}, \mathrm{A}_1^{(2)}, \mathrm{A}_1^{(3)}, \mathrm{A}_2^{(1)}, \dots$.

We want to find the function $u^e \in \mathcal{V}$ such that

$$a_{\mathrm{A}}(u^e, v) = b(v), \qquad \forall v \in \mathcal{V}.$$

We pursue a Ritz-Galerkin approach and employ a sparse grid space $\mathcal{V}_\ell^{(1)}$ with N grid points instead of a full grid space $\mathcal{V}_\ell$ with $\mathcal{N}$ grid points. Thus, our test and ansatz space is $\mathcal{V}_\ell^{(1)} \subseteq \mathcal{V}$ and we want to find $u_N \in \mathcal{V}_\ell^{(1)}$ such that

$$a_{\mathrm{A}}(u_N, v) = b(v), \qquad \forall v \in \mathcal{V}_\ell^{(1)}. \tag{26}$$

Note that in this case the number of degrees of freedom n of the reduced-order model is equal to the number of sparse grid points N, cf. Sec. 2.3.1. We obtain the Galerkin approximation u_N by solving the system of linear equations

$$a_{\mathrm{A}}(u_N, \psi) = b(\psi), \qquad \forall \psi \in \Psi_\ell \tag{27}$$

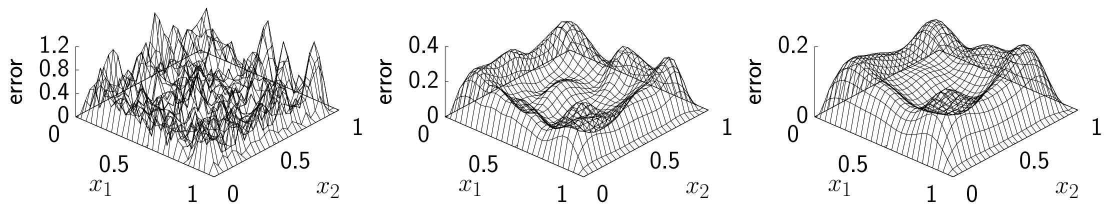
where $\Psi_\ell = \Phi_\ell$ is the hierarchical basis of the sparse grid space $\mathcal{V}_\ell^{(1)}$. Note that the ansatz and test space are equal but we denote a basis function with ϕ and a test function with ψ for an easier exposition. In the following, we may skip the superscript e and N if it is clear whether we mean u^e or u_N.

We could form the system matrix corresponding to (27) and solve the system with any general purpose solvers. However, this is computationally prohibitive because the number of sparse grid points N is too large. Therefore, matrix-free solvers have to be employed. In case of Krylov subspace methods only the matrix-vector product is required. Multigrid methods relax the equations (27) with, e.g., Jacobi or Gauss-Seidel relaxation on each grid of the grid hierarchy and thus also do not explicitly form the system matrix. We refer to Sec. 3.2 for related sparse grid solvers, and to Sec. 3.3 for details on multigrid methods suited for sparse grid discretizations.

3.2. Sparse Grid Solvers

By using sparse grids instead of full grids in the Galerkin projection (26), orders of magnitude fewer grid points are necessary to obtain a solution that has a similar accuracy as the full grid solution. The drawback of sparse grids is their unconventional structure and the hierarchical basis which render already simple tasks into challenges and lead to highly involved algorithms.

Preconditioned Krylov subspace methods When Krylov subspace methods such as the conjugate gradient (CG) method are employed to solve systems such as (27), we only have to provide the matrix-vector product with the stiffness matrix and do not have to explicitly assemble the matrix. For the matrix-vector product with the stiffness matrix stemming from a sparse grid discretization, the so-called UpDown algorithm can be used [1, 22, 38]. It provides the matrix-vector product in $\mathcal{O}(d \cdot 2^d \cdot N)$ where N is the number of sparse grid points and d the dimension of the spatial domain Ω. The UpDown algorithm uses the so-called unidirectional principle which means that the multi-dimensional algorithm is composed of algorithms operating only in one direction. This is possible because it uses grid points to temporarily store intermediate results. However, the structure of sparse grids does not allow a separate straightforward traversal of all grid points in each direction [1, 67]. That is why a highly recursive and complicated traversal through the grid points is necessary. The numerous recursive calls make the algorithm not only hard to implement but also slow on modern hardware [107]. Finally, in order to obtain an efficient CG method, appropriate preconditioners have to be applied. However, this introduces additional costs. Many preconditioning techniques have been presented [191, 67, 92, 93]. In [191] it is shown for the two-dimensional case that already the change from the nodal point to the hierarchical basis reduces the condition number of the system matrix. It is also stated that even though there is also an improvement for $d > 2$, it becomes almost negligible. In [92] and in the follow up [93] advanced subspace splitting techniques for sparse grids are analyzed. They also present numerical results for the Helmholtz equation. Newer approaches with prewavelets or multilevel frames are discussed in [67, 101].



(a) initial error (b) after 1 relaxation (c) after 2 relaxations

Figure 8: The initial error and the error after one and two steps of a classical iterative method such as Jacobi or Gauss-Seidel relaxation is shown. The example demonstrates the smoothing effect of the relaxation method. The axis with the error is scaled in order to make the smoothing effect more evident.

More recently, algorithms have been presented which achieve a log-linear complexity for the matrix-vector product with the stiffness matrix for a wavelet basis by exploiting the tensor product structure of the operators [167, 194]. They are developed and used in the context of stochastic elliptic problems.

Multigrid principles Besides preconditioned CG, other frequently used efficient methods to solve large systems of linear equations resulting from a finite element discretization are multigrid approaches [31, 99]. Multigrid methods build on two principles: error smoothing and coarse grid correction. Classical iterative methods, such as Jacobi or Gauss-Seidel relaxation, require many iterations to remove low-frequency error modes, i.e., they have a strong smoothing effect on the error. This smoothing effect is exploited to remove high-frequency components of the error, see Fig. 8. Only low-frequency components remain. A smooth quantity on a fine grid can be well approximated on a coarse grid. Hence, the smooth error term can be transferred to a coarse grid without loss of essential information. There, however, the smooth error with respect to the fine grid becomes a high-frequency error with respect to the coarse grid. Thus, just a few relaxation sweeps on the coarse grid are necessary to remove the high-frequency error what was low-frequency error on the fine grid. Since a coarse grid has less grid points than a fine grid, a smoothing step on the coarse grid is distinctly cheaper than on the fine grid. This makes it computationally feasible to recursively continue this process on a whole hierarchy of grids. The so-called multigrid cycle determines the path through this hierarchy. To achieve optimal performance, i.e., a convergence rate independent of the mesh width of the finest grid [99], not only the transfer from the fine to the coarse grid (restriction) and vice versa (prolongation) has to be done properly, but also the smoothing procedure, number of smoothing steps per multigrid cycle, coarse grid operator, and multigrid cycle have to be chosen adequately [175].

Multigrid methods for sparse grids Multigrid methods have been extensively studied in the context of sparse grids, see, e.g., [151, 37, 87, 95, 1]. These multigrid methods are either used as preconditioners or as real multigrid methods. In [87] the implementation details of a multigrid preconditioner for finite difference discretizations on sparse grids are shown. A similar approach for the finite element method is used in [1]. We will focus on real multigrid schemes as in [150, 151, 37]. Again, the sparse grid structure makes a straightforward generalization of multigrid methods for full grids impossible. The difficulty is that the discrete equations from the Galerkin projection onto sparse grid spaces lead to very cumbersome right-hand sides on the coarse grids which are hard to compute in an efficient way. Efficient computation means to construct the right-hand side in $\mathcal{O}(N)$ where N is the number of grid points. In [151, 150] the bilinear form corresponding to the variational problem of the Helmholtz (or Poisson) equation is modified in such a way that an efficient computation of the right-hand side is possible. It is then shown that the discretization with the simplified bilinear form converges to the sparse grid solution. However, this is only demonstrated for the case $d \leq 2$. We will elaborate below on how the bilinear form is simplified and what this exactly means for the solution. In [37] a similar approach is used for the two-dimensional Poisson equation with higher order finite elements.

3.3. Multigrid Methods and Sparse Grids

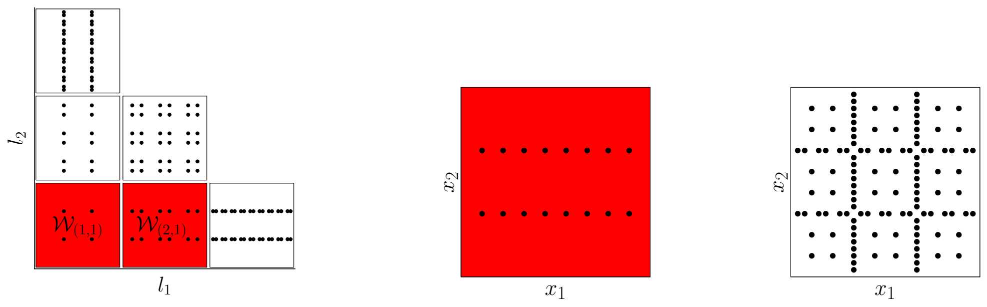
In the previous section we have seen several multigrid methods for sparse grids. They all rely on the same grid hierarchy which is discussed in this section. To relax the equations corresponding to the grids of this hierarchy, we introduce a splitting of the sparse grid space. We then show why it is computationally expensive to compute the right-hand side when we switch in the grid hierarchy from one grid to another.

Multigrid on sparse grids Consider a sparse grid and its hierarchical subspaces, see Figures 5 and 7. Clearly, a sparse grid does not have a natural grid hierarchy if we coarsen in all directions simultaneously as common multigrid methods do. However, if we consider the hierarchical subspaces in

$$\tilde{\mathcal{H}} = \{\tilde{\mathcal{H}}_{\boldsymbol{k}} : |\boldsymbol{k}|_1 \leq \ell + d - 1\} \tag{28}$$

of a sparse grid space $\mathcal{V}_\ell^{(1)}$, we can think of the grids corresponding to these subspaces as the coarse grids of a sparse grid. Thus, we define the grid hierarchy of a sparse grid to consist of the grids of the subspaces in $\tilde{\mathcal{H}}$.

These are full grids where we can employ an ordinary Gauss-Seidel relaxation by applying the corresponding stencils at the grid points [175]. In order to reach these grids, we need to allow semi-coarsening which means that we can coarsen and refine in each direction separately. Such a semi-coarsening allows us to walk through the subspace scheme of a sparse grid and relax with respect to any subspace. This means that we can start with, e.g., subspace $\tilde{\mathcal{H}}_{(k_1,\dots,k_d)}$ and then decide to coarsen in, say, direction $\tilde{d}$ to end up in subspace $\tilde{\mathcal{H}}_{(k_1,\dots,k_{\tilde{d}}-1,\dots,k_d)}$. The so-called multigrid cycle defines the way through the subspace scheme.



(a) hierarchical increments (b) $\tilde{\mathcal{H}}_{(2,1)}$ (c) $\hat{\mathcal{H}}_{(2,1)}$

Figure 9: The direct sum of the hierarchical increments $\mathcal{W}_{(1,1)}$ and $\mathcal{W}_{(2,1)}$ equals the subspace $\tilde{\mathcal{H}}_{(2,1)}$ (red). The rest is denoted by $\hat{\mathcal{H}}_{(2,1)}$ (white). Note that we employ sparse grids with coarsening by three, cf. Sec. 2.4.

Comparison with the combination technique Even though these multigrid methods relax on full grids only, the finite element solution in the sparse grid space is obtained. With the so-called combination technique, it is also possible to work only on full grids and to arrive at a sparse grid solution [104, 94]. It exploits that a sparse grid interpolant $f_\ell = \sum_i \alpha_i \phi_i \in \mathcal{V}_\ell^{(1)}$ of a d-dimensional function f can be represented as the sum

$$f_\ell(\boldsymbol{x}) = \sum_{i=0}^{d-1} (-1)^i \binom{d-1}{i} \sum_{|\boldsymbol{l}|_1 = \ell - i} f_{\boldsymbol{l}}(\boldsymbol{x}) \,, \tag{29}$$

where $f_{\boldsymbol{l}} \in \tilde{\mathcal{H}}_{\boldsymbol{l}}$ is the interpolant of f on a full grid with level $\boldsymbol{l} = (l_1, \dots, l_d)$, see [94]. Thus, only the interpolants on coarse full grids have to be constructed to get the sparse grid interpolant f_ℓ. If we carry (29) over to solution functions of PDEs, we do not discretized the PDE on a sparse grid but compute multiple full grid solutions and combined them according to (29). There, however, the solution is not the finite element solution in the sparse grid space but a linear combination of solutions in ordinary finite element spaces. For selected elliptic model problems it can be shown that the solution obtained by the combination technique has similar approximation properties as the sparse grid solution [40, 86]. However, this is not clear in the general case, and that is why the finite element solution in the sparse grid space is still preferred. Furthermore, it is hard to extend the combination technique to spatially (locally) adaptive sparse grids.

Relaxation on subspaces Let us now discuss what it means to relax with respect to a subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ of a sparse grid space $\mathcal{V}_\ell^{(1)}$. In Fig. 9, we plot the grid points corresponding to the subspace $\tilde{\mathcal{H}}_{(2,1)}$ of $\mathcal{V}_3^{(1)}$ in two dimensions.

Consider the hierarchical subspaces (28) of a sparse grid space $\mathcal{V}_\ell^{(1)}$. We split $\mathcal{V}_\ell^{(1)}$ into subspaces $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ and $\hat{\mathcal{H}}_{\boldsymbol{k}}$ such that

$$\mathcal{V}_\ell^{(1)} = \tilde{\mathcal{H}}_{\boldsymbol{k}} \oplus \hat{\mathcal{H}}_{\boldsymbol{k}} \,. \tag{30}$$

We know that all subspaces $\tilde{\mathcal{H}}_{\boldsymbol{k}} \in \tilde{\mathcal{H}}$ are spanned by a nodal point basis $\tilde{\Phi}_{\boldsymbol{k}}$, cf. Sec. 2.4. Thus, we have a nodal point basis $\tilde{\Phi}_{\boldsymbol{k}}$ of $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ and the basis $\bigcup_{\boldsymbol{l} \succ \boldsymbol{k}} \hat{\Phi}_{\boldsymbol{l}}$ of $\hat{\mathcal{H}}_{\boldsymbol{k}}$ where $\hat{\Phi}_{\boldsymbol{l}}$ is the basis of the hierarchical increment $\mathcal{W}_{\ell}$. The basis of $\hat{\mathcal{H}}_{\boldsymbol{k}}$ can be seen as a hierarchical basis with respect to level $\boldsymbol{k}$. Following (30), we can then decompose a sparse grid function $u \in \mathcal{V}_{\ell}^{(1)}$ into a function $\tilde{u}$ of the space $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ and a function $\hat{u}$ of the space $\hat{\mathcal{H}}_{\boldsymbol{k}}$ with

$$u = \tilde{u} + \hat{u}.$$

Now, when we traverse through the subspaces $\tilde{\mathcal{H}}_{\boldsymbol{k}} \in \tilde{\mathcal{H}}$ as prescribed by the multigrid cycle, we need to form a system of linear equations for each $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ to perform a Gauss-Seidel relaxation. Let $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ be the current subspace and let $\Psi_{\boldsymbol{k}} = \tilde{\Phi}_{\boldsymbol{k}}$ contain the test functions, then we have to relax the equations

$$a(u, \psi_{\boldsymbol{k},\boldsymbol{t}}) = 0, \qquad \forall \psi_{\boldsymbol{k},\boldsymbol{t}} \in \Psi_{\boldsymbol{k}}, \tag{31}$$

with respect to the coefficients $\alpha_{\boldsymbol{k},\boldsymbol{i}}$ with $\boldsymbol{i} \in \tilde{\mathcal{I}}_{\boldsymbol{k}}$ corresponding to the nodal point basis $\tilde{\Phi}_{\boldsymbol{k}}$ of $\tilde{H}_{\boldsymbol{k}}$. We set the bilinear form b on the right-hand side to 0 to simplify matters here.

Right-hand side Because we have $u = \tilde{u} + \hat{u}$ we can split the equations (31) such that all coefficients $\alpha_{\boldsymbol{k},\boldsymbol{i}}$ belonging to $\tilde{\Phi}_{\boldsymbol{k}}$ are on the left-hand side

$$a(\tilde{u}, \psi_{\boldsymbol{k},\boldsymbol{t}}) = -a(\hat{u}, \psi_{\boldsymbol{k},\boldsymbol{t}}), \qquad \forall \psi_{\boldsymbol{k},\boldsymbol{t}} \in \Psi_{\boldsymbol{k}}. \tag{32}$$

We see that the stiffness matrix corresponding to the bilinear form a on the left-hand side can be easily assembled because $\tilde{u} = \sum_{\boldsymbol{i}} \alpha_{\boldsymbol{k},\boldsymbol{i}} \phi_{\boldsymbol{k},\boldsymbol{i}}$ is a linear combination of the nodal point basis $\tilde{\Phi}_{\boldsymbol{k}}$. However, for the relaxation we also need to have the right-hand side with $\hat{u} = \sum_{\boldsymbol{l} \succ \boldsymbol{k}} \sum_{\boldsymbol{i}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \phi_{\boldsymbol{l},\boldsymbol{i}}$ leading to

$$a(\hat{u}, \psi_{\boldsymbol{k},\boldsymbol{t}}) = \sum_{\boldsymbol{l} \succ \boldsymbol{k}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} a\left(\phi_{\boldsymbol{l},\boldsymbol{i}}, \psi_{\boldsymbol{k},\boldsymbol{t}}\right), \tag{33}$$

which has to be computed whenever we change the current subspace $\tilde{H}_{\boldsymbol{k}}$. That the right-hand side (33) still contains a hierarchical basis has a huge impact on the computational procedure and is the reason why we cannot simply employ any standard multigrid method. We see that (33) includes a sum over the levels $\boldsymbol{l} \succ \boldsymbol{k}$. If we had a nodal point basis, the support of the test function $\psi_{\boldsymbol{k},\boldsymbol{t}}$ would only intersect with the support of a few basis functions in its neighborhood. However, because we employ hierarchical (test and) basis functions, the test function $\psi_{\boldsymbol{k},\boldsymbol{t}}$ reaches to many basis functions centered all over the domain Ω. This means, we have to take the value and the hierarchical surplus of many basis functions into account and it also means that the stiffness matrix corresponding to (26) is not sparse. In the following, we introduce a further decomposition of $\hat{u}$, and thus of the right-hand side, such that we are able to provide (33) efficiently, i.e., without iterating over all basis functions with $\boldsymbol{l} \succ \boldsymbol{k}$ every time we need to evaluate the right-hand side $a(\hat{u}, \psi_{\boldsymbol{k},\boldsymbol{t}})$.

4. Multigrid for Adaptive Sparse Grids with ANOVA-like Decomposition

In this section we present our multigrid method to solve elliptic PDEs discretized on sparse grids. We have seen that during the change from one grid to another grid in the grid hierarchy, we have to update the right-hand side of the system of linear equations corresponding to the Galerkin projection. This update becomes computational very expensive if not done properly. Our method decomposes the sparse grid solution function as well as the right-hand side in a dimension-wise fashion to allow an efficient update when moving from one grid to another. The update procedure reuses components corresponding to hierarchically lower sparse grid points and thus avoids recomputing the components for all grid points. A kind of store and load concept is introduced by hierarchising the components if we move upwards in the grid hierarchy and by dehierarchising them again if we move downwards. This mechanism of exploiting the grid hierarchy in this way is only possible because of the dimension-wise decomposition.

In the following section we introduce the dimension-wise decomposition of the sparse grid solution function and show how to evaluate the bilinear form at each component. In Sec. 4.2 we continue with the COARSEN and REFINE procedures to move to a coarser and finer grid, respectively. These procedures have to keep the components of the solution function up-to-date when the grid is changed. Finally, the storage and runtime complexity of our multigrid algorithm is discussed.

4.1. Dimension-Wise Decomposition of Sparse Grid Functions

In the previous section, we have defined what it means to relax with respect to a subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ of a sparse grid space and we have shown that the right-hand side of the corresponding system of linear equations still depends on an hierarchical representation. In order to efficiently compute the right-hand side, we have to decompose the sparse grid function dimension-wise. Previous approaches without such a decomposition, e.g., [150, 151, 37, 145, 85], are not applicable to general second-order elliptic PDEs in the d-dimensional sparse grid case because only parts of the right-hand side can be computed while traversing through the hierarchical subspace scheme.

In this section, we introduce the dimension-wise decomposition and show how to compute the bilinear form for each of these components separately. It turns out that each component can be computed by applying a stencil to specific quantities from the decomposition.

Dimension-wise decomposition We consider the dimension-wise decomposition of a d-dimensional continuous function f,

$$\begin{aligned} f(x_1,\dots,x_d) = \tilde{f} &+ \sum_{j_1}^{d} f_{j_1}(x_{j_1}) + \sum_{j_1<j_2}^{d} f_{j_1,j_2}(x_{j_1},x_{j_2}) \\ &+ \sum_{j_1<j_2<j_3}^{d} f_{j_1,j_2,j_3}(x_{j_1},x_{j_2},x_{j_3}) + \dots \\ &+ f_{j_1,\dots,j_d}(x_{j_1},\dots,x_{j_d}), \end{aligned} \tag{34}$$

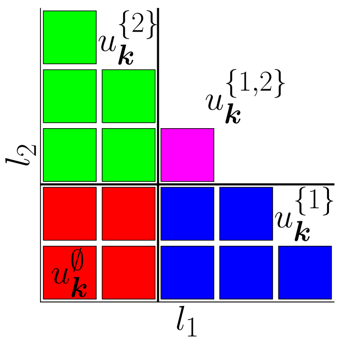
where $\tilde{f}$ is a constant function, f_{j_1} is a one-dimensional function, and so on [88]. This is a so-called ANOVA-like decomposition, cf. [59, 182, 88, 90]. For a discussion on the properties of the decomposition, we refer to [88] and the references therein. Similar decompositions of high-dimensional functions (HDMR) are discussed in [159, 130]

Here we are interested in the decomposition of a sparse grid function $u \in \mathcal{V}_\ell^{(1)}$ with respect to a subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$. This means, the constant function $\tilde{f}$ in (34) is replaced by a function $\tilde{u} \in \tilde{\mathcal{H}}_{\boldsymbol{k}}$ as well as all further functions $u_{j_1}, u_{j_1,j_2}, \dots$ are constructed with respect to $\tilde{\mathcal{H}}_{\boldsymbol{k}}$

$$\begin{aligned} u(x_1,\dots,x_d) = &\sum_{\boldsymbol{l}\le\boldsymbol{k}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \\ &+ \sum_{j_1}^{d} \sum_{\substack{\boldsymbol{l}\in\mathcal{L} \\ l_{j_1}>k_{j_1} \\ \forall i\in\{1,\dots,d\}\setminus\{j_1\}: l_i\le k_i}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \\ &+ \sum_{j_1<j_2}^{d} \sum_{\substack{\boldsymbol{l}\in\mathcal{L} \\ l_{j_1}>k_{j_1}, l_{j_2}>k_{j_2} \\ \forall i\in\{1,\dots,d\}\setminus\{j_1,j_2\}: l_i\le k_i}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \\ &+ \dots \\ &+ \sum_{\boldsymbol{l}>\boldsymbol{k}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}), \end{aligned} \tag{35}$$

where the set $\mathcal{L} \subset \mathbb{N}^d$ contains the levels of the hierarchical increments of the sparse grid space $\mathcal{V}_\ell^{(1)}$. This representation of u is unique because it is only, at first sight, a very cumbersome way to express the linear combination (17). We can relate the 2^d terms of (35) to the subsets $\omega \subseteq \{1,\dots,d\}$. For instance, we can associate $\omega = \{1,2\}$ to the term of the decomposition which contains the basis functions with $l_1 > k_1$, $l_2 > k_2$ and $l_j \le k_j$ for $j \in \{1,\dots,d\} \setminus \omega$. In general, we define the component

$$u_{\boldsymbol{k}}^{\omega} = \sum_{\substack{\boldsymbol{l}\in\mathcal{L} \\ \forall i\in\omega: l_i>k_i \\ \forall i\notin\omega: l_i\le k_i}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \tag{36}$$



$u_{\boldsymbol{k}}^{\{2\}}$
$u_{\boldsymbol{k}}^{\{1,2\}}$
$u_{\boldsymbol{k}}^{\{1\}}$
$u_{\boldsymbol{k}}^{\emptyset}$
l_2
l_1

Figure 10: The two-dimensional ANOVA-like decomposition with respect to $\tilde{\mathcal{H}}_{(2,2)}$: The bold cross indicates which hierarchical increments belong to which component $u_{\boldsymbol{k}}$.

with respect to the subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ and represent the decomposition (35) as

$$u = \sum_{\omega \subseteq \{1,\ldots,d\}} u_{\boldsymbol{k}}^{\omega} \tag{37}$$

for a sparse grid function $u \in \mathcal{V}_{\ell}^{(1)}$ and $\boldsymbol{k} \in \mathcal{L}$.

Let $u = \tilde{u} + \hat{u}$ be the splitting with respect to $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ of the previous Sec. 3.3. If we compare this to the dimension-wise splitting (37), we see that the first component $u_{\boldsymbol{k}}^{\emptyset}$ of (37) equals $\tilde{u}$ and the sum of all other $2^d - 1$ terms of (35) equals $\hat{u}$. Thus, we have

$$\tilde{u} = u_{\boldsymbol{k}}^{\emptyset}, \quad \hat{u} = \sum_{\omega \subseteq \{1,\ldots,d\}, \omega \neq \emptyset} u_{\boldsymbol{k}}^{\omega}\,. \tag{38}$$

The hierarchical increments corresponding to the four components $u_{\boldsymbol{k}}^{\{\emptyset\}}, u_{\boldsymbol{k}}^{\{1\}}, u_{\boldsymbol{k}}^{\{2\}}, u_{\boldsymbol{k}}^{\{1,2\}}$ of the decomposition of a two-dimensional sparse grid function $u \in \mathcal{V}_{\ell}^{(1)}$ are shown in Fig. 10. The bold cross indicates which hierarchical increments belong to which component. We see that we obtain a very natural decomposition of the hierarchical increment scheme. It also fits to the splitting shown in Fig. 9.

Nodal point basis in components Let $f \in \mathcal{V}_{\ell}$ be a function with

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{l} \leq \boldsymbol{k}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \tag{39}$$

where the basis functions are the hierarchical basis functions. Because all hierarchical increments with level $\boldsymbol{l} \leq \boldsymbol{k}$ are used in (39), we can represent f in a nodal point basis $\tilde{\Phi}_{\boldsymbol{k}}$ of level $\boldsymbol{k}$ by evaluating the hierarchical basis functions at the grid points, cf (16). We obtain

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{l} \leq \boldsymbol{k}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) = \sum_{\boldsymbol{t}' \in \tilde{\mathcal{I}}_{\boldsymbol{k}}} \left(\sum_{\boldsymbol{l} \leq \boldsymbol{k}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \phi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}_{\boldsymbol{k},\boldsymbol{t}'}) \right) \phi_{\boldsymbol{k},\boldsymbol{t}'}(\boldsymbol{x}) \tag{40}$$

where $\tilde{\mathcal{I}}_{\boldsymbol{k}}$ contains the indices of the nodal point basis functions in $\tilde{\Phi}_{\boldsymbol{k}}$.

A component $u_{\boldsymbol{k}}^{\omega}$ has $l_j \leq k_j$ for all $j \in \{1, \ldots, d\} \setminus \omega$, cf. (36). This means, we can replace the hierarchical basis functions with the nodal point basis functions in directions $\{1, \ldots, d\} \setminus \omega$ similar to (40) and obtain

$$u_{\boldsymbol{k}}^{\omega}(\boldsymbol{x}) = \sum_{\boldsymbol{t}' \in \tilde{\mathcal{I}}_{\boldsymbol{k}_{\{1,\ldots,d\}\setminus\omega}}} \sum_{\substack{\boldsymbol{l} \in \mathcal{L} \\ \forall i \in \omega : l_i > k_i \\ \forall i \notin \omega : l_i \leq k_i}} \underbrace{\sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j \in \{1,\ldots,d\}\setminus\omega} \phi_{l_j,i_j}(x_{k_j,t'_j})}_{\text{new coefficients}} \underbrace{\prod_{j \in \omega} \phi_{l_j,i_j}(x_j)}_{\substack{\text{hierarchical basis} \\ \text{in directions } \omega}} \underbrace{\prod_{j \in \{1,\ldots,d\}\setminus\omega} \phi_{k_j,t'_j}(x_j)}_{\substack{\text{nodal point basis} \\ \text{in directions } \{1,\ldots,d\}\setminus\omega}} \tag{41}$$

where $\tilde{\mathcal{I}}_{\boldsymbol{k}_{\{1,\ldots,d\}\setminus\omega}} = \left\{ \boldsymbol{i} \in \tilde{\mathcal{I}}_{\boldsymbol{k}} | \forall j \in \omega : i_j = 1 \right\}$ contains the indices of the nodal point basis functions. We do not achieve such a clear separation into coefficients and basis functions as in (40) because we only partially replace the hierarchical basis with a nodal point basis in the component $u_{\boldsymbol{k}}^{\omega}$. However, we can move the product with the nodal point basis functions in front of the sum over all levels because it does not depend on the level $\boldsymbol{l}$ anymore

$$u_{\boldsymbol{k}}^{\omega}(\boldsymbol{x}) = \sum_{\boldsymbol{t}' \in \tilde{\mathcal{I}}_{\boldsymbol{k}_{\{1,\ldots,d\}\setminus\omega}}} \prod_{j \in \{1,\ldots,d\}\setminus\omega} \phi_{k_j,t'_j}(x_j) \sum_{\substack{\boldsymbol{l} \in \mathcal{L} \\ \forall i \in \omega : l_i > k_i \\ \forall i \notin \omega : l_i \leq k_i}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j \in \{1,\ldots,d\}\setminus\omega} \phi_{l_j,i_j}(x_{k_j,t'_j}) \prod_{j \in \omega} \phi_{l_j,i_j}(x_j) \tag{42}$$

With (42) we found a representation of the component $u_{\boldsymbol{k}}^{\omega}$ with hierarchical basis functions in directions ω and nodal point basis functions in directions $\{1, \ldots, d\} \setminus \omega$.

Bilinear form and components We plug the ANOVA-like decomposition (37) with respect to the hierarchical subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ of level $\boldsymbol{k}$ into the Galerkin projection (31) and obtain

$$a_{\mathrm{A}}(u, \psi_{\boldsymbol{k},\boldsymbol{t}}) = \sum_{\omega \subseteq \{1,\ldots,d\}} a_{\mathrm{A}}(u_{\boldsymbol{k}}^{\omega}, \psi_{\boldsymbol{k},\boldsymbol{t}}) = 0, \qquad \forall \psi_{\boldsymbol{k},\boldsymbol{t}} \in \Psi_{\boldsymbol{k}}, \tag{43}$$

where $\Psi_{\boldsymbol{k}}$ contains the test functions corresponding to the hierarchical subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$. With the representation (42) we can write one component of (43) as

$$a_{\mathrm{A}}(u_{\boldsymbol{k}}^{\omega}, \psi_{\boldsymbol{k},\boldsymbol{t}}) = \sum_{\boldsymbol{t}' \in \tilde{\mathcal{I}}_{\boldsymbol{k}_{\{1,\ldots,d\}\setminus\omega}}} \prod_{j \in \{1,\ldots,d\}\setminus\omega} a_{\mathrm{A}^{(j)}}(\phi_{k_j,t'_j}, \psi_{k_j,t_j}) \sum_{\substack{\boldsymbol{l} \in \mathcal{L} \\ \forall i \in \omega : l_i > k_i \\ \forall i \notin \omega : l_i \leq k_i}} \sum_{\boldsymbol{i} \in \hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j \in \{1,\ldots,d\}\setminus\omega} \phi_{l_j,i_j}(x_{k_j,t'_j}) \prod_{j \in \omega} a_{\mathrm{A}^{(j)}}(\phi_{l_j,i_j}, \psi_{k_j,t_j}) \tag{44}$$

where $a_{\mathrm{A}^{(j)}}$ is the one-dimensional bilinear form corresponding to the one-dimensional operator $\mathrm{A}^{(j)}$, see Sec. 3.1. To simplify (44), we define for each test function $\psi_{\boldsymbol{k},\boldsymbol{t}}$ and for each component $u^{\omega}_{\boldsymbol{k}}$ the ANOVA quantity

$$r^{\omega}_{\boldsymbol{k},\boldsymbol{t}} = \sum_{\substack{\boldsymbol{l}\in\mathcal{L} \\ \forall i\in\omega: l_i > k_i \\ \forall i\notin\omega: l_i \leq k_i}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j\in\{1,\dots,d\}\setminus\omega} \phi_{l_j,i_j}(x_{k_j,t_j}) \prod_{j\in\omega} a_{\mathrm{A}^{(j)}}(\phi_{l_j,i_j}, \psi_{k_j,t_j})\,. \tag{45}$$

The one-dimensional bilinear form $a_{\mathrm{A}^{(j)}}(\phi_{k_j,t_j}, \psi_{k_j,t_j})$ with the nodal point basis functions can be computed by applying a one-dimensional stencil $\bigstar_{k_j,t_j}$. If we have such a stencil for each direction $j \in \{1,\dots,d\}\setminus\omega$ where a nodal point basis is used, we can write (44) as

$$a_{\mathrm{A}}(u^{\omega}_{\boldsymbol{k}}, \psi_{\boldsymbol{k},\boldsymbol{t}}) = \bigotimes_{j\in\{1,\dots,d\}\setminus\omega} \bigstar_{k_j,t_j}(r^{\omega}_{\boldsymbol{k}}) \tag{46}$$

where the stencil $\bigotimes_{j\in\{1,\dots,d\}\setminus\omega} \bigstar_{k_j,t_j}$ operates on $r^{\omega}_{\boldsymbol{k}}$ at the grid point x_{k_j,t_j} in direction j. We refer with $r^{\omega}_{\boldsymbol{k}}$ to all $r^{\omega}_{\boldsymbol{k},\boldsymbol{t}}$ with $\boldsymbol{t} \in \tilde{\mathcal{I}}_{\boldsymbol{k}}$. Note that $r^{\emptyset}_{\boldsymbol{k},\boldsymbol{t}} = u(\boldsymbol{x}_{\boldsymbol{k},\boldsymbol{t}})$ holds.

Equation (46) plays a crucial role in our multigrid method because it allows us to efficiently compute the right-hand side. Recall the splitting $u = \tilde{u} + \hat{u}$ and the corresponding right-hand side $a_{\mathrm{A}}(\hat{u}, \psi_{\boldsymbol{k},\boldsymbol{t}})$ of the Galerkin equations (31). With (38) and (46) we see that the right-hand side can be computed with a constant number of stencil applications to the ANOVA quantities $r^{\omega}_{\boldsymbol{k}}$ for all $\omega \subseteq \{1,\dots,d\}$. Thus, we need an algorithm to compute all $r^{\omega}_{\boldsymbol{k}}$ and to keep them up-to-date during grid traversals.

4.2. Multilevel Algorithm

So far we have seen that we have to provide the values $r^{\omega}_{\boldsymbol{k}}$ to compute the right-hand side $a(\hat{u}, \psi_{\boldsymbol{k},\boldsymbol{t}})$ of the system of linear equations corresponding to the Galerkin projection into the subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$. We store the values $r^{\omega}_{\boldsymbol{k}}$ and thus need an algorithm to efficiently update them while moving through the hierarchical subspace scheme. Efficiently means that we want to exploit the hierarchical structure and that we do not want to compute everything from scratch, in particular, we do not want to iterate over all grid points when we change from one grid to another.

In Sec. 4.2.1 we present the in-place storage scheme to store the values $r^{\omega}_{\boldsymbol{k}}$. The COARSEN and the REFINE procedures to change to a coarser or finer grid are introduced in Sec. 4.2.2. These procedures rely on the concept of updating and preserving the ANOVA quantities during a change from subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$. Finally, in Sec. 4.2.3 we show that the runtime and the storage complexity of the COARSEN and REFINE procedures scale only linearly with the number of grid points.

4.2.1. In-Place Storage Scheme

Our in-place storage scheme stores two values $r^{\omega}_{\boldsymbol{k},\boldsymbol{t}}$ and $r^{\omega}_{\boldsymbol{k}',\boldsymbol{t}'}$ at the same memory location if the corresponding grid points $\boldsymbol{x}_{\boldsymbol{k},\boldsymbol{t}}$ and $\boldsymbol{x}_{\boldsymbol{k}',\boldsymbol{t}'}$ are equal. Note that for example the grid point $\boldsymbol{x}_{\boldsymbol{k},\boldsymbol{t}}$ equals $\boldsymbol{x}_{(k_1,\dots,k_{\tilde{d}}-1,\dots,k_d),(t_1,\dots,3\cdot t_{\tilde{d}},\dots,t_d)}$ but $r^{\omega}_{\boldsymbol{k},\boldsymbol{t}}$ is not equal $r^{\omega}_{(k_1,\dots,k_{\tilde{d}}-1,\dots,k_d),(t_1,\dots,3\cdot t_{\tilde{d}},\dots,t_d)}$ because the test function in direction $\tilde{d}$ is different.

With this in-place storage scheme we have exactly one array of the same size as the coefficient vector $\boldsymbol{\alpha} \in \mathbb{R}^N$ for each $\omega \subseteq \{1, \ldots, d\}$. Whereas $r^{\omega}_{\boldsymbol{k},\boldsymbol{t}}$ with the subscript level $\boldsymbol{k}$ and subscript index $\boldsymbol{t}$ denotes the value (45), $[r]^{\omega}$ without the subscripts denotes the array corresponding to a subset $\omega \subseteq \{1, \ldots, d\}$. We always define $[r]^{\emptyset} = \boldsymbol{u}$ where $\boldsymbol{u} \in \mathbb{R}^N$ contains the values of the solution function u^N at the sparse grid points.

Since we always start at the coarsest subspace $\tilde{\mathcal{H}}_{(1,1,\ldots,1)}$ all hierarchical coefficients $\alpha_{\boldsymbol{l},\boldsymbol{i}}$ with $\boldsymbol{l} \succ (1, 1, \ldots, 1)$ are zero at the beginning. Hence, all $[r]^{\omega}$ with $\omega \subseteq \{1, \ldots, d\}$ are zero. Inhomogeneous Dirichlet boundary conditions might be imposed on $[r]^{\emptyset} = \boldsymbol{u}$ step by step as the grid hierarchy is traversed.

4.2.2. Update of ANOVA Quantities and Multilevel Algorithm

Let us consider a coarsening step from $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$ with $\boldsymbol{k} = (k_1, \ldots, k_d)$ and $\boldsymbol{k}' = (k_1, \ldots, k_{\tilde{d}} - 1, \ldots, k_d)$. We coarsen in direction $\tilde{d}$. We assume that the solution function u is represented as $\tilde{u} + \hat{u}$ with respect to $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ and that the ANOVA quantities $r^{\omega}_{\boldsymbol{k}}$ are stored in $[r]^{\omega}$ for all $\omega \subseteq \{1, \ldots, d\}$. Furthermore, we call the grid points corresponding to $\tilde{\mathcal{H}}_{\boldsymbol{k}} \backslash \tilde{\mathcal{H}}_{\boldsymbol{k}'}$ and $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$ the fine and coarse grid points (with respect to level $\boldsymbol{k}$), respectively.

We want to exploit the hierarchical structure imposed by the hierarchical basis to update the values in $[r]^{\omega}$. This requires two steps. First, the values $[r]^{\omega}$ are changed at the coarse grid points to contain $r^{\omega}_{\boldsymbol{k}'}$. Second, the values $[r]^{\omega}$ at the fine grid points are hierarchised and thus preserved for their later use.

Update at coarse grid points Let us first consider the one-dimensional case where we change from subspace $\tilde{\mathcal{H}}_k$ to $\tilde{\mathcal{H}}_{k-1}$. At the coarse grid points we have the ANOVA quantities

$$r^{\{1\}}_{k,t} = \sum_{l>k} \sum_{i \in \hat{\mathcal{I}}_l} \alpha_{l,i} a_{\mathrm{A}}(\phi_{l,i}, \psi_{k,t})$$

stored in $[r]^{\omega}$. The goal is to compute $r^{\{1\}}_{k-1,t}$ with respect to test function $\psi_{k-1,t}$ of level $k-1$, i.e.,

$$r^{\{1\}}_{k-1,t} = \sum_{l>k-1} \sum_{i \in \hat{\mathcal{I}}_l} \alpha_{l,i} a_{\mathrm{A}}(\phi_{l,i}, \psi_{k-1,t})\,.$$

We first have a look at how $r^{\{1\}}_{k-1}$ and $r^{\{1\}}_{k}$ are related. Therefore, we split $r^{\{1\}}_{k-1,t}$ into two terms

$$r^{\{1\}}_{k-1,t} = \sum_{i \in \hat{\mathcal{I}}_k} \alpha_{k,i} a_{\mathrm{A}}(\phi_{k,i}, \psi_{k-1,t}) + \sum_{l>k} \sum_{i \in \hat{\mathcal{I}}_l} \alpha_{l,i} a_{\mathrm{A}}(\phi_{l,i}, \psi_{k-1,t}). \tag{47}$$

The first term includes only basis functions of level k and the second term the remaining ones of level greater than k. The first sum includes only four summands because a test function of level $k-1$ has overlapping support only with four basis functions of level k. Note that we employ sparse grids with coarsening by three, see Sec. 2.4. The second sum is analogous to $r^{\{1\}}_{k,t}$ except that now the test function $\psi_{k-1,t}$ of level $k-1$ instead of $\psi_{k,3t}$ of level k is used. However, we can represent a test function of level $k-1$ as a linear combination of test functions of level k as in (15) and as shown in Fig. 4. Putting all this together, we can rewrite (47) as

$$\begin{aligned} r_{k-1,t}^{\{1\}} &= \alpha_{k,3t-2} a_{\mathrm{A}}(\phi_{k,3t-2}, \psi_{k-1,t}) + \alpha_{k,3t-1} a_{\mathrm{A}}(\phi_{k,3t-1}, \psi_{k-1,t}) \\ &+ \alpha_{k,3t+1} a_{\mathrm{A}}(\phi_{k,3t+1}, \psi_{k-1,t}) + \alpha_{k,3t+2} a_{\mathrm{A}}(\phi_{k,3t+2}, \psi_{k-1,t}) \\ &+ \tfrac{1}{3}\left(r_{k,3t-2}^{\{1\}} + r_{k,3t+2}^{\{1\}}\right) + \tfrac{2}{3}\left(r_{k,3t-1}^{\{1\}} + r_{k,3t+1}^{\{1\}}\right) + r_{k,3t}^{\{1\}}, \end{aligned} \tag{48}$$

where we have explicitly written the four summands of the first sum and employed the linear combination (15) to compute the second sum. We emphasize that (48) does not contain a sum over the levels. This means we can compute the value $r_{k-1,t}^{\{1\}}$ by applying only two stencils: One stencil to $\tilde{u}$ and one stencil to the $r_k^{\{1\}}$ stored in $[r]^{\{1\}}$, i.e.,

$$[r]_{k-1,t}^{\{1\}} = \left[\begin{array}{ccccc} * & * & 0 & * & * \end{array}\right]_{k,3t}(\tilde{u}) + \frac{1}{3}\left[\begin{array}{ccccc} 1 & 2 & 3 & 2 & 1 \end{array}\right]_{k,3t}\left([r]^{\{1\}}\right). \tag{49}$$

Whereas the first stencil evaluates the bilinear forms a_{A} which are now additionally needed on level $k-1$, the second stencil operates on $r_k^{\{1\}}$ stored in $[r]^{\{1\}}$ at the fine grid points to compute $r_{k-1}^{\{1\}}$. The first stencil depends on the current operator A, the second stencil depends on the test functions and is derived from (15).

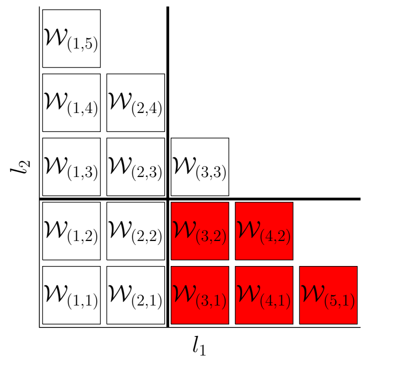
We emphasize once more that for this update we do not have to run through all basis functions with level greater than k, but that we exploit the hierarchical structure of the hierarchical basis functions by reusing $r_k^{\{1\}}$ stored in $[r]^{\{1\}}$ at the fine grid points.

In the multi-dimensional case, we can apply the one-dimensional procedure in each dimension separately because our basis and test functions as well as our operators have tensor product structure. This also means that for the update of $[r]^{\omega}$ in direction i we can already use the values $[r]^{\omega\setminus\{i\}}$. Thus, just as in the one-dimensional case, only one-dimensional stencils are needed to update a multi-dimensional component $[r]^{\omega}$.

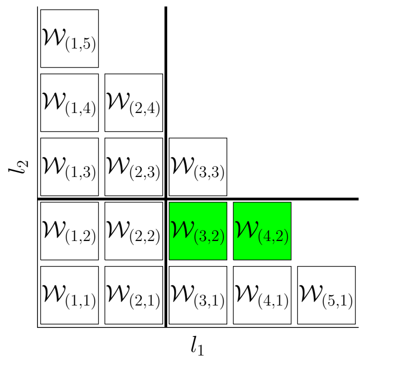
Preservation at fine grid points In the coarsening step from $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$, the values in $[r]^{\omega}$ are updated at the coarse grid points with the one-dimensional stencils to obtain the ANOVA quantities with respect to level $\boldsymbol{k}'$. The values in $[r]^{\omega}$ at the fine grid points are not needed to relax the equations corresponding to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$ but we want to keep them because they might be needed later on when we come back to $\tilde{\mathcal{H}}_{\boldsymbol{k}}$. However, the values $r_{\boldsymbol{k}}^{\omega}$ stored at the fine grid points depend on coefficients corresponding to hierarchical increments $\mathcal{W}_{\boldsymbol{l}}$ with $\boldsymbol{l} \leq \boldsymbol{k}'$. This means, if we relax the equations corresponding to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$ and thus change the coefficients, the values $r_{\boldsymbol{k}}^{\omega}$ change too and our stored values in $[r]^{\omega}$ become obsolete. That is why we do not store $r_{\boldsymbol{k}}^{\omega}$ but the hierarchised counterpart

$$\hat{r}_{\boldsymbol{k},\boldsymbol{t}}^{\omega} = \sum_{\substack{\boldsymbol{l}\in\mathcal{L} \\ \forall i\in\omega: l_i > k_i \\ \forall i\in\{1,\ldots,d\}\setminus\omega: l_i = k_i}} \sum_{\boldsymbol{i}\in\hat{\mathcal{I}}_{\boldsymbol{l}}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j\in\{1,\ldots,d\}\setminus\omega} \phi_{k_j,i_j}(x_{k_j,t_j}) \prod_{j\in n} a_{\mathrm{A}^{(j)}}(\psi_{k_j,t_j}, \phi_{l_j,i_j}), \tag{50}$$

where $r_{\boldsymbol{k}}^{\omega}$ has been hierarchised in all directions in $\{1,\ldots,d\}\setminus\omega$. In Fig. 10 and by comparing (45) with (50) we clearly see that the hierarchisation removes the dependence



(a) dehierarchised ANOVA quantity $r^{\{1\}}_{(2,2)}$



(b) hierarchised ANOVA quantity $\hat{r}^{\{1\}}_{(2,2)}$

Figure 11: Shows the hierarchical increments which are included in $r^{\{1\}}_{(2,2)}$ (red) and those included in $\hat{r}^{\{1\}}_{(2,2)}$ (blue). The hierarchised quantity $\hat{r}^{\{1\}}_{(2,2)}$ depends only on the hierarchical increments with level two in direction 2 and is thus independent from those with level one.

on the coefficients $\boldsymbol{\alpha}$ corresponding to basis functions with $l_j < k_j$ for all $j \in \{1, \dots, d\} \setminus \omega$. Thus, we can relax the equations corresponding to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$ and our $\hat{r}^{\omega}_{\boldsymbol{k}}$ values stored in $[r]^{\omega}$ at the fine grid points stay valid. To get $r^{\omega}_{\boldsymbol{k}}$ in a later relaxation, we only have to dehierarchise these values again.

Multigrid procedure Algorithm 1 summarizes the steps to keep the values in $[r]^{\omega}$ up-to-date when coarsening from $\tilde{\mathcal{H}}_{\boldsymbol{k}}$ to $\tilde{\mathcal{H}}_{\boldsymbol{k}'}$. It combines the update and the preservation step of ANOVA quantities developed in the previous two paragraphs. Note that the REFINE procedure is not explicitly formulated because it is just the inverse of the COARSEN procedure.

First, the update and preservation step is applied in direction $\tilde{d}$ of the coarsening step. If $\tilde{d} \in \omega$ then the $r^{\omega}_{\boldsymbol{k}}$ includes the bilinear form in direction $\tilde{d}$ and thus the value in $[r]^{\omega}$ has to be updated at the coarse grid points. Nothing has to be done at the fine grid points because we cannot hierarchise the values $[r]^{\omega}$ in directions in ω. If $\tilde{d} \notin \omega$, then we do not have to update the values $[r]^{\omega}$ at the coarse grid points because they do not include a bilinear in that direction but we have to hierarchise them at the fine grid points.

In all other directions $\overline{d} \in \{1, \dots, d\} \setminus \{\tilde{d}\}$ the same steps are executed. However, because we perform the coarsening in direction $\tilde{d}$ we remove grid points of only one level in direction $\tilde{d}$. This is not the case for all other dimensions in $\{1, \dots, d\} \setminus \{\tilde{d}\}$. There we have to deal with grid points of different levels, see Fig. 12b and Fig. 12c. We cannot hierarchise grid points of different levels at once but have to process them level by level. That is why we introduce an additional loop $\overline{k}'_{\overline{d}} = k'_{\overline{d}}, \dots, 1$ from level $k'_{\overline{d}}$ to level 1. To be consistent with this additional loop we have to further split the fine grid points into

Algorithm 1 coarsen in direction $\tilde{d}$

```
procedure COARSEN($\tilde{d}$)
    In direction $\tilde{d}$
    for $\omega \subseteq \{1, \dots, d\}$ do
        if $\tilde{d} \in \omega$ then compute $r^{\omega}_{\boldsymbol{k}'}$
            Update $[r]^{\omega}$ with stencils (49) at coarse grid points
        else if $\tilde{d} \notin \omega$ then transform $r^{\omega}_{\boldsymbol{k}}$ to $\hat{r}^{\omega}_{\boldsymbol{k}}$
            Hierarchise $[r]^{\omega}$ at fine grid points in direction $\tilde{d}$
        end if
    end for

    In all other directions
    for $\overline{d} \in \{1, \dots, d\} \setminus \{\tilde{d}\}$ do
        for $\overline{\boldsymbol{k}'} = (k'_1, \dots, k'_{\overline{d}}, \dots, k'_d)$ to $(k'_1, \dots, 1, \dots, k'_d)$ do
            for $\omega \subseteq \{1, \dots, d\}$ do
                if $\overline{d} \in \omega$ then compute $r^{\omega}_{\boldsymbol{k}'}$
                    Update $[r]^{\omega}$ with stencils (49) at grid points $\mathcal{C}_{\overline{k}'_{\overline{d}}}$
                else if $\tilde{d} \notin \omega$ then transform $r^{\omega}_{\boldsymbol{k}}$ to $\hat{r}^{\omega}_{\boldsymbol{k}}$
                    Hierarchise $[r]^{\omega}$ at grid points $\mathcal{F}_{\overline{k}'_{\overline{d}}}$ in direction $\overline{d}$
                end if
            end for
        end for
    end for
end procedure
```

the sets $\mathcal{F}_{\overline{k}'_{\overline{d}}}$ and $\mathcal{C}_{\overline{k}'_{\overline{d}}}$ corresponding to the spaces

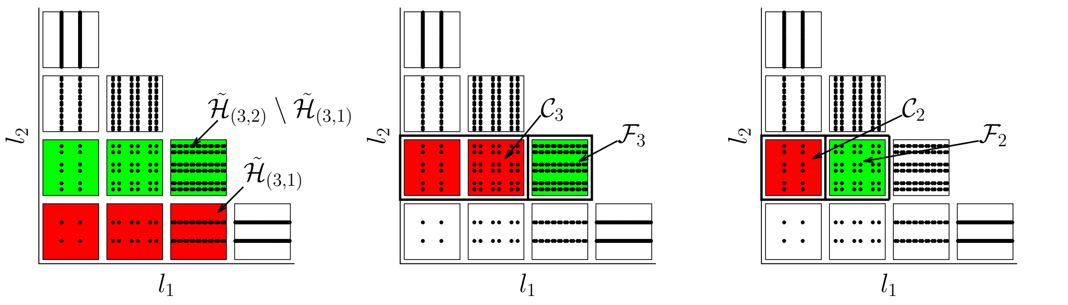
$$\bigcup_{\boldsymbol{l} \leq \boldsymbol{k},\, l_{\tilde{d}} = k_{\tilde{d}},\, l_{\overline{d}} < \overline{k}'_{\overline{d}}} \mathcal{W}_{\boldsymbol{l}}, \quad \text{and,} \quad \bigcup_{\boldsymbol{l} \leq \boldsymbol{k},\, l_{\tilde{d}} = k_{\tilde{d}},\, l_{\overline{d}} = \overline{k}'_{\overline{d}}} \mathcal{W}_{\boldsymbol{l}}, \tag{51}$$

respectively, see Fig. 12.

4.2.3. Runtime and Storage Complexity

In this section, we show that the runtime and the storage complexity of the COARSEN and REFINE procedure is only linear in the number of grid points. Besides that we discuss how the structure and properties of the ANOVA components allow us to decrease the storage requirements even further.

Let us first consider the runtime complexity. Executing a multigrid cycle means that we have to provide the COARSEN and REFINE procedure to go from one grid or subspace to another, as well as a relaxation method to reduce the residual corresponding to the current subspace. Both, the grid transfer and the relaxation, influence the runtime of one multigrid iteration.


$\tilde{\mathcal{H}}_{(3,2)} \setminus \tilde{\mathcal{H}}_{(3,1)}$
$\tilde{\mathcal{H}}_{(3,1)}$
l_2
l_1
$\mathcal{C}_3$
$\mathcal{F}_3$
$\mathcal{C}_2$
$\mathcal{F}_2$

(a) coarse/fine grid points (b) further partition of fine grid points into $\mathcal{C}_3$ and $\mathcal{F}_3$ (c) final partition of $\mathcal{C}_3$ into $\mathcal{C}_2$ and $\mathcal{F}_2$

Figure 12: In (a) we show the fine (green) and coarse (red) grid points for the coarsening step from $\tilde{\mathcal{H}}_{(3,2)}$ to $\tilde{\mathcal{H}}_{(3,1)}$. For the case $\overline{d} \in \{1, \ldots, d\} \setminus \{\tilde{d}\}$ in Alg. 1 we further partition the fine grid points as shown in (b) and (c).

The relaxation method, e.g., Jacobi or Gauss-Seidel, computes the residual of all equations corresponding to the current subspace and then updates the solution. Of course, the computation of the residual is the computationally expensive part, because the bilinear form $a_{\mathrm{A}}(u, \psi_{\boldsymbol{k},\boldsymbol{t}})$ has to be evaluated for every test function, i.e., for every grid point corresponding to the current subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$. With the ANOVA-like decomposition (43), the bilinear form $a_{\mathrm{A}}(u, \psi_{\boldsymbol{k},\boldsymbol{t}})$ is represented as a sum of 2^d terms associated to the ANOVA quantities $r^{\omega}_{\boldsymbol{k}}$. Each of these 2^d terms can be evaluated by applying a stencil of dimension $d - |\omega|$ to the corresponding $[r]^{\omega}$ where $|\omega|$ is the cardinality of the set ω. Since there are $\binom{d}{|\omega|}$ quantities $r^{\omega}_{\boldsymbol{k}}$ and a stencil of dimension i needs 3^i operations, we require

$$\sum_{i=0}^{d} \binom{d}{i} 3^i = 4^d$$

operations to evaluate the bilinear form $a_{\mathrm{A}}(u, \psi_{\boldsymbol{k},\boldsymbol{t}})$. Hence, the relaxation on the whole grid with level $\boldsymbol{k}$ is in $\mathcal{O}(4^d \cdot N_{\boldsymbol{k}})$ where $N_{\boldsymbol{k}} = |\tilde{\Phi}_{\boldsymbol{k}}|$ is the number of test and basis functions corresponding to the current subspace $\tilde{\mathcal{H}}_{\boldsymbol{k}}$, respectively.

The runtime complexity for the COARSEN and REFINE procedure can be determined by analyzing Alg. 1. In all d directions, we have to update all 2^d components $[r]^{\omega}$. Depending on whether only a hierarchisation or a real update is needed, only a one- or two one-dimensional stencils have to be applied at all $N_{\boldsymbol{k}} = \prod_{i=1}^{d} \left(3^{k_i} + 1\right)$ grid points. Except for direction $\tilde{d}$, the hierarchical structure has to be taken into account which means we need a loop from level $k_{\overline{d}}$ to 1. In iteration j only grid points up to level j are processed which are

$$(3^j + 1) \cdot \prod_{i \neq \overline{d}} \left(3^{k_i} + 1\right)$$

grid points. If we consider all $j = k_{\overline{d}}, \ldots, 1$ iterations we obtain

$$\sum_{j=1}^{k_{\overline{d}}} (3^j + 1) \cdot \prod_{i \neq \overline{d}} \left(3^{k_i} + 1\right) = \left(\frac{3}{2} \cdot \left(3^{k_{\overline{d}}} - 1\right) + k_{\overline{d}}\right) \cdot \prod_{i \neq \overline{d}} \left(3^{k_i} + 1\right) \leq 3 \cdot N_{\boldsymbol{k}},$$

processed grid points. Overall, with the update of all 2^d components $[r]^\omega$ in all d directions, the COARSEN and REFINE procedure are in $\mathcal{O}(d \cdot 2^d \cdot N_{\boldsymbol{k}})$. Hence, their runtime is only linear in the number of grid points $N_{\boldsymbol{k}}$.

The storage requirements are dominated by the ANOVA quantities $r^\omega_{\boldsymbol{k}}$ rather than the coefficient vector $\boldsymbol{\alpha}$. Clearly, no component $[r]^\omega$ stores more values than the sparse grid corresponding to the space $\mathcal{V}^{(1)}_\ell$ of the solution vector $\boldsymbol{u}$. Thus, because we have 2^d components, we can bound the storage requirements by $\mathcal{O}(2^d \cdot |\mathcal{V}^{(1)}_\ell|)$. This becomes computationally infeasible for higher-dimensional problems very soon. However, the ANOVA-like decomposition (43) of our bilinear form ranks the components with respect to their importance for the solution, see, e.g., [88, 91] for an in-depth discussion. This ANOVA property has already been exploited in previous work. The multigrid method presented in [150] for two-dimensional problems simply ignores the last term $r^{\{1,2\}}_{\boldsymbol{k}}$ of (43) which tremendously simplifies the multigrid algorithm. For Poisson's equation in two dimensions ($\Delta = \partial^2_{x_1} \otimes \mathrm{I}_{x_2} + \mathrm{I}_{x_1} \otimes \partial^2_{x_2}$) these terms are zero. For the Helmholtz equation in two dimensions, it is shown that the solution still converges in a Sobolev norm even if the term $r^{\{1,2\}}_{\boldsymbol{k}}$ is ignored. However, to the author's knowledge, there are no such results available for problems in more than two dimensions. That is why we do not skip whole ANOVA components but only ignore very selected values of the quantities $r^\omega_{\boldsymbol{k}}$. Instead of allocating memory for 2^d sparse grids of level ℓ, we use an adaptive storage scheme where we only store $r^\omega_{\boldsymbol{k},\boldsymbol{t}}$ if its absolute value is greater than a threshold.

With a numerical example (Poisson's equation) we want illustrate the rapid decay of the quantities $r^\omega_{\boldsymbol{k}}$ and thus the memory savings due to the adaptive storage scheme. Note that the quantities $r^\omega_{\boldsymbol{k}}$ of our ANOVA-like decomposition not only include the solution u but also the bilinear form a_{A}. Hence, we combine the rapid decay of the hierarchical surpluses with the influence of the bilinear form on them. In Fig. 13 we show how many more grid points are needed to store all values, including all $r^\omega_{\boldsymbol{k}}$ with $\omega \subseteq \{1, \ldots, d\}$ as well as the solution vector $\boldsymbol{u}$ itself, and compare it to the number of grid points of a sparse grid. The threshold is set to 1e-14 so that the influence on the solution is below the discretization error. The result is that only 1.5 to 2.5 times the number of grid points of a sparse grid have to be kept. Because this holds for different levels ℓ and dimensions d, we can conclude that in practice the constant 2^d can be replaced by a small constant independent of level ℓ and dimension d.

5. Case Study: Multi-Dimensional Convection-Diffusion Equation

Finally, in this section, numerical examples are presented to confirm the practicability of the multigrid method of Sec. 4, as well as the expected multigrid performance. The

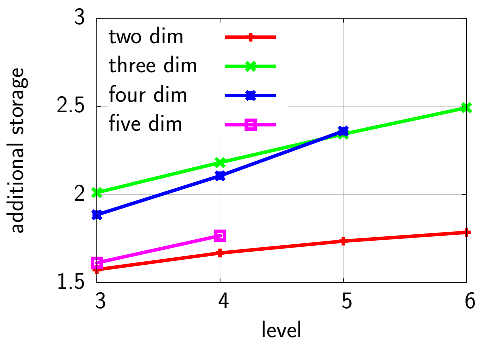


Figure 13: Between $1.5\cdot|\mathcal{V}_\ell^{(1)}|$ and $2.5\cdot|\mathcal{V}_\ell^{(1)}|$ grid points are required to store all ANOVA quantities. Thus, empirically, only $\mathcal{O}(|V_\ell^{(1)}|)$ grid points are needed to store the sparse grid solution $\boldsymbol{u}$ as well as all ANOVA quantities.

parametrized convection-diffusion equation is solved in a sparse grid space requiring distinctly fewer degrees of freedom as in a full grid space. We show results for Helmholtz and convection-diffusion equations on both, regular and adaptively refined sparse grids. In all examples, we perform two pre- and post-smoothing Gauss-Seidel relaxations and set the threshold of the adaptive storage scheme to 1e-14.

5.1. Problem Formulation

We consider boundary value problems

$$\begin{aligned} -\sum_{i,j=1}^{d} \frac{\partial}{\partial x_i}\left(\mu_{ij}^D \frac{\partial u}{\partial x_j}\right) + \sum_{i=1}^{d} \mu_i^C \frac{\partial u}{\partial x_i} + \mu u &= f \quad \text{in } \Omega \\ u &= g \quad \text{on } \Gamma, \end{aligned}$$

with domain $\Omega = (0,1)^d$ and boundary Γ, $g \in L_2(\Gamma)$, and $f \in L_2(\Omega)$. The coefficients $\mu_{ij}^D, \mu_i^C, \mu \in \mathbb{R}$ are the parameters. We assume the resulting differential equation to be elliptic. For the variational formulation of the boundary value problem, we need the bilinear form

$$a(u,v) = \int_\Omega \sum_{i,j=1}^{d} \mu_{ij}^D \frac{\partial u}{\partial x_i}\frac{\partial v}{\partial x_j} + \sum_{i=1}^{d} \mu_i^C \frac{\partial u}{\partial x_i} v + \mu uv \, \mathrm{d}x \tag{52}$$

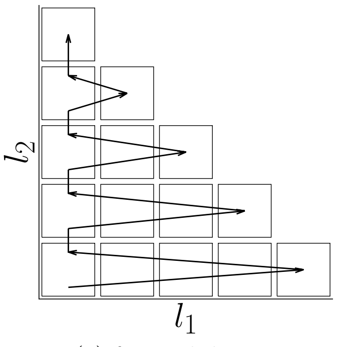
and the linear form

$$b(v) = (f,v)_{L_2} = \int_\Omega fv \, \mathrm{d}x \, .$$

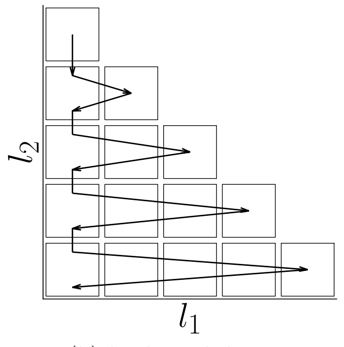
As discussed in Sec. 3.1 we can split the bilinear form (52) into a sum of bilinear forms which have tensor product structure.

Finally, we have to specify the stencils corresponding to the discretization of the one-dimensional operators on piecewise linear hat functions. These stencils are required for the computation (46) of the bilinear from with the ANOVA quantities. Let h_i be the mesh width in direction i, for the identity operator I_{x_i} we obtain the stencil

$$\frac{1}{6}h_i \left[\begin{array}{ccc} 1 & 4 & 1 \end{array}\right],$$



(a) forward direction



(b) backward direction

Figure 14: The path of the Q-cycle through the hierarchical subspaces is illustrated.

for the convection term ∂_{x_i} the stencil

$$\frac{1}{2}\left[\begin{array}{ccc} -1 & 0 & 1 \end{array}\right],$$

and for $\partial_{x_i}^2$

$$\frac{1}{h_i}\left[\begin{array}{ccc} -1 & 2 & -1 \end{array}\right],$$

see, e.g., [33].

5.2. Q-Cycle

We need to determine the multigrid cycle and thus the order in which we pass through the hierarchical scheme. We choose the so-called Q-cycle [151, 37] as shown in Fig. 14. Note that a Q-cycle can of course also be employed in case of full grids if semi-coarsening is possible, cf. [175] for a general discussion and [143, 145] for a discussion in the context of a multigrid method with the hierarchical basis and full grids.

In the d-dimensional case, the Q-cycle can be formulated by a recursive process. It consists of one-dimensional V-cycles which refine and coarsen only in one direction, see Alg. 2. The V-cycle moves only to finer grids as long as they exist in the current hierarchical scheme, i.e., as long as $\boldsymbol{k}$ is in $\mathcal{L}$ in Alg. 2. The Q-cycle then nests the V-cycles as shown in Alg. 3. The default value for parameter $\tilde{d}$ is the dimension d. This means, we start a Q-cycle by passing only the number of pre- and post-smoothing sweeps ν. The Q-cycle starts at the coarsest grid, then moves to finer grids, and ends at the coarsest grid again. Hence, first an approximation on a coarse grid is computed. It is then used on the next finer grid as a good initial guess. Because of the hierarchical basis, this can be seen as a kind of full multigrid (FMG) method which is typically the most efficient multigrid method, see [175, 31].

Algorithm 2 V-cycle

```
1: procedure VCYCLE($\tilde{d}, g, \nu$)
2:     for $\boldsymbol{k} = (k_1, \dots, k_{\tilde{d}-1}, 1, k_{\tilde{d}+1}, \dots, k_d)$ to $(k_1, \dots, k_{\tilde{d}-1}, \ell_{\tilde{d}-1}, k_{\tilde{d}+1}, \dots, k_d)$ do
3:         if $(k_1, \dots, k_{\tilde{d}} + 1, \dots, k_d) \notin \mathcal{L}$ then
4:             break
5:         end if
6:         Call method $g$ $\nu$ times
7:         Refine in direction $\tilde{d}$
8:     end for
9:     Call method $g$ $\nu$ times
10:    for $\boldsymbol{k} = (k_1, \dots, k_{\tilde{d}-1}, k_{\tilde{d}}, k_{\tilde{d}+1}, \dots, k_d)$ to $(k_1, \dots, k_{\tilde{d}-1}, 2, k_{\tilde{d}+1}, \dots, k_d)$ do
11:        Coarsen in direction $\tilde{d}$
12:        Call method $g$ $\nu$ times
13:    end for
14: end procedure
```

Algorithm 3 Q-cycle

```
1: procedure QCYCLE($\nu$, $\tilde{d} = d$)
2:     if $\tilde{d} == 1$ then
3:         VCYCLE($\tilde{d}$, GaussSeidel, $\nu$)
4:     else
5:         VCYCLE($\tilde{d}$, QCYCLE($\nu, \tilde{d} - 1$), $\nu$)
6:     end if
7: end procedure
```

5.3. Numerical Results

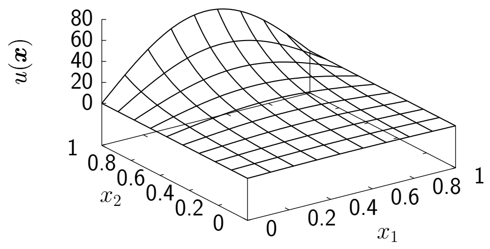
We first solve the Helmholtz and convection-diffusion equations on regular sparse grids and then show results for adaptive sparse grids.

Helmholtz equation As we have argued above in Sec. 4.2.3, the method presented in [151] cannot cope with all ANOVA components and has therefore already problems to employ the Q-cycle to compute the solution for the Helmholtz equation $\Delta u + \mu u = 0$ in only two dimensions. Because our method takes all ANOVA components into account, we can solve the Helmholtz equation in d dimensions. Let

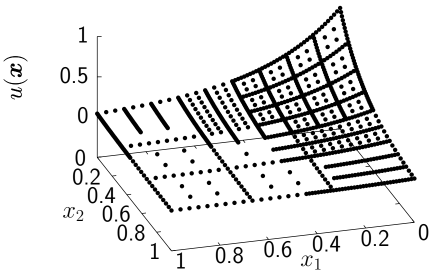
$$\begin{aligned} \Delta u + \mu u &= 0, \quad \text{in } \Omega, \\ u &= g, \quad \text{on } \Gamma \end{aligned} \tag{53}$$

be our Helmholtz problem with the parameter and boundary conditions

$$\mu = 2\pi\sqrt{d-1} + 1, \qquad g(\boldsymbol{x}) = \exp\left(\left(-\sqrt{d-1}\pi + \mu\right) x_d\right) \prod_{j=1}^{d-1} \sin\left(x_j \pi\right),$$



(a) solution surface



(b) solution surface on adaptive sparse grid

Figure 15: In (a) the surface of the solution of the Helmholtz problem (53) in two dimensions is shown. The surface of the solution of the convection-diffusion equation (55) in two dimensions on an adaptively refined sparse grid is plotted in (b).

dimension	level	grid points
$d = 2$	$\ell = 5$	3,160
$d = 2$	$\ell = 6$	10,936
$d = 3$	$\ell = 4$	12,880
$d = 3$	$\ell = 5$	51,760
$d = 4$	$\ell = 3$	34,912
$d = 4$	$\ell = 4$	180,064

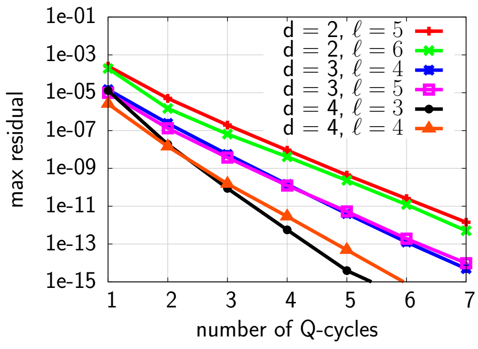
Table 1: Lists the number of points of sparse grids with coarsening by three in dimension two, three, and four. Note that a d-dimensional sparse grid with coarsening by three of level ℓ has more grid points than a sparse grid with coarsening by two of the same level and dimension.

and the solution

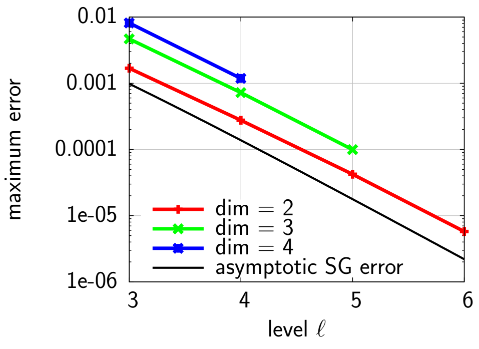
$$u(\boldsymbol{x}) = \exp\left(\left(-\sqrt{d-1}\pi + \mu\right) x_d\right) \prod_{j=1}^{d-1} \sin(x_j \pi)\,,$$

see Fig. 15.

In Fig. 16a the behavior of the residual with respect to the number of Q-cycles is shown for different levels and dimensions. A clear multigrid performance, i.e., convergence rate independent of the mesh width, can be seen. The residual is reduced below the discretization error with a fixed number of Q-cycles. We evaluated the obtained solution function at the grid points of a full grid of level three, determined the maximum error, and plotted the corresponding curve in Fig. 16b with the expected error rate $\mathcal{O}(h_\ell^2 \ell^{d-1})$, see Sec. 2.4 and [39]. The numerically obtained error rate (slope) is conform to the theoretical expected rate. Hence, even though the adaptive storage scheme ignores selected values of the ANOVA components, they do not deteriorate the accuracy of the sparse grid solution. Note the translation of the curves for dimensions two, three, and four in Fig. 16b due to the constants depending on the dimension d, see, again, [39].



(a) residual



(b) error

Figure 16: On the left, the residual with respect to the number of Q-cycles for the Helmholtz problem (53) is plotted. It shows that the convergence rate is independent of the mesh width. See Tab. 1 for the number of sparse grids. On the right, the corresponding maximum error and the expected asymptotic convergence rate is shown. The numerically obtained error rate (slope) for dimensions two to four is conform to the theoretically expected rate.

Convection-diffusion equation Let us now consider the convection-diffusion equation

$$\begin{aligned} -\Delta u + 10\partial_{x_d} u + \mu u = 0, &\quad \text{in } \Omega \\ u = g, &\quad \text{on } \Gamma \end{aligned} \tag{54}$$

with Dirichlet boundary conditions and

$$\mu = 10\sqrt{d-1}\pi, \qquad g(\boldsymbol{x}) = u(\boldsymbol{x}) = \exp\left(-\sqrt{d-1}\pi x_d\right) \prod_{j=1}^{d-1} \sin(x_j \pi).$$

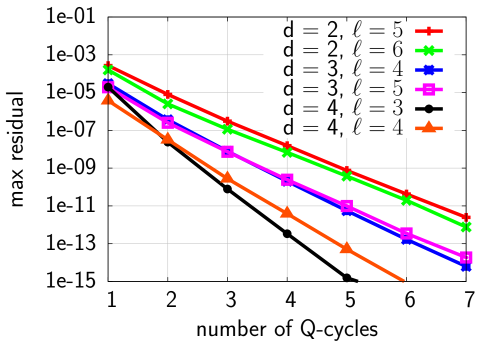
Again, a convergence rate independent from the mesh width is obtained, see Fig. 17a. The same holds for the convection diffusion equation

$$\begin{aligned} -\Delta u + 10\sum_{i=1}^{d} \partial_{x_i} u + \mu u = 0, &\quad \text{in } \Omega \\ u = g, &\quad \text{on } \Gamma \end{aligned} \tag{55}$$

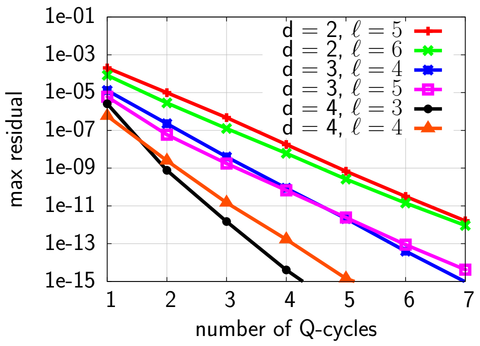
with

$$\lambda = d\pi^2 + 10d\pi, \qquad g(\boldsymbol{x}) = u(\boldsymbol{x}) = \prod_{j=1}^{d} \exp\left(-\pi x_j\right),$$

see Fig. 17b. In both cases, the residual is reduced slightly faster for $\ell = 3$ and $d = 4$ than for other levels and dimensions. The reason is, that $\ell = 3$ still describes a rather coarse grid and is as such not representative. The same can be observed for $\ell = 3$ and dimension $d = 2$ and $d = 3$.



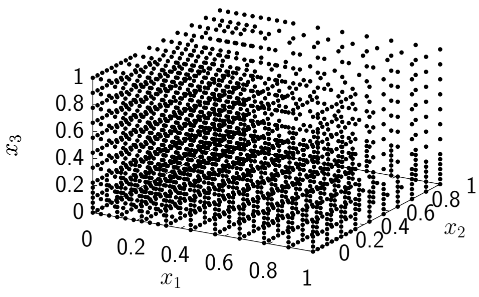
(a) residual for (54)



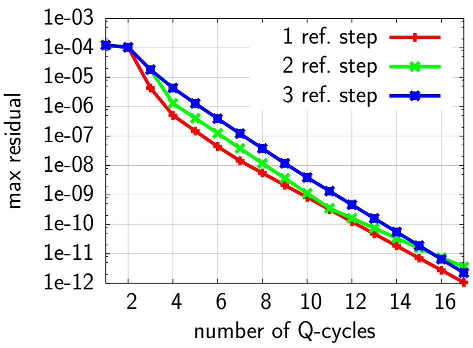
(b) residual for (55)

Figure 17: The convergence curve for the convection-diffusion equation (54) is plotted in (a) and for equation (55) in (b). The corresponding numbers of sparse grid points are shown in Tab. 1.

Convection-diffusion equation on adaptive sparse grids As a last example, we consider again the convection-diffusion equation (55) but on spatially refined sparse grids. In Fig. 15b the solution of the problem corresponding to the equation (55) in two dimensions on an adaptively refined sparse grid is shown. A sparse grid in three dimensions after one refinement step is plotted in Fig. 18a. The refinement has been determined by the commonly used surplus-based criterion which refines those grid points which have the greatest absolute hierarchical coefficients, see Sec. 2.4. In our example, one refinement step refines the first two percent of all grid points sorted by their absolute coefficient. For many sparse grid algorithms it is necessary that for each grid point all hierarchical ancestors exist, see, e.g., [154, 153]. In our case, the existence of all hierarchical ancestors means that we do not have any hanging nodes, i.e., it is guaranteed that every inner grid point is relaxed at least once during a Q-cycle. Hence, we do not have to care about a special hanging nodes treatment which is a clear advantage compared to common multigrid methods, cf., e.g., [175]. In Fig. 18b, the reduction of the residual with respect to the number of Q-cycles for grids with one, two, and three refinement steps in three dimensions is shown. Of course, more Q-cycles are needed to reduce the residual than in the case of regular sparse grids. This can also be observed for other multigrid methods [175]. The first refinement step adds many rather coarse grid points which have a big influence on the result. Thus, the residual stays almost constant from iteration one to iteration two. But already in refinement step two the residual can be distinctly decreased even though new grid points are still added. This is even more obvious for refinement step three. Overall, our proposed multigrid method reduces the residual at an equal rate no matter whether the grid is refined only once or several times.



(a) adaptive sparse grid



(b) residual on adaptive sparse grid

Figure 18: In (a) a refined sparse grid in three dimensions is shown. In (b) the convergence curves for the convection-diffusion equation (55) in three dimensions on adaptive sparse grids with one (2,296 points), two (6,840 points), and three (18,696 points) refinement steps are plotted. The convergence rate does not dependent on the number of refinement steps.

6. Remarks

We presented COARSEN and REFINE procedures for a multigrid method to solve multi-dimensional parametrized second-order elliptic PDEs on spatially adaptive sparse grids. When we traverse the hierarchical subspace scheme of a sparse grid with these COARSEN and REFINE procedures, it is guaranteed that we always have a valid system of linear equations corresponding to the Galerkin projection into the current subspace. Bundling the COARSEN and REFINE procedures with a multigrid cycle leads to a highly efficient multigrid method which only needs a fixed number of iterations independent from the mesh width to solve the problem.

Multigrid methods have to provide a valid system of linear equations at each grid of the grid hierarchy. This is particularly cumbersome in case of sparse grid discretizations because the right-hand side of the system of linear equations contains the hierarchical basis. Our multigrid method splits the right-hand side into multiple more easier manageable components. The key ingredient is the ANOVA-like decomposition of the sparse grid solution function leading to this component-wise representation of the bilinear form and the right-hand side. Each component of this representation can be computed by applying operator-dependent stencils to ANOVA quantities. Thus, the problem has been reduced from providing the right-hand side to providing the ANOVA quantities. These ANOVA quantities can be updated while moving through the grid hierarchy without iterating over all sparse grid points by exploiting the hierarchical structure of the hierarchical basis. We have shown that the update of the ANOVA components scales only linearly with the number of grid points and the storage complexity is linear in the number of grid points as well. Furthermore, by exploiting the structure of the ANOVA-like decomposition, we can reduce the constants in the complexity estimates even further.

We demonstrated our multigrid method on the multi-dimensional convection-diffusion equation on regular and spatially adapted sparse grids. The numerical experiments showed that our method indeed achieves multigrid performance and that the adaptive storage scheme which skips certain ANOVA quantities can greatly improve the storage requirements but does not deteriorate the error of the sparse grid solution.

Our multigrid method has to be modified to be applicable to elliptic PDEs with coefficient functions which do not have tensor product structure. Such coefficient functions lead to operators without tensor product structure and thus the ANOVA quantities cannot be updated anymore as described in the COARSEN and REFINE procedures. However, it is not only possible to decompose the solution function but also the coefficient function leading to two interwoven decompositions. In the general case, where any multigrid cycle through the hierarchical subspace scheme is allowed, it is not straightforward to carry our COARSEN and REFINE procedures over to these interwoven decompositions. But if the multigrid cycle is restricted to run only along the diagonal of the subspace scheme, many of the ANOVA components become zero and our update principle of the ANOVA quantities should be applicable again. However, this is part of future research. It is only clear that “it is not clear yet that it does not work” [42].

Part II.
MOR II: A Posteriori Model Order Reduction

In the previous part, we have discussed that sparse grids are, in a certain way, optimal to discretize functions with bounded, mixed derivatives up to order two. Thus, if the state variable $u^e(\boldsymbol{\mu}) \in \mathcal{U}^e$ corresponding to our problem $s : \mathcal{D} \to \mathcal{U}^e \to \mathcal{Y}$ satisfies this smoothness assumption, we can discretize $u^e(\boldsymbol{\mu})$ on a sparse grid and obtain a similar accuracy as on a full grid but with distinctly fewer degrees of freedom. The knowledge about the smoothness of $u^e(\boldsymbol{\mu})$ is determined by theory alone and not by experience in the form of data.

In this section, we assume our problem defined by the output function $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$ has already been solved for several parameter configurations in the past. The result is data. It might contain state vectors $\boldsymbol{u}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ and outputs of interest $\boldsymbol{y} \in \mathcal{Y}$. To find a more cost-efficient representation of the input-output relationship s, we extract characteristics of the problem from this data. As laid out in Sec. 2.3, we assume such a low-cost representation exists. We consider the data as our experience and thus call this procedure *a posteriori* model order reduction.

This is the setting of most common model order reduction methods, see Sec. 2.3.2. For instance, consider projection-based methods, e.g., POD and RBM. There, a set of snapshots $\mathcal{S} = \{\boldsymbol{u}(\boldsymbol{\mu}_1), \ldots, \boldsymbol{u}(\boldsymbol{\mu}_M)\} \subset \mathbb{R}^{\mathcal{N}}$ is computed by solving $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$ for a set of parameters $\mathcal{P} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M\} \subset \mathcal{D}$. The snapshots in $\mathcal{S}$ are then used to construct the reduced space $\mathcal{U}_n$ to approximate the space $\mathcal{U}^{\mathcal{N}}$. Because $n \ll \mathcal{N}$, we can compute a reduced solution $u_n(\boldsymbol{\mu}) \in \mathcal{U}_n$ for a parameter $\boldsymbol{\mu} \in \mathcal{D}$ distinctly faster than the solution $u^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathcal{U}^{\mathcal{N}}$ of the full model. This in turn allows us to rapidly evaluate the (reduced) output function $s_n : \mathcal{D} \to \mathcal{U}_n \to \mathcal{Y}$.

In the context of *a posteriori* model reduction, we cannot only employ projection-based but also data-fit methods such as interpolation and regression. The parameters in $\mathcal{P} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_M\}$ are the interpolation points and depending on whether we want to interpolate the whole state vector or just the outputs of interest, the snapshots in $\{\boldsymbol{u}(\boldsymbol{\mu}_1), \ldots, \boldsymbol{u}(\boldsymbol{\mu}_M)\} \subset \mathbb{R}^{\mathcal{N}}$ or the outputs in $\{s(\boldsymbol{\mu}_1), \ldots, s(\boldsymbol{\mu}_M)\} \subset \mathcal{Y}$ become the target values. Note that if the number of degrees of freedom $\mathcal{N}$ of the state variable $u^{\mathcal{N}}(\boldsymbol{\mu})$ is very large, the interpolation of the whole state vector $\boldsymbol{u}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ might become prohibitively expensive, see Sec. 8 for more details.

A posteriori model order reduction is split into an Offline and an Online phase. During the Offline phase, the data is generated and the reduced model is built. This is usually very expensive because to obtain the snapshots (data), the problem has to be solved multiple times in the high-dimensional space $\mathcal{U}^{\mathcal{N}}$. The Offline phase can also be seen as the learning part in the sense that we learn the reduced model from the data. Note that the Offline phase does not exist in *a priori* model order reduction, as is discussed in Part I. In the Online phase, we employ the reduced model to evaluate the input-output relationship $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$. It is important that we construct the reduced model only once but then use it over and over again. Thus, we compensate the high pre-processing or Offline costs by a large number of cheap Online evaluations, see Sec. 2.3.

In principle, the *a posteriori* model order reduction task is a supervised learning problem: We learn the input-output relationship $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$ from pre-computed data where we consider the parameters in $\mathcal{P}$ as data points and the components of the snapshots or the outputs of interest as target values. The generation of the data and the training of the classifier or regression function can be considered as the Offline phase, and the prediction of the target value for a new point as the Online phase. Data-fit methods implement directly this point of view.

In this part, we discuss how we can employ supervised learning techniques based on sparse grids for *a posteriori* model order reduction. We first introduce several novel supervised learning methods based on sparse grids in Sec. 7 and evaluate them on numerous classical learning problems. We then show how to employ sparse grid interpolation and sparse grid regression for model order reduction in Sec. 8.

7. Supervised Learning with Sparse Grids

We start this section on supervised learning with density estimation — a fundamental *un*supervised learning problem of statistics but also a building block for many supervised learning methods. We employ the sparse grid density estimation method to solve the classification problem and then discuss ensemble learning methods and AdaBoost in the context of sparse grids.

7.1. Density Estimation with Adaptive Sparse Grids

Density estimation has always been a very important problem in statistics and data mining [177]. But lately, there has also been a brisk demand for a reliable and fast density estimation method in science and engineering applications, not least due to the recent uncertainty quantification trend in CSE, see, e.g., [4, 9, 6]. Here, we present a density estimation method based on adaptive sparse grids. It is reliable in the sense that the parameters do not have to be fine-tuned to obtain good results and it is fast and applicable to large data sets because it follows a grid-based approach.

Having discussed the sparse grid density estimation method in detail, we introduce algorithms for standard tasks in the context of density estimation. In particular, we show how to marginalize and to conditionalize sparse grid density functions and how to sample from the distribution described by a sparse grid density function.

7.1.1. Grid-Based Density Estimation

Suppose we have a data set $\mathcal{S} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\} \subset \mathbb{R}^d$ of samples drawn from an unknown distribution $p(X)$ of a random variable X. Note that we follow [28] and denote the distribution of the random variable X with $p(X)$. Density estimation is the construction of an estimation $\hat{p}$ of the probability density function p based on the data $\mathcal{S}$. Estimated density functions can either be used to present, visualize, and retrieve information about the data at hand [169], or they can be a means for other common tasks of data mining and statistics such as clustering and classification, see Sec. 7.2 and Sec. 10.2. We refer to [168, 169] for a thorough and general study on density estimation and its applications.

Parametric and nonparametric density estimation In general, we can distinguish between parametric and nonparametric density estimation methods. A parametric density estimation method assumes that the form of the underlying distribution is known and that only a small number of parameters has to be estimated. Very popular is a "mixture of Gaussians" where a linear combination of Gaussian densities is taken and their means and (co)variances are estimated with the expectation-maximization (EM) algorithm, see, e.g., [28]. If information about the data is known, for example, its structure can be described by a superposition of basic density functions, then parametric methods can approximate the underlying density function with a high accuracy. However, such information is rarely available.

We consider nonparametric density estimation. It uses only the given data samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ to estimate the density and it does not require any additional information about the data. There is a variety of nonparametric density estimation methods, cf. the survey [114]. Kernel density estimation has become the most widely used method of this kind. The (one-dimensional) estimator

$$\hat{p}(x) = \frac{1}{M} \sum_{i=1}^{M} K\left(\frac{x - x_i}{\sigma^2}\right),$$

is a linear combination of kernel functions K centered at the data points $x_i \in \mathcal{S}$. The performance of the estimator depends on the choice of the kernel function K and the bandwidth $\sigma > 0$. Whereas usually simply the Gaussian kernel $K(x) = (2\pi)^{-1/2} \mathrm{e}^{-x^2/2}$ is used, the selection of the bandwidth σ is a far more delicate matter. Approaches to determine a good σ reach from rules of thumb to highly sophisticated methods requiring a good amount of computational effort, see, e.g., [116, 58] and the references therein. Furthermore, the evaluation of $\hat{p}$ depends on the number M of (training) data points $\mathcal{S}$. Thus, in order to evaluate the estimated density function $\hat{p}$, all M kernel functions centered at all data points have to be evaluated. One remedy is to divide the data into a small number of bins and place a kernel function at each bin. Then the evaluation of $\hat{p}$ depends only on the number of bins and not on the number of data points anymore. However, the number of bins increases exponentially with the dimension of the data points (curse of dimensionality) and thus this is only feasible in up to, say, four dimensions.

Grid-based density estimation A promising density estimation method to overcome these two drawbacks — high sensitivity with respect to parameters and long runtimes for large data sets — has been introduced in [105]. The idea is to start with a highly-overfitted guess p_ϵ and then use spline smoothing to obtain a smoother and more generalized approximation $\hat{p}$.

Suppose we have an initial guess p_ϵ of the density function underlying the data $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$. Following spline smoothing [182], it is then proposed in [105], to look for $\tilde{p}$ in a function space $\mathcal{V}$ such that

$$\tilde{p} = \arg\min_{f \in \mathcal{V}} \int_\Omega (f(\boldsymbol{x}) - p_\epsilon(\boldsymbol{x}))^2 \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda f\|_{L^2}^2. \tag{56}$$

Whereas the left term of (56) ensures that the function $\tilde{p}$ closely fits the initial guess p_ϵ, the right term $\|\Lambda f\|_{L^2}^2$ is a regularization or penalty term imposing a smoothness constraint. For example, Λ might be chosen to be ∇. The regularization parameter $\lambda > 0$ controls the trade-off between fidelity and smoothness. Let $\delta_{\boldsymbol{x}_i}$ be the Dirac delta function centered on $\boldsymbol{x}_i$. If we set $p_\epsilon = \frac{1}{M} \sum_{i=1}^{M} \delta_{\boldsymbol{x}_i}$, we obtain after some transformations (see [105]) the variational equation

$$\int_\Omega \tilde{p}(\boldsymbol{x}) \psi(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda \tilde{p}(\boldsymbol{x}) \cdot \Lambda \psi(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \frac{1}{M} \sum_{i=1}^{M} \psi(\boldsymbol{x}_i), \tag{57}$$

for all test functions $\psi \in \mathcal{V}$. Of course, other choices for the initial guess p_ϵ are possible. However, here and in the following, we stick to this simple but sufficient choice. With Galerkin projection, we can find a solution $\hat{p}$ of the variational problem (57) in a finite-dimensional space. In [105], the density function $\tilde{p}$ is discretized with basis functions centered on grid points rather than with kernels centered at data points. However, just as binning for kernel density estimation, this full grid approach suffers from the curse of dimensionality: The number of grid points grows exponentially with the dimension of the data points. That is why we employ sparse grids.

Sparse-grid-based density estimation Let Φ_ℓ be the set of hierarchical basis functions of the (adaptive) sparse grid space $\mathcal{V}_\ell^{(1)}$ of level ℓ, cf. Sec. 2.4. We are then looking for $\hat{p} \in \mathcal{V}_\ell^{(1)}$ such that

$$\int_\Omega \hat{p}(\boldsymbol{x})\phi(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda\hat{p}(\boldsymbol{x}) \cdot \Lambda\phi(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \frac{1}{M}\sum_{i=1}^{M} \phi(\boldsymbol{x}_i) \tag{58}$$

holds for all $\phi \in \Phi_\ell$. Note that we are in the case of Ritz-Galerkin were the ansatz space is equal to the test space. Because $\hat{p} = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}$ is a linear combination, we can write (58) as a system of linear equations

$$(\boldsymbol{R} + \lambda\boldsymbol{C})\,\boldsymbol{\alpha} = \boldsymbol{b}, \tag{59}$$

with $R_{ij} = (\phi_i, \phi_j)_{L_2}$, $C_{ij} = (\Lambda\phi_i, \Lambda\phi_j)_{L_2}$ and $b_i = \frac{1}{M}\sum_{j=1}^{M} \phi_i(\boldsymbol{x}_j)$, where we used an arbitrary ordering of the N sparse grid basis functions $\phi_{l,i}$ and the corresponding coefficients $\alpha_{l,i}$. Note that a pure theoretical outline of a convergence analysis of a related approach without adaptive refinements and without a regularization term can be found in [163].

Just as for the classification problem in Sec. 2.4.2, the matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ are of size $N \times N$ and the right-hand side $\boldsymbol{b}$ is a vector of size N. Thus, the number of unknowns does not depend on the number of data points M anymore but only on the number of grid points N. With sparse grids we ensure that the number of grid points grows only moderately with the number of dimensions. Employing adaptivity, we can even further reduce the number of grid points.

Let us have a look at the system (59) from a computational point of view. We solve the system of linear equations with the conjugate gradient method. Hence, we do not have to form the matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ but only need to provide the matrix-vector product with, e.g., the UpDown algorithm, see Sec. 2.4. However, with the same arguments as for the classification problem, we can replace the matrix $\boldsymbol{C}$ with the identity operator $\boldsymbol{I}$ and thus the matrix-vector product with $\boldsymbol{C} = \boldsymbol{I}$ is for free, see Sec. 2.4.2.

7.1.2. Working with Estimated Sparse Grid Density Functions

In the previous section we have seen how to estimate the density function of data $\mathcal{S}$ with sparse grids. However, for many applications it is not enough to just have an estimated density function. In many practical cases, they need to, e.g., conditionalize the density function or compute the variance. In the following, we present algorithms to perform such standard operations with sparse grid density functions and show that their complexity is only linear in the number of grid points.

Expected value and variance Let us first consider the expected value and the variance. Suppose a random variable X has the sparse grid density function $\hat{p} = \sum_{(l,i)\in\mathcal{I}} \alpha_{l,i}\phi_{l,i} \in \mathcal{V}_\ell^{(1)}$. We have

$$\begin{aligned}\int_0^1 x \cdot \phi_{l,i}(x)\,\mathrm{d}x &= \int_{(i-1)2^{-l}}^{i2^{-l}} x \cdot \phi_{l,i}(x)\,\mathrm{d}x + \int_{i2^{-l}}^{(i+1)2^{-l}} x \cdot \phi_{l,i}(x)\,\mathrm{d}x \\ &= \frac{1}{6}\cdot 2^{-2l}\cdot(3i-1) + \frac{1}{6}\cdot 2^{-2l}\cdot(3i+1) = i\cdot 2^{-2l},\end{aligned}$$

and similarly we obtain

$$\int_0^1 x^2 \cdot \phi_{l,i}(x)\,\mathrm{d}x = \left(i^2 + \frac{1}{6}\right)\cdot 2^{-3l}.$$

These lead to the expected value

$$\hat{\mathbb{E}}[X] = \int_0^1 \boldsymbol{x}\cdot\hat{p}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j=1}^{d} i_j \cdot 2^{-2l_j}, \tag{60}$$

and, with $\hat{\mathrm{Var}}[X] = \hat{\mathbb{E}}[X^2] - \hat{\mathbb{E}}[X]^2$, to the variance

$$\hat{\mathrm{Var}}[X] = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} \alpha_{\boldsymbol{l},\boldsymbol{i}} \prod_{j=1}^{d} \left(i_j^2 + \frac{1}{6}\right)\cdot 2^{-3l_j} - \hat{\mathbb{E}}[X]^2. \tag{61}$$

It is obvious that the sums in (60) and in (61) are independent of the number of data points M and can be evaluated in $\mathcal{O}(N)$. Thus, they are linear in the number of sparse grid points.

Marginal and conditional density functions Assume we have estimated the joint probability function $\hat{p}$ of two random variables X_1 and X_2, see Fig. 19. The marginal probability density function associated with X_1 is

$$\hat{p}_1(x_1) = \int_{x_2} \hat{p}(x_1, x_2)\,\mathrm{d}x_2.$$

In case of a sparse grid density function

$$\hat{p}(x_1, x_2) = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{l_1,i_1}(x_1)\phi_{l_2,i_2}(x_2),$$

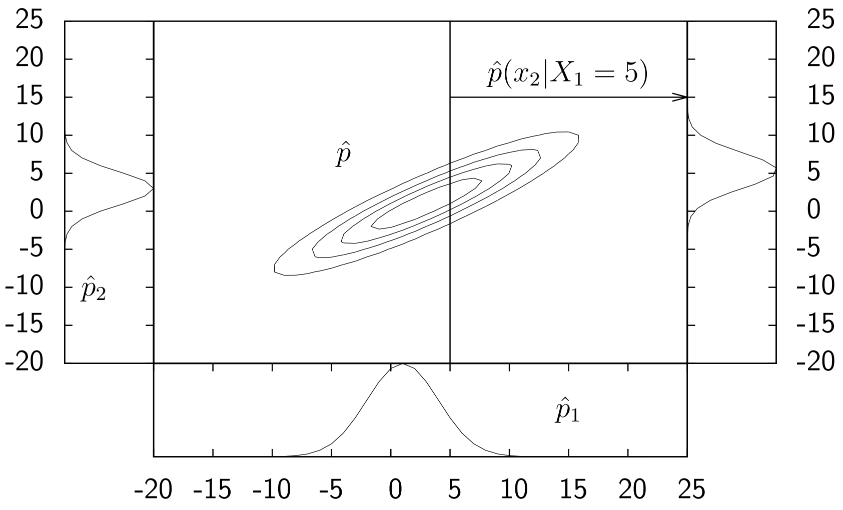


Figure 19: Contour plot of the joint probability density function $\hat{p}$ and visualization of the marginal $\hat{p}_1, \hat{p}_2$ and conditional $\hat{p}(x_2|X_1 = 5)$ density functions.

we have the marginal density function associated with X_1

$$\hat{p}_1(x_1) = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} 2^{-l_2}\alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{l_1,i_1}(x_1), \tag{62}$$

because $\int_0^1 \phi_{l,i}(x)\,\mathrm{d}x = 2^{-l}$. Since a d-dimensional sparse grid consists of several $d-1$-dimensional sparse grids [39], the marginal density function $\hat{p}_1$ can be represented without an additional approximation error on a one-dimensional sparse grid. This leads to significant savings in the number of grid points, cf. the numerical results in Sec. 7.1.4. This is also crucial if we marginalize in multiple dimensions. There the marginal operation is repeated several times and in each iteration it becomes cheaper, cf. the sampling method in the next section. Due to the tensor product approach (11) this procedure carries over to the d-dimensional case. Note that marginalizing a sparse grid density function does not depend on the number of data points and is only linear in the number of grid points.

The conditional probability density function of $\hat{p}$ given $X_1 = x_1$ is

$$\hat{p}\left(x_2|X_1 = x_1\right) = \frac{\hat{p}(x_1, x_2)}{\hat{p}_1(x_1)}.$$

Thus, if we have the estimated sparse grid density function $\hat{p} = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} \alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{\boldsymbol{l},\boldsymbol{i}}$, we first have to construct the marginal density function $\hat{p}_1$ to obtain the conditional density function

$$\hat{p}\left(x_2|X_1 = x_1\right) = \sum_{(\boldsymbol{l},\boldsymbol{i})\in\mathcal{I}} \frac{\alpha_{\boldsymbol{l},\boldsymbol{i}}\phi_{l_1,i_1}(x_1)}{\hat{p}_1(x_1)}\phi_{l_2,i_2}(x_2). \tag{63}$$

Just as in the case of the marginal density functions, we can represent (63) on a one-dimensional sparse grid without introducing an additional approximation error. The new coefficients can be easily computed with the marginalize procedure described above. Again, with the tensor product construction of the hierarchical basis functions, we can extend the procedure to the d-dimensional case. Furthermore, the procedure is linear in the number of grid points and does not depend on the number of data points.

Cross Validation The minimization problem (56) depends on the regularization parameter λ which controls the trade-off between fidelity and smoothness. It ensures that the estimated density function generalizes to new data, i.e., it prevents overfitting. The numerical experiments in Sec. 7.1.4 confirm that our proposed method is not very sensitive with respect to the regularization parameter and that $\lambda = 10^{-5}$ is a very good choice in most cases. Of course, the value $\lambda = 10^{-5}$ is only a rule of thumb and does not work for every data set. Therefore, sometimes, it might be worthwhile to tune the parameter to a particular data set.

From the training data $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ we estimate the density function, i.e., we obtain the hierarchical coefficients $\boldsymbol{\alpha}$, and test it by computing the residual of the system of linear equations (59) but with a new right-hand side $\boldsymbol{b}_\mathcal{T}$ which has been built by using the test data in $\mathcal{T}$. The L_2-norm of the residual

$$\|\boldsymbol{R}\alpha - \boldsymbol{b}_\mathcal{T}\|_2 \tag{64}$$

is then the accuracy indicator. Note that we ignore the regularization term $\boldsymbol{C}$ here. The residual (64) is a reasonable indicator because if the estimated density function captures the underlying density function and distribution we should obtain a small value for the test data as well. If our estimated density does not generalize to the test data then the value of the residual of the system of linear equation remains high. We compare the norms of the residuals for different choices of the parameter λ and select the one for which the residual is smallest, i.e., we select the λ for which the corresponding estimated density function generalizes best to the test data.

To reduce variability, the training data $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ is split into k equally sized sets to ensure that each sample is used for training and test. Each of these k sets is then used for testing and the union of the remaining ones for training. This is standard k-fold cross validation widely used in classification and other data mining tasks, cf. Sec. 2.2.1.

7.1.3. Sampling from Sparse Grid Density Functions

In the previous section, we have discussed several standard methods in the context of density estimation which operate directly on the estimated density function $\hat{p}$ and do not require samples drawn from the distribution $\hat{p}(X)$. But there are many situations where samples from $\hat{p}(X)$ are required such as in the SIR algorithm [9]. That is why we present a direct sampling method for sparse grid density functions, discuss its properties, and give details on the implementation.

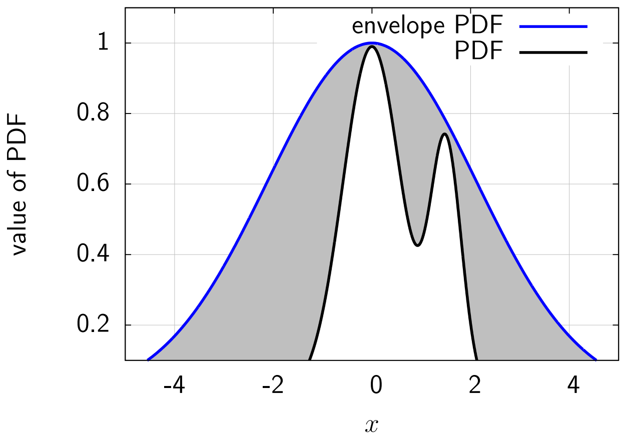
This sampling method is the result of many fruitful discussions with Sergio M. Amaral.

Sampling Our task is to draw samples from the distribution $\hat{p}(X)$ given only the sparse grid density function $\hat{p}$. For instance, we might want to find the expectation $\mathbb{E}[f] = \int f(\boldsymbol{x})\,\hat{p}(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ of a function f with respect to the estimated density $\hat{p}$. With samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ drawn from the distribution $\hat{p}(X)$ we can easily approximate the expected value by $1/M \sum_i f(\boldsymbol{x}_i)$. Another example where samples are required is the SIR algorithm which re-weights samples from one distribution to obtain samples from another one. See [9] where the SIR algorithm is extensively used in connection with uncertainty quantification.

We have several options to draw samples from the distribution $\hat{p}(X)$. Following [162], we distinguish between direct and Markov chain Monte Carlo (MCMC) sampling methods. The direct methods, e.g., inverse transform, acceptance-rejection, and importance sampling, usually rely only on the Law of Large Numbers whereas the MCMC approaches, e.g., Metropolis-Hastings, slice, Gibbs sampling, are based on asymptotic convergence properties of Markov chains. MCMC methods have the severe drawback that the generated samples are not independent before the Markov chain converged to the target distribution. It is difficult to assess if an MCMC sampler has already converged or not. Furthermore, MCMC sampling is inherently sequential and thus hard to parallelize [185]. Therefore, we do not consider MCMC sampling methods here but stick to direct methods. However, of course, sparse grid density functions are applicable in the context of MCMC. It is also worth noting that MCMC methods work very well in many situations and have extended the scope of sampling methods [162].

Acceptance-rejection sampling A common direct sampling method is acceptance-rejection sampling [28, 162]. Starting with a simple (estimated) density function $\hat{g}$ from which we can easily draw samples, we introduce a constant w such that $w \cdot \hat{g}(\boldsymbol{x}) \geq \hat{p}(\boldsymbol{x})$ for all $\boldsymbol{x}$ in the domain of $\hat{g}$ and $\hat{p}$. We then draw a sample $\boldsymbol{x}_0$ from the distribution $\hat{g}(X)$ and a uniformly distributed sample z over $[0, w \cdot \hat{g}(\boldsymbol{x}_0)]$. The sample $\boldsymbol{x}_0$ is rejected if $z > \hat{p}(\boldsymbol{x}_0)$, otherwise it is accepted. As thoroughly discussed in [28], the rejection sampling method leads to a tremendous amount of rejected samples which in turn means that it becomes very slow if we draw a large number of samples. This can have two reasons. First, the so-called envelope density $\hat{g}$ has to approximate the density $\hat{p}$ rather well in order to minimize the number of rejected samples, see Fig. 20. However, usually, we do not know much about the structure of $\hat{p}$ and thus we cannot easily find an appropriate $\hat{g}$. Furthermore, the density $\hat{g}$ has to be simple enough to allow us to rapidly draw samples from the corresponding distribution. In [81, 80] the adaptive rejection sampling method is introduced which automatically constructs an appropriate envelope density. However, even though this might lead to a good envelope density $\hat{g}$ in certain situations, it is still difficult in practical situations where densities usually have multiple sharp peaks and consist of different modes. The second crucial disadvantage of rejection

Figure 20: To keep the number of rejected samples low, the envelope density used in the acceptance-rejection sampling method has to approximate the density function well. The gray area visualizes the amount of samples which are rejected. It increases exponentially with the number of dimensions [28].



sampling is that the acceptance ratio diminishes exponentially with the dimension. This is illustrated in [28] on a simple example.

These two drawbacks — hard to find envelope density and curse of dimensionality — suggest that rejection sampling works well if we need only a small number of samples but that it reaches its limits soon if large sets of samples are required. In the following, we develop a direct sampling method for sparse grid density functions which does not rely on an envelope density and is not so prone to the curse of dimensionality.

Ancestral sampling We can represent a density function p as product of conditionalized densities

$$
\begin{aligned}
p(\boldsymbol{x}) = p(x_1, \dots, x_d) = & p_1(x_1) \cdot \\
& p_2(x_2 | X_1 = x_1) \cdot \\
& p_3(x_3 | X_2 = x_2, X_1 = x_1) \cdot \\
& \dots \\
& p_d(x_d | X_{d-1} = x_{d-1}, \dots, X_1 = x_1)
\end{aligned}
\tag{65}
$$

where, for example, $p_2(x_2|X_1 = x_1)$ is the density p conditionalized at $X_1 = x_1$ and then marginalized to X_2. Equation (65) immediately follows from the definition of conditional probability [121]. With (65) we have decomposed the density function p into a product of one-dimensional densities. If we assume we can sample from a one-dimensional sparse grid density function, we can start to sample the marginal density p_1 and obtain x_1 which is the first component of the d-dimensional sample $\boldsymbol{x}$. With x_1 we conditionalize p and marginalize to X_2 to obtain a one-dimensional density $p_1(x_2|X_1 = x_1)$ which we in turn use to sample the second component of $\boldsymbol{x}$. We continue this process until we have drawn a complete sample $\boldsymbol{x} = [x_1, \dots, x_d]^T$. This decomposition is related to the so-called ancestral sampling known in the context of graph models, see [28]. The ingredients for the sampling method are procedures to marginalize and to conditionalize a sparse grid density function and a method to draw samples from a distribution with a

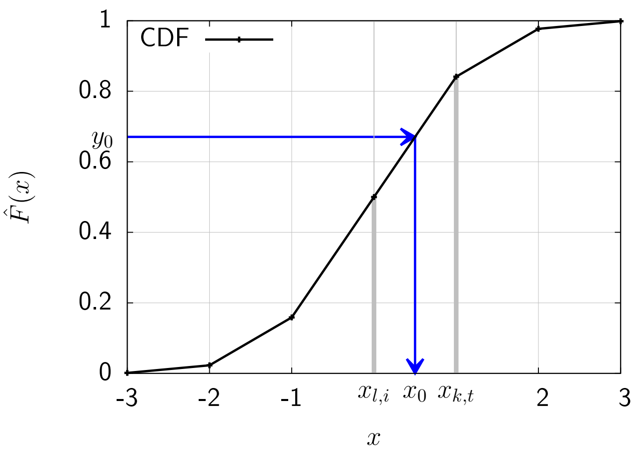


Figure 21: To evaluate $\hat{F}^{-1}$ at a point y_0, the two grid points $x_{l,i}$ and $x_{k,t}$ with $\hat{F}(x_{l,i}) \leq y_0 < \hat{F}(x_{k,t})$ are determined to compute x_0 with linear interpolation.

one-dimensional sparse grid density function. In the previous section, we have seen how to marginalize and to conditionalize. Thus, we only have to discuss the one-dimensional sampling.

Let $\hat{p}$ be a one-dimensional sparse grid density function. We employ inverse transform sampling [28] for $\hat{p}$ and thus have to construct the inverse function $\hat{F}^{-1}$ of the cumulative distribution function (CDF)

$$\hat{F}(x) = \int_{-\infty}^{x} \hat{p}(x)\,\mathrm{d}x.$$

In case of sparse grids and the linear hierarchical hat basis, the one-dimensional density function $\hat{p}$ is piecewise linear and with it the corresponding distribution function $\hat{F}$. Furthermore, we know that the range and domain of $\hat{F}$ is $[0,1]$ and that it is strictly monotone. To find $x_0 \in [0,1]$ such that $y_0 = \hat{F}(x_0)$, we iterate through the (sparse) grid points on which $\hat{F}$ is discretized and determine the two unique grid points $x_{l,i}$ and $x_{k,t}$ with $\hat{F}(x_{l,i}) \leq y < \hat{F}(x_{k,t})$. With linear interpolation between $\hat{F}(x_{l,i})$ and $\hat{F}(x_{k,t})$ we find $x_0 \in [x_{l,i}, x_{k,t})$ with $\hat{F}(x_0) = y_0$, see Fig. 21.

Having now all building blocks together — marginalize, conditionalize, one-dimensional sampling — we can summarize the multi-dimensional sampling method for sparse grid density functions in Alg. 4. The method SAMPLEdD is a recursive function which calls itself d times to generate a sample $\boldsymbol{x} \in \mathbb{R}^d$. Its parameters are the sparse grid function $\hat{p}$ represented by the basis Φ and the coefficient vector $\boldsymbol{\alpha}$, the sample point $\boldsymbol{x}$ which will be filled step by step, and the counter variable i with default value 1. The recursion terminates when all components are generated, i.e., if $i > \mathrm{len}(\boldsymbol{x})$. Except for the first component ($i = 1$) we have to conditionalize with respect to the dimensions $1, \ldots, i-1$. In each call it is enough to conditionalize just with respect to dimension 1 because from the previous recursive call we already get the $d - i + 1$-dimensional density function which is the result of conditionalizing $\hat{p}$ given $X_1 = x_1, \ldots, X_{i-1} = x_{i-1}$. The next step is to marginalize in all dimensions except 1 to obtain a one-dimensional density and then to draw the i-th component of $\boldsymbol{x}$ with the one-dimensional sampling method SAMPLE1D described above. After that, we perform the recursive call to draw the $i+1$-th component.

Complexity One call to SAMPLEdD returns a d-dimensional vector $\boldsymbol{x}$ which means we have drawn one sample from the distribution $\hat{p}(X)$. To draw M' samples $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{M'}\}$, we require M' calls of SAMPLEdD. First, we can execute them in parallel. Second, a call of SAMPLEdD is not expensive. Even though in each recursive step a conditionalize and marginalize operation is necessary, they become cheaper and cheaper as the recursion proceeds because if we are in the i-th step we only have to deal with a $d-i+1$-dimensional function. We show that this is indeed the case with the numerical results in the following section.

Mixing dimensions We are not only interested in the runtime of the sampling method but also in the quality of the generated samples. Note that, in general, we do not have quality issues such as MCMC samplers (independence of samples) because we have a direct sampling method. However, we also have to draw many samples to match the distribution of the random variable due to the Law of Large Numbers.

The d-dimensional sampling method SAMPLEdD builds on the one-dimensional procedure SAMPLE1D. As far as SAMPLE1D is concerned, we only have to deal with one-dimensional piecewise linear functions $\hat{F}$ which we can invert without introducing any additional error. This means that the quality of the one-dimensional samples depends only on the one-dimensional density function $\hat{p}$ and its distribution function. In the case of SAMPLEdD, the one-dimensional density functions are obtained by marginalizing (MARGTODIM1) and conditionalizing (COND) the joint probability function $\hat{p}$. Because we evaluate the corresponding formulas (62) and (63) exactly, we do not introduce any additional error there too, cf. Sec. 7.1.2. This shows that we cannot improve the building blocks SAMPLE1D, COND, and MARGTODIM1 of our sampling method anymore and thus we have to concentrate on SAMPLEdD itself.

One drawback of SAMPLEdD is that if we introduce a large error in the first component x_1 of $\boldsymbol{x} = [x_1, \ldots, x_d]^T$, e.g., because our estimated density function does not fit the data well, this error is propagated through the rest of the recursive process. It can be observed that the last component x_d is always the worst. Fortunately, we usually generate not only one but many samples. To prevent that always the last component is at a disadvantage, we randomly change the order in which the components $x_1, \ldots, x_d$ are generated. This distributes the error induced by the estimated density function across all components and generates better samples in average.

7.1.4. Numerical Results

We now evaluate the sparse grid density estimation method and the algorithms introduced in Sec. 7.1.2 and Sec. 7.1.3 on artificial and real-world data sets. In case of the artificial data sets, we know the exact density function p and thus can compute the average L_2 error

$$\frac{1}{M_{\mathcal{T}}} \sqrt{\sum_{\boldsymbol{x} \in \mathcal{T}} \left(p\left(\boldsymbol{x}\right) - \hat{p}\left(\boldsymbol{x}\right)\right)^2}, \tag{66}$$

Algorithm 4 Sampling from sparse grid density functions

```
1: procedure SAMPLEdD($\Phi$, $\alpha$, $\boldsymbol{x}$, $i = 1$)
2:     if $i > \mathrm{len}(\boldsymbol{x})$ then
3:         return $\boldsymbol{x}$
4:     end if
5:     if $i > 1$ then
6:         $\boldsymbol{\alpha}, \Phi \leftarrow$ COND($\Phi$, $\boldsymbol{\alpha}$, 1, $\boldsymbol{x}[i-1]$)      ▷ conditionalize in dimension 1
7:     end if
8:     $\tilde{\boldsymbol{\alpha}}, \tilde{\Phi} \leftarrow$ MARGTODIM1($\Phi$, $\boldsymbol{\alpha}$)      ▷ marginalize all dimensions except 1
9:     $\boldsymbol{x}[i] \leftarrow$ SAMPLE1D($\tilde{\Phi}$, $\tilde{\boldsymbol{\alpha}}$)      ▷ draw one-dimensional sample
10:    $\boldsymbol{x} \leftarrow$ SAMPLEdD($\Phi$, $\boldsymbol{\alpha}$, $\boldsymbol{x}$, $i + 1$)
11:    return $\boldsymbol{x}$
12: end procedure
```

where $\mathcal{T}$ is a test set with $|\mathcal{T}| = M_{\mathcal{T}}$. Our artificial data sets are sampled in each direction independently from either a normal distribution with mean 0.4 and standard deviation 0.1 or a beta distribution with $\alpha = 3$ and $\beta = 5$. A data set denoted by "GGBBB" is a five-dimensional data set ($d = 5$) where the samples in the first two dimensions are drawn from the normal distribution and in the third, fourth and fifth from the beta distribution. Besides these very simple data sets, we also consider common multi-modal benchmarks and real-world data sets where the exact density function is not always available.

We compare our density estimation method based on sparse grids (SGDE) with the kernel density estimation (KDE) implemented in the widely-used R package KS [58] with the Gaussian kernel. Note that we also compared to other KDE implementations (R KernSmooth, MATLAB KDE Toolbox, and Python SciPy) where we obtained similar results as with the KS package. Furthermore, the KS package not only implements the pure KDE method but also includes several improvements. It supports binning in up to four dimensions (curse of dimensionality) to cope with large data sets and it provides several kernel bandwidth selection methods of which we employed Hns (normal scale bandwidth selection method). Even though more sophisticated bandwidth selection methods exist, Hns turned out to be a good trade-off between run time and accuracy.

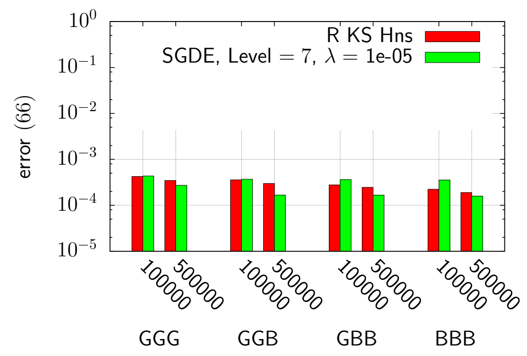
Accuracy We show the results for the artificial data sets with dimensions three, four, five, and six in Fig. 22 and with dimensions eight and nine in Fig. 23. We have $M = 100,000$ and $M = 500,000$ data points. In the three-dimensional and the four-dimensional case, the test set $\mathcal{T}$ has to be a full grid with 2^5 points in each direction due to constraints of the binned kernel density estimator of the R KS package. In dimensions > 4 we set $\mathcal{T}$ to the first 100,000 points of the low-discrepancy Sobol sequence generated with the GNU Scientific Library. The regularization parameter λ of the SGDE method to control the smoothness of $\hat{p}$ is always set to 10^{-5}. This emphasizes that the choice of λ is indeed a good rule of thumb. In case of adaptive sparse grids, we start with a regular sparse grid of level three and perform five refinement steps. In each step we refine the 100 grid points with the highest absolute hierarchical coefficient weighted with the function value at the corresponding grid point [153].

Fig. 22 clearly shows that our method is competitive with KDE even though we keep the regularization parameter λ fixed over all data sets and all dimensions. In eight out of 13 comparisons with KDE, our SGDE method achieves better results. Note that because the KS package supports binning only in up to four dimensions, we were not able to evaluate the kernel density estimator in five and six dimensions with $M = 500,000$ sample points, cf. the runtimes in Tab. 2. The plot also shows that the results of our method are still very good with adaptive sparse grids where we need even fewer ($\approx$ 5,000) grid points. Thus, whereas the KDE method has to iterate over all 100,000 or 500,000 samples to evaluate the estimator, our method has to iterate only over $\approx 5,000$ grid points which is orders of magnitude less than for KDE.

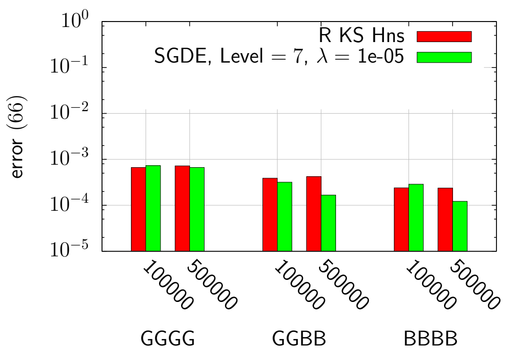
In the higher-dimensional case in Fig. 23, we show the results only of our method because KDE becomes computationally very expensive for such higher-dimensional and large data sets. It is important to emphasize that only because we exchanged the full grid discretization proposed in [105] with our sparse grid discretization, it is computationally feasible to handle such higher-dimensional data sets with up to nine dimensions with the discussed density estimation method. We do not only plot the results with regularization parameter λ set to 10^{-5} but also for the best choice in $\{10^{-8}, 10^{-7}, \dots, 10^{-3}\}$ with respect to the error (66). Clearly, the results do not distinctly differ which emphasizes that a rough choice of λ is sufficient if we have large data sets.

Runtime In Tab. 2 we summarize the runtimes in the five- to nine-dimensional settings. The number of points where we evaluate the estimated density function is always set to $M_{\mathcal{T}} = 100,000$ but the training sample size M varies from 10,000 to 500,000. All measurements where performed on a system with an Intel Core i7-870 and 8GB RAM. The parallel version (OpenMP) of SGDE with four threads uses parallelized algorithms for the matrix-vector product with $\boldsymbol{R}$ of the system of linear equations (59) and a parallelized sparse grid function evaluation procedure based on the algorithms introduced in [109, 108]. We split the total runtime into the time spent for solving the system of linear equations (59) and the time needed to evaluate the estimator $\hat{p}$. It is important to note that a direct comparison of the runtimes of our SGDE implementation in C++ and the KDE implemented in the R KS package is not reasonable because we cannot compare compiled and interpreted code. However, since we only want to show how the runtime of each method scales with the number of sample points M, we nevertheless include the runtimes of R KS in Tab. 2.

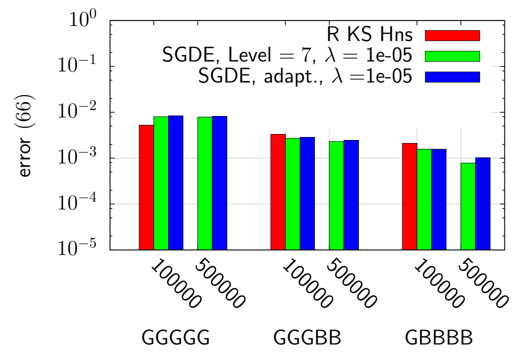
As expected, the runtime of KDE increases proportional to the number of sample points M. This is not the case for SGDE. The evaluation time stays constant, no matter if we have 10,000 or 500,000 training points. The time to solve the system of linear equations (59) increases slightly because of the computation of the right-hand side which depends on the number of data points. Whereas the sample size is increased by a factor of 50, the runtime to solve the system increases only by a factor of three. The



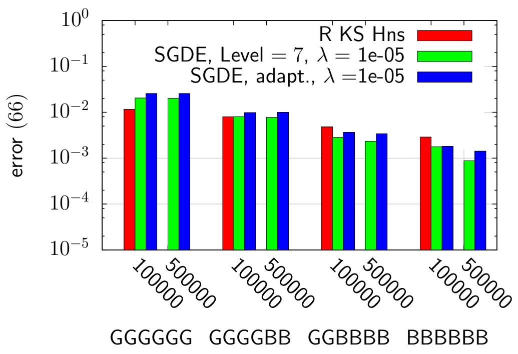
(a) three dimensions



(b) four dimensions

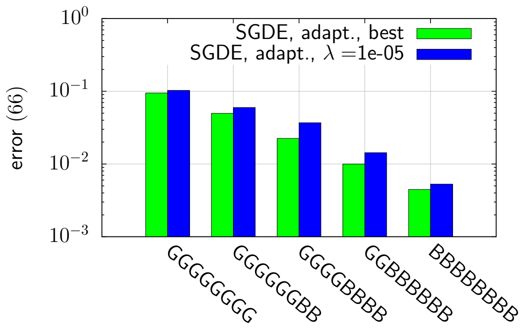


(c) five dimensions

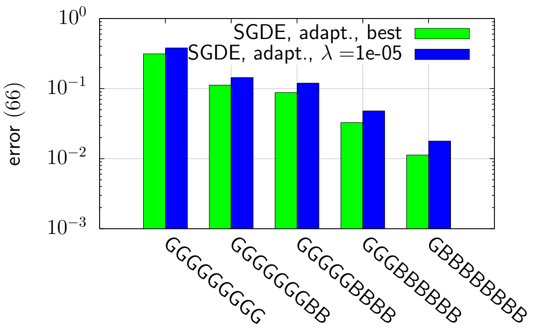


(d) six dimensions

Figure 22: Comparison of our density estimation method based on sparse grids (SGDE) and kernel density estimation (R KS Hns): The numbers (100,000 and 500,000) on the x axes are the sample sizes (M). The results for the kernel density estimation with sample size $M = 500,000$ and dimensions five and six are missing due to very long computation times.



(a) eight dimensions



(b) nine dimensions

Figure 23: Comparison of our SGDE method with optimal parameter λ ("best") and with fixed parameter λ = 1e-05 in up to nine dimensions: The number of sample points M is set to $500,000$. The optimal choice of λ does not distinctly improve the results which emphasizes that a rough choice of λ is sufficient.

time to solve the system of linear equations is dominated by the algorithm to compute the matrix-vector product with $\boldsymbol{R}$. Even though this algorithm scales only linearly in the number of grid points, it scales in $\mathcal{O}(2^d)$ with the dimension d. This explains the increase of the solution time with the dimension by roughly a factor of two. The increase of the evaluation time with the number of dimensions is due to the product of the basis functions (11) and time consuming index lookups. We refer to [109, 108] for an in-depth analysis of these sparse grid algorithms. Nevertheless, even in nine dimensions with 500,000 sample points our SGDE method only needs 1,219 seconds on one core to estimate the density function and evaluate it at 100,000 test points. The parallel version on four cores takes only 401 seconds.

Cross Validation Even though we have seen that in all examples a very rough choice of the regularization parameter λ was sufficient, see Fig. 23, we want to demonstrate the parameter selection procedure described in Sec. 7.1.2 on a six-dimensional example. We performed three-fold cross validation and searched the parameter in the range $[10^{-8}, 10^{-1}]$ with 10 steps on a logarithmic scale. On the one hand, the results shown in Fig. 24 underline that we can slightly improve the error by such a cross validation method and that the accuracy indicator of Sec. 7.1.2 works. On the other hand, the results emphasize once more that a standard choice (here $\lambda = 10^{-5}$) for the regularization parameters works very well indeed.

Real-world data sets So far we have evaluated our SGDE method only on artificial data sets (normal and beta distributions). Here we now consider five multi-modal real-world data sets from various application areas which are frequently used in the data mining community. However, because we have real-world data sets we do not know the

			SGDE, single thread			SGDE, four threads			R KS
d	M	gp	solve[s]	eval[s]	total[s]	solve[s]	eval[s]	total[s]	total[s]
5	10,000	4230	37	3	40	16	1	17	467
5	100,000	4230	50	3	53	18	1	19	4857
5	500,000	4285	115	4	119	37	1	38	-
6	10,000	5503	78	5	83	32	1	33	529
6	100,000	5311	95	5	100	36	1	37	6551
6	500,000	5198	208	5	213	67	1	68	-
7	500,000	6823	394	7	401	128	2	130	-
8	500,000	9391	749	11	760	245	4	249	-
9	500,000	12301	1205	14	1219	397	4	401	-

Table 2: Runtime (in seconds) of the SGDE method for the data sets GGGBB (5 dim.), GGBBBB (6 dimensions), GGGBBB (7 dimensions), GGGGBBBB (8 dimensions), and GGGGGBBBB (9 dimensions) are reported. In contrast to the kernel density estimation, the runtime of our method increases only slightly with the number of samples.

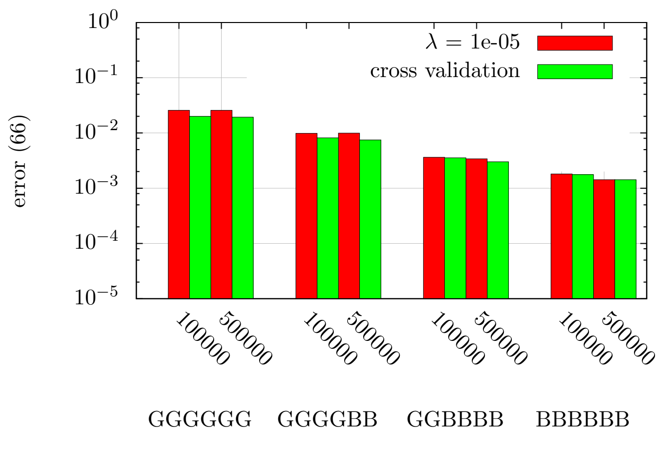


Figure 24: Six-dimensional example to show the effect of the selection of the regularization parameter λ with the cross validation method discussed in Sec. 7.1.2: The error improves if we employ cross validation but overall the standard choice $\lambda = 10^{-5}$ seems to be sufficient.

data set	dim	M	$M_{\mathcal{T}}$	modes	SGDE	R KS (Hns)
old faithful	2	184	46	2	**137.64**	50.84
iris	4	135	15	3	**59.68**	35.39
svmguide1	4	3089	4000	2	**16201.60**	16091.76
olives	8	348	88	3	175.06	**712.59**
codRNA	8	59535	1000	2	7474.91	**8906.72**

Table 3: Log-Likelihood for real-world data sets: The number of modes corresponds to the number of clusters in the data sets. Overall, our SGDE method performs better than R KS.

exact density function anymore. Following [172], we split the data sets into a training set $\mathcal{S}$ and test set $\mathcal{T}$, estimate the density function on the training data, and compute the log-likelihood of the test data

$$\sum_{\boldsymbol{x}\in\mathcal{T}} \log\left(\hat{p}(\boldsymbol{x})\right).$$

The log-likelihood of the test samples is then a measure how well our estimated density function generalizes to before unseen data. In Tab. 3, we compare our SGDE method with the kernel density estimation of the R KS package. We started with a sparse grid of level four and performed seven refinement steps where we added 100 grid points in each step. The regularization parameter λ has been determined by cross validation. In three out of five data sets, SGDE is better than KDE. Both, the codRNA and the olive oil data set, contain one mode (or cluster) with a very sharp peak. Even though adaptivity helps a lot, such sharp peaks are hard to catch with a grid-based discretization. That might be a reason why SGDE does not perform so well for those two data sets. In case of the olives data set, we find the same effect in the density-based clustering example in Sec. 10.2.

Marginal density functions We draw $100,000$ samples from the distributions listed in Tab. 4 independently in each dimension and estimate the joint density function with the R KS package (with binning) and our SGDE method. Thus, we have a four-dimensional data set with $M = 100,000$ samples. With the procedure of Sec. 7.1.2 we marginalize the joint density function obtained with our SGDE method such that only one dimension remains, see Fig. 25. To cope with the very sharp peaks in the marginalized density functions, we employ a regular sparse grid of level eight (23,297 grid points) and allow rougher functions by slightly decreasing the smoothness parameter λ to 10^{-6}. Marginalization in case of KDE means that only the kernel in the remaining dimension is evaluated. In Tab. 5, we compare the error (66) of the marginalized density functions on a one-dimensional equidistant mesh with 2^8 points. The SGDE method achieves in dimension one and three a smaller error than the kernel density estimation. We would like to emphasize that after marginalization the SGDE method employs only 255 grid

d	Target PDFs (G is Gaussian PDF)	$\mathbb{E}$	Var
1	$\frac{2}{3}G(\frac{1}{2}, \frac{1}{6}) + \frac{1}{3}G(\frac{1}{2}, \frac{1}{60})$	0.50	1.86e-2
2	$\sum_{l=0}^{7} \frac{1}{8}G(\frac{6}{11}\left(\frac{2}{3}\right)^l + \frac{1}{11}, \frac{2}{11}\left(\frac{2}{3}\right)^l)$	0.28	3.56e-2
3	$\frac{1}{2}G(\frac{1}{18}, \frac{1}{90}) + \frac{1}{2}G(\frac{11}{18}, \frac{1}{9})$	0.33	8.34e-2
4	$\frac{1}{2}G(\frac{1}{2}, \frac{1}{35}) + \frac{1}{4}\left(G(\frac{1}{4}, \frac{1}{50}) + G(\frac{3}{4}, \frac{1}{50})\right)$	0.50	3.19e-2

Table 4: Marginalized density functions in dimensions one to four, cf. [132], and the error, expected value, and variance corresponding to the density functions.

	SGDE			R KS		
d	error (66)	$\hat{\mathbb{E}}$	$\hat{\text{Var}}$	error (66)	$\hat{\mathbb{E}}$	$\hat{\text{Var}}$
1	**3.37e-3**	0.50	1.80e-2	3.63e-3	0.50	1.79e-2
2	4.85e-3	0.29	3.40e-2	**2.58e-3**	0.29	3.43e-2
3	**7.18e-3**	0.33	8.32e-2	9.31e-3	0.33	8.37e-2
4	3.35e-3	0.50	3.15e-2	**2.42e-3**	0.50	3.19e-2

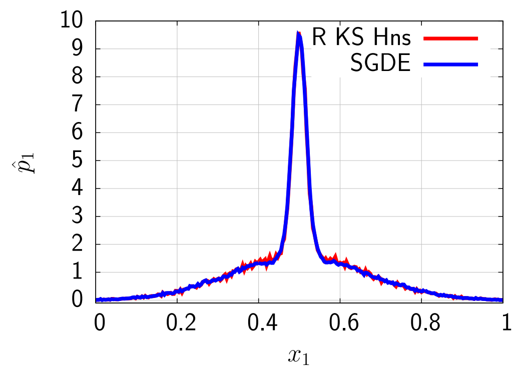
Table 5: Error, expected value, and variance for the marginalized density functions, cf. the exact PDFs in Tab. 4.

points (one-dimensional sparse grid of level eight). Hence, to evaluate the marginalized density function we only have to iterate over 255 grid points whereas in the case of the kernel density estimation we still have to iterate over the kernels centered at all 100,000 samples. We also compute the expected value and the variance as described in Sec. 7.1.2. For the kernel density estimation the derivation of the corresponding formulas is straightforward. Both our SGDE method and the kernel density estimation achieve very good results, see Tab. 5.

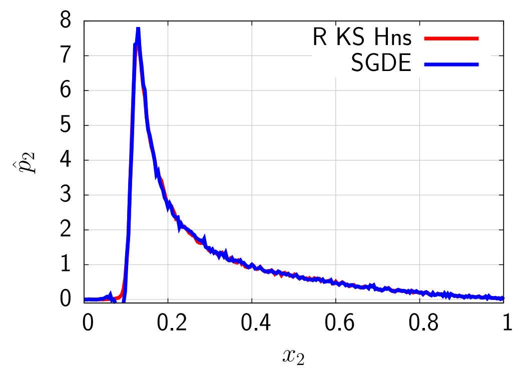
Sampling We now do not compare kernel density estimation and our sparse grid density estimation but acceptance-rejection sampling and our ancestral sampling method. We draw samples with the acceptance-rejection and ancestral sampling method described in Sec. 7.1.3. We use the probability density function of the uniform distribution in the unit cube as envelope density for the acceptance-rejection sampling and we always mix the start dimension for the ancestral sampling method, cf. the remark in Sec. 7.1.3. We always perform cross validation to obtain the regularization parameter λ.

In Fig. 26 we show 1,000 samples drawn from the distribution corresponding to the old faithful data set. Our sparse grid density estimation method clearly detects the two clusters and thus both sampling methods yield samples which fit well to the original data.

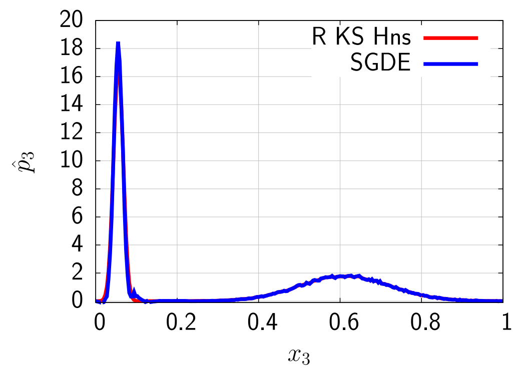
Let us now consider Tab. 6. For each data set, we drew 10,000 samples and computed the empirical mean $\hat{\overline{\boldsymbol{m}}} = [\hat{\overline{m}}_1, \ldots, \hat{\overline{m}}_d]^T \in \mathbb{R}^d$ of the samples and empirical mean $\overline{\boldsymbol{m}} =$



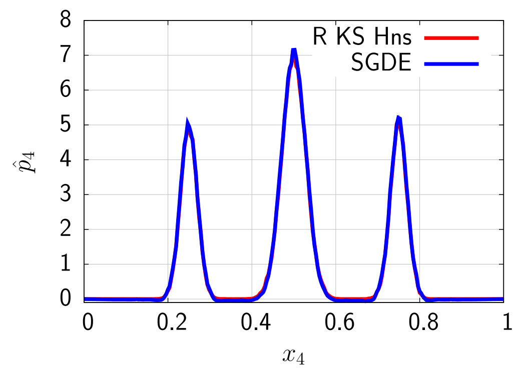
(a) dimension one



(b) dimension two

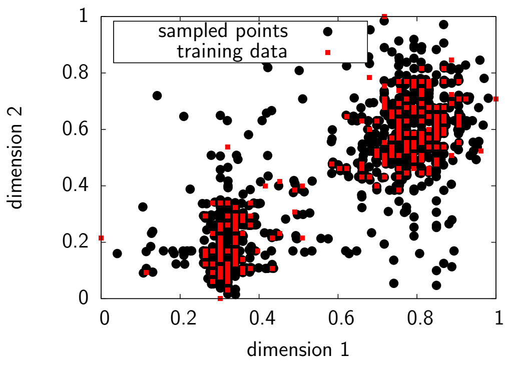


(c) dimension three

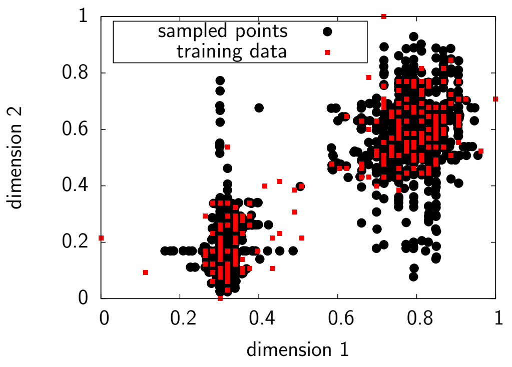


(d) dimension four

Figure 25: The marginalized density functions corresponding to the four-dimensional data set generated from the distribution shown in Tab. 4: The data exhibits in every dimension a very different behavior. Whereas the kernel density estimator tends to oscillate (dimension one and two), our sparse grid method yields a smoother results.



(a) ancestral sampling



(b) acceptance-rejection sampling

Figure 26: With a sparse grid density function (level five with seven refinement steps) for the old faithful data set, we drew 1,000 samples with ancestral (left) and acceptance-rejection (right) sampling. Both methods draw samples with good agreement to the original training data points. However, the samples generated by our ancestral method, better fit the data. This is confirmed by the results in Tab. 6.

$[\overline{m}_1, \ldots, \overline{m}_d]^T \in \mathbb{R}^d$ of the training data. The sum of the relative errors

$$\frac{100}{d} \sum_{i=1}^{d} \left(\frac{|\hat{\overline{m}}_i - \overline{m}_i|}{\overline{m}_i} \right) \tag{67}$$

combines the two vectors $\hat{\overline{\boldsymbol{m}}}$ and $\overline{\boldsymbol{m}}$. The value (67) can be interpreted as the difference between the mean of the samples and the training data in percent. We did the same for the empirical variance and report both values in Tab. 6. In addition, we show the runtime of the two methods on a single core of an Intel Xeon E5-2670. In almost all cases, our ancestral sampling method is not only better with respect to the error (67) corresponding to the empirical mean and the empirical variance but it also distinctly faster.

In Tab. 7, we show more details on the runtimes of the two methods on one and four cores. Again, our ancestral sampling method is faster than acceptance-rejection sampling in all cases. The runtime of the acceptance-rejection sampling method heavily depends on the generated pseudo-random numbers. Thus the runtimes are nondeterministic because not always the same amount of samples is rejected. The runtimes and speedups from one to four cores in Tab. 7 also show that our method can cope with today's multi- and many-core systems. Furthermore, the results confirm that our ancestral sampling method scales only linearly with the number of samples and with the number of grid points.

7.2. Classification with Density Functions

We present a novel classification method based on sparse grid density estimation. In contrast to the common classification method based on sparse grid regression which

				ancestral sampling			acc-rej sampling		
	ℓ	ref	gp	mean[%]	var[%]	time[s]	mean[%]	var[%]	time[s]
oldfaithful	5	0	129	**0.39**	**0.68**	1	0.44	1.15	**0**
	5	5	1,736	**1.32**	4.83	7	1.89	**4.63**	**5**
	5	7	2,681	**0.60**	2.99	**10**	2.25	**2.44**	12
iris	5	0	769	13.34	**24.87**	11	**12.65**	26.10	**5**
	5	5	5,746	**4.23**	**10.14**	**59**	4.93	37.68	240
	5	7	9,414	**3.64**	**11.02**	**93**	6.28	39.54	286
svmguide	5	0	769	**1.96**	**48.91**	10	10.62	84.18	**6**
	5	5	5,115	**0.44**	**13.28**	**42**	8.33	29.65	141
	5	7	8,896	**0.74**	**12.49**	**67**	9.87	34.95	374
sergio	5	0	1,471	**1.30**	51.87	**30**	1.56	**48.48**	60
	5	5	9,967	**0.41**	27.89	**131**	1.63	**26.96**	313
	5	7	19,908	**0.37**	23.55	**229**	2.22	**18.67**	497
olives	5	0	6,401	**14.00**	**23.60**	**357**	18.92	26.91	1,966
	5	5	28,593	9.01	19.23	**1,625**	**8.93**	**16.48**	9,152
codRNA	5	0	6,401	**8.35**	**60.65**	**372**	21.39	122.50	404
	5	5	36,137	**5.47**	**65.44**	**1,297**	7.57	67.00	6,997

Table 6: Reports the relative difference (percent) between the empirical mean and variance of the training data and 10,000 drawn samples. The timings (seconds) were performed on a single core only. In almost all cases the ancestral method is better than acceptance-rejection sampling with respect to accuracy and runtime.

	ℓ	ref	gp	#samples	one core anc[s]	one core rej[s]	four cores anc[s]	four cores rej[s]
iris	5	5	5,746	1,000	**6**	15	**1**	3
				10,000	**59**	240	**15**	28
				100,000	**584**	674	**149**	409
	5	7	9,390	1,000	**9**	17	**2**	14
				10,000	**93**	286	**23**	192
				100,000	**931**	3,087	**234**	927
svmguide	5	5	5,115	1,000	**4**	15	**1**	5
				10,000	**42**	141	**11**	39
				100,000	**413**	1,324	**105**	517
	5	7	8,896	1,000	**7**	56	**1**	13
				10,000	**67**	374	**17**	65
				100,000	**669**	5,296	**167**	1,462
sergio	5	5	9,967	1,000	**13**	46	**4**	12
				10,000	**131**	313	**33**	129
				100,000	**1,313**	4,363	**332**	709
	5	7	20,549	1,000	**23**	74	**7**	14
				10,000	**229**	497	**58**	160
				100,000	**2,288**	6,635	**590**	2,612
olives	5	5	28,593	1,000	**128**	291	**35**	70
				10,000	**1,625**	9,152	**324**	1,081
				100,000	**17,207**	39,400	**3,253**	30,575
codRNA	5	5	36,137	1,000	**124**	1,582	**37**	269
				10,000	**1,297**	6,997	**339**	1,955
				100,000	**16,499**	124,925	**3,552**	13,653

Table 7: Our ancestral sampling method is always faster than acceptance-rejection sampling. The results also show that our method scales only linearly with the number of samples and with the number of grid points.

has been discussed in Sec. 2.4.2, our method can deal with more than two classes in a natural way and it provides a stochastically motivated confidence value which indicates how to rate the respond to a new point. Furthermore, the underlying sparse grid density estimation method introduced in the previous section allows us to split the computational procedure into an expensive Offline step (pre-processing) in favor of a very fast Online phase where the actual classification of new points takes place.

7.2.1. Classification with Sparse Grid Density Estimation

We consider the multi-class classification problem as defined in Sec. 2.2.1: We want to reconstruct the unknown function $c : \mathbb{R}^d \to \{1, \dots, k\}$ which assigns class labels $1, \dots, k$ to points in the d-dimensional space $\mathbb{R}^d$. Given is only the training data $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^M \subset \mathbb{R}^d \times \{1, \dots, k\}$ which might contain noise.

Non-probabilistic and probabilistic approaches to classification with sparse grids What we call the classical sparse grid classification method employs regression where the regression function is discretized on a sparse grid. This is a non-probabilistic approach, i.e., the mapping between data points and class labels is directly determined [115]. In contrast, we now introduce a probabilistic, generative approach for classification with sparse grids, cf. Sec. 2.2.1. The method is based on sparse grid density estimation. We learn the class-conditional[1] densities $p(\boldsymbol{x}|Y_i)$ for each class and uses Bayes' theorem (3) to find the posterior class probabilities $p(Y_i|\boldsymbol{x})$. Recall that Y_i is the binary random variable which indicates if we have class i or not, see Sec. 2.2.1. To estimate $p(\boldsymbol{x}|Y_i)$, we split the training data into its classes and estimate the density function for each class separately. To assign a class label to a new point, we evaluate the estimated density functions $\hat{p}(\boldsymbol{x}|Y_i)$ at the new point $\boldsymbol{x}$, compute the posterior density $\hat{p}(Y_i|\boldsymbol{x})$ with Bayes' theorem, and respond with the class label associated to the density function which yields the highest value. In contrast to the naive Bayes classifier, we do not have to invoke the conditional independence assumption because we estimate the joint probability density function [102].

Whereas in the classical approach one usually has to distinguish between binary and multi-class classification problems, our method can cope with more than two classes in a natural way. We can also derive a natural and stochastically motivated confidence value: If we assign a label to a data point for which one density function yields a much higher value than the others, it is very likely that the data point is correctly classified. In contrast, if several density functions yield about the same value, the data point most probably lies in a region where different classes overlap and thus the class label cannot be assigned with high confidence. However, the information that a respond should be considered more like an educated guess than a founded statement is already valuable for certain applications, see, e.g., [176]. Furthermore, our classification method has all advantages of a generative model, e.g., generate new data points, compensate for class priors, and detect outliers [28, 115].

[1]Because we directly learn the class-conditional $p(\boldsymbol{x}|Y_i)$, this is sometimes called conditional learning [115].

Probabilistic, generative sparse grid model for classification We split the training data set $\mathcal{S}$ into k partitions $\mathcal{S}_1, \dots, \mathcal{S}_k$ with

$$\mathcal{S}_j = \{(\boldsymbol{x}_i, y_i) \in \mathcal{S} \,|\, y_i = j\},$$

such that the set $\mathcal{S}_j$ contains all M_j pairs $(\boldsymbol{x}_i, y_i)$ with class label $y_i = j$. We then construct the estimated probability density functions $\hat{p}(\boldsymbol{x}|Y_1), \dots, \hat{p}(\boldsymbol{x}|Y_k)$ with the sets $\mathcal{S}_1, \dots, \mathcal{S}_k \subset \mathcal{S}$ and the sparse grid method described in Sec. 7.1. The prior class probabilities are either simply constants with $\hat{p}(Y_i) = 1/M$, or are estimated from the fractions

$$\hat{p}(Y_i) = \frac{|\mathcal{S}_i|}{|\mathcal{S}|} = \frac{M_i}{M}.$$

To assign a class label to a new data point $\boldsymbol{x}$, we evaluate the estimated density functions $\hat{p}(\boldsymbol{x}|Y_1), \dots, \hat{p}(\boldsymbol{x}|Y_k)$, and compute with Bayes' theorem the posterior density

$$\hat{p}(Y_i|\boldsymbol{x}) \propto \hat{p}(\boldsymbol{x}|Y_i)\hat{p}(Y_i).$$

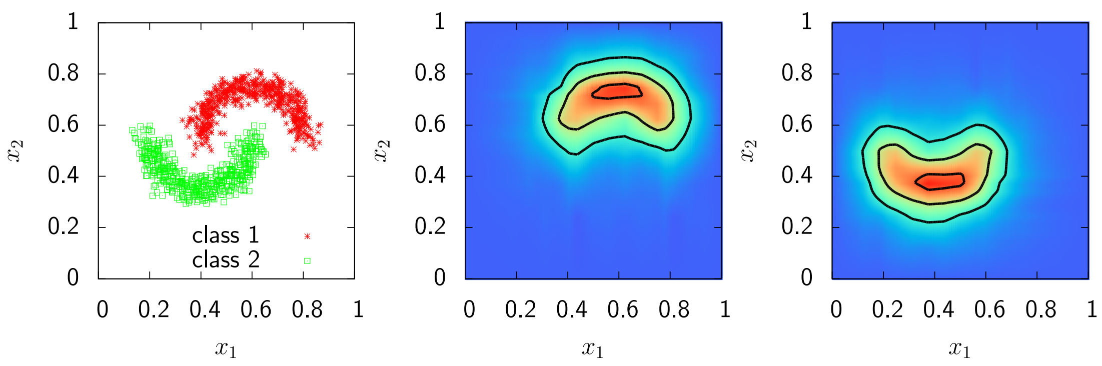
Note that we ignore the denominator $\hat{p}(\boldsymbol{x})$ in Bayes' theorem. The classifier $\hat{c}$ approximates the unknown function c and is then defined as

$$\hat{c}(\boldsymbol{x}) = \underset{i \in \{1,\dots,k\}}{\arg\max}\, \hat{p}(Y_i|\boldsymbol{x}). \tag{68}$$

We summarize our classification method based on sparse grid density estimation in four steps:

1. Split the training data set $\mathcal{S}$ into its separate classes $\mathcal{S}_1, \dots, \mathcal{S}_k$.
2. Construct the estimated the density functions $\hat{p}(\boldsymbol{x}|Y_1), \dots, \hat{p}(\boldsymbol{x}|Y_k)$ with the data sets $\mathcal{S}_1, \dots, \mathcal{S}_k$ on a sparse grid with the method discussed in Sec. 7.1.
3. For all $i = 1, \dots, k$, approximate the prior probability $p(Y_i)$ with a constant value $\hat{p}(Y_i) = 1/M$ or estimate it with $\hat{p}(Y_i) = M_i/M$.
4. Evaluate the classifier (68) to assign a label $y \in \{1, \dots, k\}$ to a data point $\boldsymbol{x} \in \mathbb{R}^d$, i.e.,
$$y = \hat{c}(\boldsymbol{x}).$$

Figure 27 shows a contour map of the two estimated density functions $\hat{p}(Y_1|\boldsymbol{x})$ and $\hat{p}(Y_2|\boldsymbol{x})$ corresponding to the two classes of the two moons data set. Since the data points are equally distributed between the two classes, the priors $\hat{p}(Y_1)$ and $\hat{p}(Y_2)$ have no influence. Clearly, density function $\hat{p}(Y_1|\boldsymbol{x})$ evaluates to greater values in those regions where most data points with label 1 lie than in the rest of the domain. Just as density function $\hat{p}(Y_2|\boldsymbol{x})$ yields greater values in the region of class 2. Furthermore, the contour lines of the two density functions approximate the shape of the boundary of the data point cluster with label 1 and label 2, respectively. Figure 27 also demonstrates the confidence value. For example, consider point $(0.4, 0.5)$. It lies between the two classes, thus it is very difficult to assign the correct class to this point. This fact is reflected by the two density functions. Both functions evaluate to about the same value at $(0.4, 0.5)$, which confirms that we can only make an educated guess but not a profound statement about the class label of $(0.4, 0.5)$.



(a) data (b) class 1 (c) class 2

Figure 27: The data points of the two moons data set and the contours of the corresponding two density functions.

Computational costs Let us now have a look at the classification method from a computational point of view. For both, the classification method based on density estimation as well as the one based on regression, we have to solve k systems of linear equations if we want to construct a classifier for a problem with $k > 2$ classes. However, the systems of linear equations have a crucial difference: Whereas the system matrix $\boldsymbol{R} + \lambda \boldsymbol{C}$ for the density estimation problem consists of the matrices $\boldsymbol{R}$ and $\boldsymbol{C}$ with dimensions $N \times N$, the system matrix $\frac{1}{M}\boldsymbol{B}\boldsymbol{B}^T + \lambda \boldsymbol{C}$ corresponding to the classification method based on regression includes a matrix $\boldsymbol{B}$ with dimension $N \times M$ where M is the number of data points. Since usually the matrix $\boldsymbol{B}\boldsymbol{B}^T$ is not assembled but only a procedure for the matrix-vector product with $\boldsymbol{B}$ and $\boldsymbol{B}^T$ is provided, the product with $\boldsymbol{B}\boldsymbol{B}^T$ still depends on the number of data points M and thus a matrix-vector product with a vector of size M has to be performed in each iteration of the CG method, cf. Sec. 2.4.2. This is a severe drawback of the classification method based on sparse grid regression when it comes to large data sets. In contrast, the minimization problem corresponding to our classification method based on sparse grid density estimation relies on a system matrix which is independent from the data points. Therefore, the matrix-vector product with the system matrix is completely decoupled from the number of data points M. Furthermore, we set the operator $\boldsymbol{C}$ to the identity matrix $\boldsymbol{I}$, cf. Sec. 2.4.2 and Sec. 7.1.

After having constructed a classifier with the density-based or the regression-based method we can classify a new point $\boldsymbol{x} \in \mathbb{R}^d$ by evaluating k sparse grid functions. Thus, classifying a new data point with the method based on density estimation has the same costs as with the method based on regression. Note that evaluating a sparse grid function is in $\mathcal{O}(\ell^d)$ because basis functions of a hierarchical increment have disjoint support, see Sec. 2.4.2.

7.2.2. Offline/Online Splitting

In the following, we describe an Offline/Online splitting of the computational procedure of the classification method based on sparse grid density estimation. In the Offline phase, the system matrix $\boldsymbol{R} + \lambda \boldsymbol{I}$ is pre-computed and stored in a decomposed form. When we get data points in the Online phase, the pre-processed matrix is loaded and the corresponding system of linear equation is solved in $\mathcal{O}(N^2)$ instead of in $\mathcal{O}(N^3)$. Two matrix decompositions of the system matrix are proposed and their properties in the context of the classification problem are discussed.

Offline/Online splitting in the context of classification As we have discussed above, one advantage of our classification method is that the system matrix is truly decoupled from the number of the training data points M. However, the system matrix $\boldsymbol{R} + \lambda \boldsymbol{I}$ of the density estimation problem is even independent from the data points themselves. Whereas an entry B_{ij} of the matrix $\boldsymbol{B}$ in the system matrix $\frac{1}{M}\boldsymbol{B}\boldsymbol{B}^T + \lambda \boldsymbol{C}$ of the regression problem is the hierarchical basis function ϕ_i evaluated at the data point $\boldsymbol{x}_j$, an entry R_{ij} for the density estimation problem is the L_2 dot product of the two hierarchical basis functions ϕ_i and ϕ_j which does not depend on the data points. As far as the density estimation problem is concerned, the data points influence only the right-hand side but not the system matrix of the system of linear equations (59). This allows us to introduce an Offline/Online splitting of the computational procedure. Even though very common in model order reduction, the data mining community has paid little attention to such Offline/Online splittings of the computational procedure.

Such an Offline/Online scheme pays off if we compensate the costly Offline phase by repeating the Online phase many times or if our (real-time) application requires a classifier immediately after new training data has been provided. Such scenarios can be found in, e.g., online learning or data stream mining [78, 18, 77]. Another example is cross validation. There we want to construct a classifier for different regularization parameters λ with different training data sets and test them on a test data set. Thus, we have to construct many classifiers where only the parameter λ as well as the training and test data set change.

Matrix pre-computation Here we want to pre-compute the system matrix $\boldsymbol{R} + \lambda \boldsymbol{I}$ of the density estimation problem (59). Note that we set the regularization operator $\boldsymbol{C}$ to the identity matrix $\boldsymbol{I}$. We would like to emphasize once more that the following decomposition is not possible in case of the classical sparse grid classification method based on regression because there the system matrix depends on the data points.

Let $\mathcal{V}_\ell^{(1)}$ be the sparse grid space of level ℓ and dimension d spanned by the hierarchical basis Φ_ℓ. To explicitly form the matrix $\boldsymbol{R} + \lambda \boldsymbol{I}$ corresponding to $\mathcal{V}_\ell^{(1)}$ we can employ the already available matrix-vector product procedures by multiplying with the unit vectors. Especially for higher dimensional data sets, this becomes rather expensive because one matrix-vector product is in $\mathcal{O}(2^d N)$ which is only linear in the number of grid points N but with the factor 2^d depending exponentially on the dimension d. Therefore, in our case, where we want to assemble the system matrix, it is better to follow the naive approach where we explicitly compute the L_2 dot products of all combinations of the basis functions. Just as N matrix-vector products with complexity $\mathcal{O}(2^d N)$, this scales also quadratically with the number of grid points N but we do not have the factor 2^d anymore, see, e.g., [119] for a detailed algorithm.

A key question is how to store the matrix during the Offline phase. We discuss here an LU decomposition and an eigendecomposition of the system matrix.

LU decomposition Since we want to solve a system of linear equations, an LU decomposition is an obvious choice. Thus, in the Offline phase, we compute the LU decomposition and store the lower left and upper right triangle matrices. They are then used in the Online phase to solve the corresponding system of linear equations with backward and forward substitution which is in $\mathcal{O}(N^2)$ only. A severe drawback of the LU decomposition is that we have to fix the regularization parameter λ already in the Offline phase because the decomposition is completely different for $\boldsymbol{R}+\lambda_1\boldsymbol{I}$ and $\boldsymbol{R}+\lambda_2\boldsymbol{I}$ with $\lambda_1 \neq \lambda_2$, see [50] and the references therein. This is indeed a disadvantage as cross validation with respect to the regularization parameter λ is one key application of such an Offline/Online splitting. There we have to solve a whole set of systems $\boldsymbol{R}+\lambda_1\boldsymbol{I}, \dots, \boldsymbol{R}+\lambda_m\boldsymbol{I}$ where only the regularization parameters $\lambda_1, \dots, \lambda_m$ are changed.

However, it has also been shown that the sparse grid density estimation method is not very sensitive to different parameters λ for huge data sets, see Sec. 7.1.4. Since the LU decomposition is cheaper to compute than the eigendecomposition discussed in the following paragraph, the LU decomposition might be used in cases where also the costs of the Offline phase are crucial.

Eigendecomposition If we store the eigendecomposition of the matrix $\boldsymbol{R}$, we are able to vary the parameter λ, and the Online phase is still in $O(N^2)$. Let $\boldsymbol{R} = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T$ be the eigendecomposition of $\boldsymbol{R}$ where $\boldsymbol{V}$ is an orthonormal matrix and $\boldsymbol{D}$ a diagonal matrix. Such an eigendecomposition exists because $\boldsymbol{R}$ is a Gram matrix with respect to the L_2 dot product and the basis functions in Φ_ℓ are linearly independent. We store $\boldsymbol{V}$ and $\boldsymbol{D}$ in the Offline phase. To construct a classifier in the Online phase, we have to solve the system of linear equations $(\boldsymbol{R}+\lambda\boldsymbol{I})\boldsymbol{\alpha} = \boldsymbol{b}$ to obtain the coefficients $\boldsymbol{\alpha}$, i.e., we have to compute $(\boldsymbol{R}+\lambda\boldsymbol{I})^{-1}\boldsymbol{b}$. This can be accomplished with the stored matrices $\boldsymbol{V}$ and $\boldsymbol{D}$ as follows

$$(\boldsymbol{R}+\lambda\boldsymbol{I})^{-1}\boldsymbol{b} = (\boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T + \lambda\boldsymbol{V}\boldsymbol{V}^T)^{-1}\boldsymbol{b} = \boldsymbol{V}\left(\boldsymbol{D}+\lambda\boldsymbol{I}\right)^{-1}\boldsymbol{V}^T\boldsymbol{b}.$$

Because $\boldsymbol{V}$ is orthonormal we have $\boldsymbol{V}^{-1} = \boldsymbol{V}^T$. The matrix $\boldsymbol{D}+\lambda\boldsymbol{I}$ is an $N\times N$ diagonal matrix which can be easily inverted in $O(N)$. Therefore, to solve the system (59), we only have to invert the diagonal matrix $\boldsymbol{D}+\lambda\boldsymbol{I}$ and perform three matrix-vector products with $N\times N$ matrices. Hence, the Online phase is still in $O(N^2)$ and we can modify the parameter λ in the Online phase. Note that even though we invert the diagonal matrix $\boldsymbol{D}+\lambda\boldsymbol{I}$ we have not experienced any numerical instabilities for the numerical examples in Sec. 7.2.3. Computing the eigendecomposition in the Offline phases takes usually distinctly longer than computing the LU decomposition. Nevertheless, we will always employ the eigendecomposition in the following because it gives us greater flexibility with respect to the regularization parameter λ.

7.2.3. Numerical Results

In this section, we report on the performance of the proposed classification method based on sparse grid density estimation. Note that the reported accuracy results do not depend on whether we use the Offline/Online splitting or not, of course. We compare with the classification approach based on sparse grid regression. We always employ sparse grids without basis functions at the boundary because we simply transform the data set into the $[0.1, 0.9]^d$ cube if necessary [152]. The selection of the regularization parameter λ was performed with five-fold cross validation. To tackle the multi-class classification problem with the classical approach, we compute a sparse grid regression function for each class separately as discussed in Sec. 2.4.2. Note that in rare cases our method can achieve slightly better results for the test data set than for the training set due to the optimization problem originating from density estimation which does not target explicitly the classification error.

Regular sparse grids Let us first consider the results with density functions estimated on regular sparse grids. In Tab. 8, we report the results of the regression-based and the density-based method on nine data sets. The computations with the density-based method were performed with the prior $\hat{p}(Y_i) = M_i/M$ and $\hat{p}(Y_i) = 1/M$, respectively. For all nine data sets, our method achieves competitive results. For six out of the nine data sets we obtain either 100% test accuracy or a better test accuracy than with the regression-based approach. For the three other data sets (svmguide1, olives, and shuttle) the classical approach is only slightly better. These results clearly suggest that the data sets can be learned with our density-based classification method. Hence, our method can cope with both artificial (two-moons, 3S, and svmguide1) as well as real-world data sets (old faithful, iris, bupa, olives, shuttle, and oil flow).

In Fig. 28, we show a dimension-wise plot of the iris flower data set. The four-dimensional data points are projected onto two dimensions each. We also show contour plots of the three density functions corresponding to the three classes of the data set. If we compare the dimension-wise plot of the data set with the dimension-wise plots of the density functions, we see that the density functions evaluate to higher values near the center of the corresponding class. This visualizes that it is reasonable to use the density value as a measure of confidence in the membership to its class.

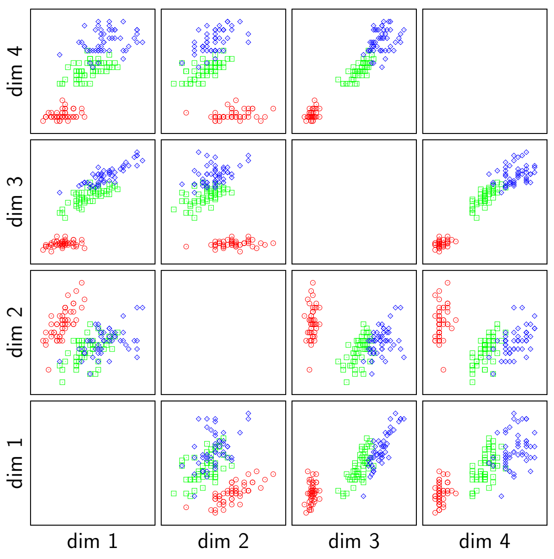
Adaptive sparse grids Let us now come to the results for adaptively refined sparse grids, see Tab. 9. We employ the standard sparse grid refinement criterion based on the absolute value of the hierarchical coefficients. We only consider the data sets for which we did not obtain 100% test accuracy in Tab. 8 and skip the data sets with more than six dimensions because we will discuss them in the context of the Offline/Online splitting below. We need distinctly fewer sparse grid points to achieve a better or similar accuracy if we employ adaptive refinement, i.e., if we adapt the grid to the data set.

			regression-based		density-based $\hat{p}(Y_i) = M_i/M$		density-based $\hat{p}(Y_i) = 1/M$	
	d	gp	train[%]	test[%]	train[%]	test[%]	train[%]	test[%]
two moons	2	17	100.00	**100.00**	100.00	**100.00**	100.00	**100.00**
old faith.	2	17	100.00	97.82	100.00	**100.00**	100.00	**100.00**
3S	3	351	100.00	**100.00**	99.50	**100.00**	99.83	98.52
iris	4	769	100.00	80.00	97.77	93.33	97.77	**100.00**
svmguide1	4	769	97.24	**96.15**	96.43	94.55	94.56	95.52
bupa	6	545	79.65	67.27	58.62	56.36	70.68	**69.09**
olives	8	6,401	100.00	**100.00**	100.00	97.72	97.12	96.59
shuttle	9	9,439	99.74	**99.64**	83.12	83.82	97.67	97.75
oil flow	12	3,249	90.06	66.52	85.96	85.73	86.57	**86.75**

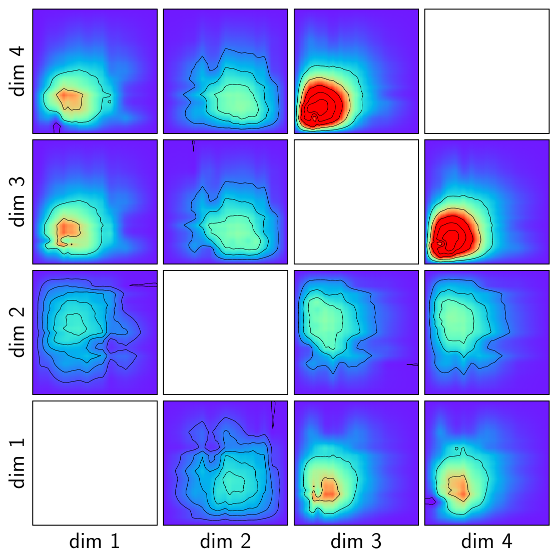
Table 8: Percent of correctly classified data points for the regression-based and our density-based sparse grid classification method: Results for the density-based method are shown with prior ($\hat{p}(Y_i) = M_i/M$) and ($\hat{p}(Y_i) = 1/M$), respectively. Besides the results on the training and test sets, we also report for each data set the dimension (d) and the number of grid points (gp). In six out of nine data sets, our density-based method performs better than the classical, regression-based approach.

		density, $\hat{p}(Y_i) = M_i/M$			density, $\hat{p}(Y_i) = 1/M$		
	d	gp	train[%]	test[%]	gp	train[%]	test[%]
iris	4	160	96.29	**100**	117	96.29	**100**
svmguide1	4	457	95.01	**95.32**	988	95.24	95.22
bupa	6	1,729	71.72	67.27	268	71.03	**70.90**

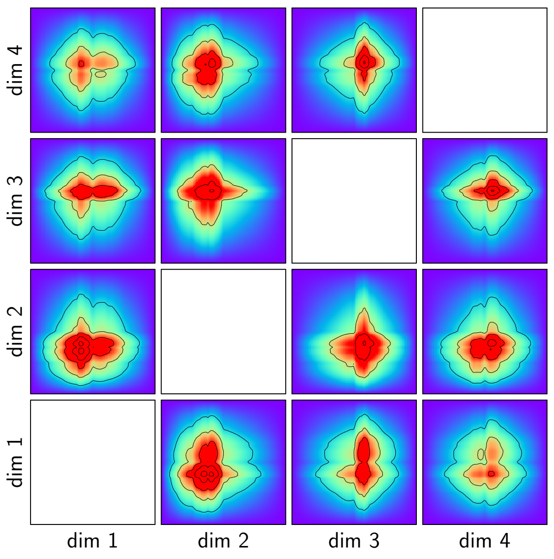
Table 9: The results for the density-based classification method on adaptive sparse grids are listed. With adaptive refinement, we achieve similar results as on regular sparse grids but with distinctly fewer grid points, cf. Tab 8.



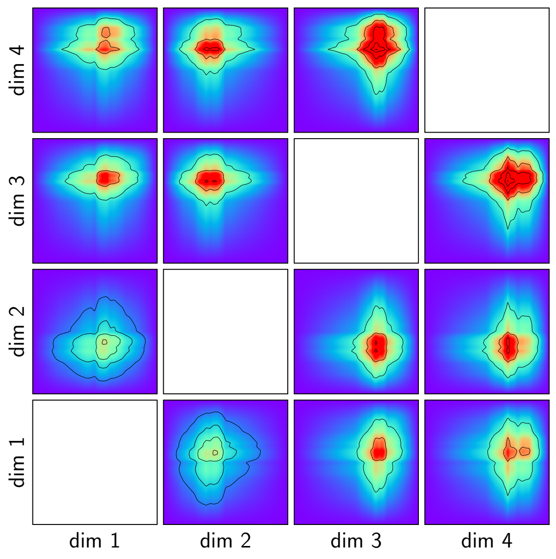
(a) data



(b) class 1



(c) class 2



(d) class 3

Figure 28: Projection of the iris flower data set onto two dimensions each and the density functions corresponding to the three classes of the data set: All three density functions evaluate to high values near the center of their corresponding class.

Overall, we obtain either better or almost as good results as with the classical approach. And, just as with the classical approach, we can drastically improve the results using adaptively refined sparse grids.

Offline/Online splitting Finally, we want to report on the runtime of the proposed method with and without the Offline/Online splitting. We only consider the olives, shuttle, and oil flow data sets because only for those the classification took longer than one second. In Tab. 10, we show the runtime of the regression-based and the density-based method. In case of the density-based method, we report the runtime without (standard) and with (pre-computed) Offline/Online splitting. Without the Offline/Online splitting we did not assemble the system matrix but used the matrix-vector product procedures in combination with the CG method to solve the system of linear equations. We stopped either after 50 CG iterations or when the norm of the residual was below 10^{-10}.

Let us first compare the runtimes of the regression-based with the density-based method without Offline/Online splitting. The olives data set is eight-dimensional and consists of a few data points only. We see hardly a difference in the runtime of the regression-based and the density-based method. However, the shuttle data set has also only eight dimensions but about 43,500 data points. Since the complexity of the matrix-vector product with the system matrix $\boldsymbol{R}$ of the density-based approach is independent form the data points, the density-based method is about five times as fast as the regression-based method where the matrix-vector product depends on the number of data points. We have the exact opposite situation for the oil flow data which is 12-dimensional and consists of only a few data points. Since the matrix-vector product with $\boldsymbol{R}$ is in $\mathcal{O}(2^d N)$, the density-based approach is slower. However, in all cases, we can drastically reduce the runtimes of the density-based method if we switch on the Offline/Online splitting. For all data sets, the density-based method with Offline/Online splitting is by far the fastest. For the shuttle data set, the construction of the right-hand side of the system (59) becomes the most expensive part of the computation. The right-hand side cannot be pre-computed because it involves the data points. Overall, we gain speed ups of up to 100 compared to the regression-based method and of up to 500 compared to the density-based method without Offline/Online splitting. The measurements where performed on an Intel Core i7-870 with a single thread only.

7.3. Ensemble Learning with Sparse Grid Classifiers

Let us now return to the classical sparse grid classification method based on regression. From Sec. 2.4.2 and the comprehensive study in [153], we know that the method can be greatly enhanced by adaptively refining the underlying sparse grid. The drawback of adaptive sparse grids is that the corresponding adaptive algorithms are distinctly harder to implement than their regular counterparts. In particular, to develop an efficient and parallel implementation, a thorough understanding of the hardware is necessary [108]. That is why we consider here ensemble learning to improve the sparse grid classification method based on regression without employing adaptive sparse grids. We construct

			regression-based		density-based standard		density-based pre-computed	
	d	M	solve[s]	total[s]	solve[s]	total[s]	solve[s]	total[s]
olives	8	348	55	55	60	61	<1	<1
shuttle	8	43,500	1,006	1,018	215	242	10	36
oil flow	12	1,318	25	25	567	567	<1	1

Table 10: Runtime in seconds of the regression-based approach and the density-based classification method without (standard) and with (pre-computed) Offline/Online splitting are reported. We split the (Online) runtime in the time spent to solve the system of linear equations and the total time including all reading, writing and pre-processing of the data.

several classifiers relying only on regular sparse grids and combine their predictions to classify a new data point.

7.3.1. Ensemble Learning and AdaBoost

In ensemble learning, multiple classifiers, so-called base or weak learners, form a team. The individual decisions of the team members are then combined to obtain the final prediction. Ensembles work well if the base learners are accurate and diverse [55, 100]. A base learner is accurate if it is better than random guessing. Usually, this can be easily achieved because this is the goal of every classification method. Base learners are diverse if they make different errors on new data points. We can construct diverse base learners by training each of them on a different data set. Many strategies exist to construct these individual training data sets. We consider bagging, stacking, and boosting.

Bagging, stacking, and boosting Let $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{M} \subset \mathbb{R}^d \times \{-1, 1\}$ be the training set of our binary classification problem.

Bagging constructs the training data sets $\mathcal{S}_1, \dots, \mathcal{S}_m$ for the base learners by drawing samples randomly with replacement from the original training data set $\mathcal{S}$. The training sets $\mathcal{S}_1, \dots, \mathcal{S}_m$ are called bootstrap replicates of the original training set and contain, on the average, $100 - 100/\mathrm{e} \approx 63.2\%$ of the original training set, with several samples appearing multiple times. The final answer is obtained by a majority vote between the team members [32].

Stacking is highly related to cross validation where the training data $\mathcal{S}$ is partitioned into subsets $\mathcal{S}_1, \dots, \mathcal{S}_m \subset S$ and classifiers are trained on each of the subsets and validated on the remaining ones. In the end, the classifier with the best result is kept and all others are discarded. Of course, this is very wasteful. Therefore, stacking keeps all classifiers and combines them in an ensemble [189, 102]. Let $g_1, \dots, g_m$ be the classifiers trained on the subsets $\mathcal{S}_1, \dots, \mathcal{S}_m$. The final classifier is

$$\hat{c}(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{m} \beta_i g_i(\boldsymbol{x})\right),$$

where the stacking weights $\{\beta_1, \ldots, \beta_m\}$ are given by the optimization problem

$$\boldsymbol{\beta} = \arg\min_{\boldsymbol{w}} \sum_{i=1}^{M} \left(y_i - \sum_{i=1}^{m} w_i g_i(\boldsymbol{x}) \right)^2 . \tag{69}$$

Note that stacking has also been applied to density estimation in [172].

Boosting also achieves diverse base learners by training them on different data sets. But in contrast to bagging and stacking, it assigns weights to the data points which are then gradually adapted during the construction of the base learners such that more weight is placed on those data points that are hard to classify. The most popular boosting algorithm is AdaBoost which we discuss in the following paragraph.

AdaBoost AdaBoost has been introduced in [75]. It had a significant impact on the machine learning community and is ranked as one of the ten most important algorithms in data mining [190]. There are several versions of AdaBoost, see, e.g., [76]. We restrict the following discussion to the most common version called AdaBoost.M1, see Alg. 5

AdaBoost produces a sequence of base classifiers $g_1, \ldots, g_m$ which are sequentially trained on an ever-changing training data set. To accomplish this, a weight $w_i > 0$ is introduced for each data point in $\mathcal{S}$. These weights have to be taken into account by the base learners. Initially, all weights are equal and thus the first base learner g_1 is trained in the usual manner. However, in the subsequent iterations of AdaBoost, the weights change such that more weight is placed on data points which are hard to classify. The final AdaBoost classifier

$$\hat{c}(\boldsymbol{x}) = \operatorname{sign}\left(\sum_{i=1}^{m} \beta_i g_i(\boldsymbol{x}) \right) \tag{70}$$

is a linear combination of the base learners $g_1, \ldots, g_m$ with coefficients $\beta_1, \ldots, \beta_m$.

Let us have a closer look at the AdaBoost.M1 algorithm shown in Alg. 5. The weights are initialized with $w_i = 1/M$. In each iteration, we train a base classifier g_j and compute the weighted classification error

$$\mathrm{err}_j = \frac{\sum_{i=1}^{M} w_i I\left(y_i \neq g_j\left(\boldsymbol{x}_i\right)\right)}{\sum_{i=1}^{M} w_i},$$

where $I\left(y_i \neq g_j\left(\boldsymbol{x}_i\right)\right)$ is 1 if the data point $(\boldsymbol{x}_i, y_i)$ is misclassified by g_j and else 0. An error err_j greater or equal to 0.5 means that our base learner performs not very good and thus this is a signal to stop. If the error is below 0.5, we can compute the coefficient β_j to determine the weight of the decision of g_j in the final AdaBoost classifier (70). Figure 29 shows the dependence of the coefficient β_j on the weighted classification error err_j. It follows the intuitive expectation that base learners with a low error should get a high coefficient and thus a significant vote in the final prediction. In the end of each iteration, the weights corresponding to the data points are recomputed. If the data point $(\boldsymbol{x}_i, y_i)$ has been correctly classified by g_j, its weight is reduced, and if it has been misclassified, it is increased. Note that the weights stay always positive. The loop continues until m base learners and coefficients have been computed.

Algorithm 5 AdaBoost.M1 [75]

```
 1: procedure AdaBoost.M1($\mathcal{S}$, $m$)
 2:     Set all weights $w_i = 1/M, i = 1, \ldots, M$
 3:     for $j = 1, \ldots, m$ do
 4:         Fit classifier $g_j$ to data with weights $w_1, \ldots, w_M$
 5:         Compute the weighted error
                $$\mathrm{err}_j \leftarrow \frac{\sum_{i=1}^{M} w_i I(y_i \neq g_j(\boldsymbol{x}_i))}{\sum_{i=1}^{M} w_i}$$
 6:         if $\mathrm{err}_j \notin (0, 0.5)$ then
 7:             return $\hat{c}(\boldsymbol{x}) = \operatorname{sign}\left[\sum_{i=1}^{j-1} \beta_i g_i(\boldsymbol{x})\right]$
 8:         end if
 9:         $\beta_j \leftarrow 0.5 \cdot \log((1 - \mathrm{err}_j)/\mathrm{err}_j)$
10:         Set new weights $w_i, i = 1, \ldots, M$
                $$w_i \leftarrow \begin{cases} w_i \cdot \mathrm{e}^{-\beta_j}, & \text{if } y_i = g_j(\boldsymbol{x}_i), \\ w_i \cdot \mathrm{e}^{\beta_j}, & \text{if } y_i \neq g_j(\boldsymbol{x}_i) \end{cases}$$
11:     end for
12:     return $\hat{c}(\boldsymbol{x}) = \operatorname{sign}\left[\sum_{i=1}^{m} \beta_i g_j(\boldsymbol{x})\right]$
13: end procedure
```

AdaBoost is a forward-stagewise additive modeling approach with respect to the exponential loss criterion [102]. In each iteration, a base learner g_j with coefficient β_j is added to an expansion

$$\hat{c}_{j-1} = \sum_{i=1}^{j-1} \beta_i g_i.$$

AdaBoost computes the coefficient β_j such that the loss

$$\sum_{i=1}^{M} L\left(y_i, \hat{c}_{j-1}(\boldsymbol{x}_i) + \beta_j g_j(\boldsymbol{x}_i)\right) = \sum_{i=1}^{M} \exp\left(-y_i\left(\hat{c}_{j-1}(\boldsymbol{x}_i) + \beta_j g_j(\boldsymbol{x}_i)\right)\right)$$

is minimized. The base learners and the coefficients of the previous $j-1$ iterations are not modified. This is a completely different approach than for the sparse grid classification method of Sec. 2.4.2 where all coefficients $\alpha_1, \ldots, \alpha_N$ are optimized simultaneously.

AdaBoost, and ensemble learning in general, fits very well to the idea of model order reduction. Instead of combining many simple basis functions (e.g., linear basis functions or radial basis functions), a few problem-dependent basis functions (base learners) are built and then used to form the classifier. This is very similar to model reduction with POD or the RBM.

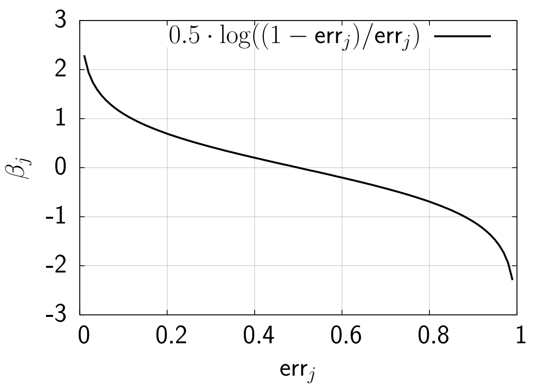


Figure 29: Shows the weight β_j for a base learner g_j with weighted error err_j. For $\text{err}_j > 0.5$ the base learner performs worse than random guessing and thus the weight β_j becomes negative. This motivates the stopping criterion in Alg. 5.

7.3.2. Sparse Grid Base Learners for AdaBoost

We want to employ the sparse grid classification method based on regression to build the base learners for AdaBoost. In the previous section, we have seen that AdaBoost computes a weight for each data point which has to be taken into account when constructing the base learners. Thus, we cannot use the optimization problem (23) of the sparse grid classification method of Sec. 2.4.2.

Let $\boldsymbol{w} = [w_1, \ldots, w_M]^T$ be the AdaBoost weights associated to the data points in the training set $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^M \subset \mathbb{R}^d \times \{-1, 1\}$. To construct a sparse grid base learner $g \in \mathcal{V}_\ell^{(1)}$, we change the minimization problem (23) to the weighted minimization problem

$$\arg\min_{f \in \mathcal{V}_\ell^{(1)}} \frac{1}{M} \sum_{i=1}^M w_i \left(f(\boldsymbol{x}_i) - y_i\right)^2 + \lambda \|\Lambda f\|_{L_2}^2. \tag{71}$$

We weight the error $(f(\boldsymbol{x}_i) - y_i)^2$ introduced by the data point $(\boldsymbol{x}_i, y_i)$ with the associated weight w_i. Analogously to the original minimization problem (23), this leads to a system of linear equations

$$\left(\frac{1}{M}\boldsymbol{D}\boldsymbol{B}\boldsymbol{B}^T + \lambda \boldsymbol{C}\right)\boldsymbol{\alpha} = \frac{1}{M}\boldsymbol{B}\boldsymbol{D}\boldsymbol{y}, \tag{72}$$

where the matrix $\boldsymbol{D} \in \mathbb{R}^{M \times M}$ is a diagonal matrix with the weights $w_1, \ldots, w_M$ on the diagonal. All the computational procedures for the matrix-vector product with the matrices $\boldsymbol{B}$, $\boldsymbol{B}^T$ and $\boldsymbol{C}$ can be reused. The system (72) is not restricted to regular sparse grids, but can also be employed in locally adaptive settings.

It is not straightforward to employ the sparse grid classification method based on density estimation to construct the base learners. The corresponding density estimation minimization problem (58) does not contain the labels y_i and thus no error term similar to $(f(\boldsymbol{x}_i) - y_i)^2$. Hence, a simple weighing as in (72) is not possible.

7.3.3. Numerical Results

Sparse grid classifiers can be greatly enhanced if adaptive sparse grids are employed, see [153, 154, 156, 155]. Instead of having a fixed, regular grid structure, new grid points are added in those regions where a higher accuracy is required, i.e., where those data points are which are hard to classify. Thus, the adaptive refinement of the grid guides the classification method by forcing it to focus on those problematic data points. However, as we have argued above, the drawback is that adaptive algorithms are harder to implement and, in particular, harder to optimize on modern hardware, see the extensive studies in [109, 108, 107].

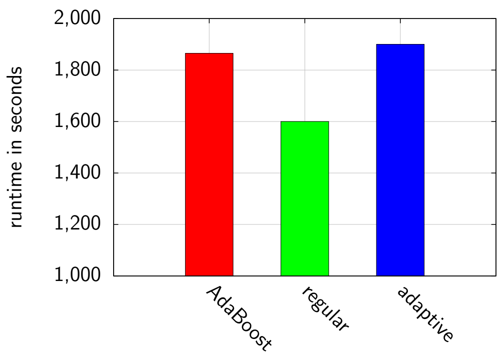
With the weights associated to the data points, AdaBoost also guides the classifier by pointing out the data points on which it should concentrate. This means, with AdaBoost, we also adapt the classifier to the current problem at hand but we can stay on regular sparse grids. That is why we want to investigate if we can achieve a similar accuracy with AdaBoost as with adaptive sparse grids. In the following, we consider runtimes only on an Intel i7-2600 CPU with four cores. For other architectures, including GPUs and combinations of CPUs and GPUs, see [106]. We consider two data sets: The five-dimensional checkerboard with a $3 \times \cdots \times 3$ pattern and the codRNA data set from a real-world biochemical application.

Checkerboard The checkerboard consists of 60,000 training and 20,000 test data points. It has $3^5 = 243$ different areas where the classifier has to jump from 1 to -1 at the borders. It does not contain any noise and thus we aim for a classification accuracy of one.

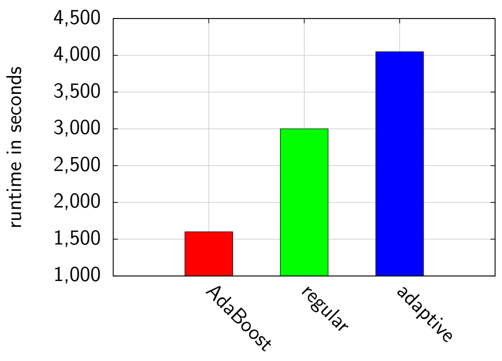
We first consider uniform basis functions with grid points at the boundary. In case of AdaBoost we have three base learner where each of them uses a sparse grid of level six. Overall, we have $\approx$ 300,000 degrees of freedom (DoFs). We compare AdaBoost with a classifier on a regular sparse grid of level seven ($\approx$ 300,000 DoFs) and with a classifier on an adaptive sparse grid with 14 refinement steps ($\approx$ 220,000 DoFs). Fig. 30a clearly shows that even though with the adaptive sparse grid we can learn the data set on distinctly fewer sparse grid points, it is not competitive in terms of runtime.

Let us now employ nonuniform basis functions, see Sec. 2.4. Nonuniform basis functions allow us to ignore the grid points at the boundary. However, a nested four-way switch statement is needed inside the innermost loop of the computing kernel [106]. Hence, it is distinctly more expensive to evaluate a nonuniform than a uniform basis function. With nonuniform basis functions, AdaBoost requires only five base learners with $\approx$ 61,000 grid points each to achieve the target accuracy of one. For the classifier on the regular sparse grid, about 590,000 grid points are necessary and in the case of the adaptive sparse grid we need 25 refinement steps. Due to the expensive evaluation of nonuniform basis function, the situation has changed completely with respect to the runtimes. AdaBoost clearly outperforms all other classifiers, see Fig. 30b.

Non-coding RNAs The eight-dimensional codRNA data set is used for computational pre-screening of RNAs. Non-coding RNAs (ncRNA) are transcripts that have function without being translated to protein. They are difficult and expensive to detect with common biochemical screens and thus it is better to first look for ncRNA candidates computationally, and then verify them biochemically [176]. Because the biochemical screening is costly, with respect to time and money, a computational method should



(a) uniform basis functions



(b) nonuniform basis functions

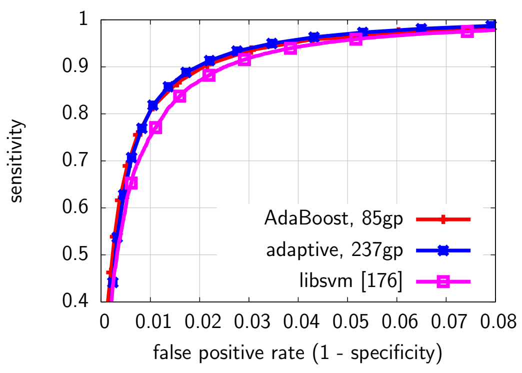
Figure 30: Runtimes of classification with AdaBoost, regular, and adaptive sparse grids for the five-dimensional checkerboard data set.

minimize the number of false positives, i.e., sequences that are not ncRNA but are classified as ncRNA by the method. Thus, we are more interested in a high specificity than in a high sensitivity, where

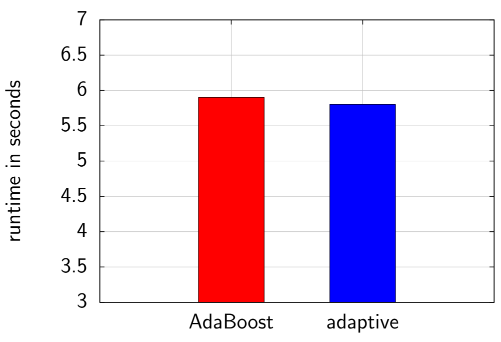
$$\text{sensitivity} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}},$$

$$\text{specificity} = \frac{\text{true negatives}}{\text{true negatives} + \text{false positives}}.$$

Just as in [176], we visualize this with a ROC (receiver operating characteristic) curve, see Fig. 31a. We want our method to have a high specificity and sensitivity, i.e., we want the curve to be in the top left corner. We compare the AdaBoost approach with an adaptive sparse grid classifier and a support vector machine (SVM) classifier (libsvm [49]) used in [176]. The AdaBoost classifier consists of a team of four sparse grid base learners. For the adaptive sparse grid classifier we performed two refinement steps, and in each step we refined two grid points. Overall, we obtain with AdaBoost the same result as with adaptive sparse grids. Both sparse grid approaches are distinctly better than the SVM classifier. We obtain a similar result with respect to the runtimes, see Fig. 31b. Note that libsvm is more than 100 times slower (706 seconds) than our approach. It is not included in the plot because libsvm does not support multi-threading and offers no vectorization support.



(a) ROC curve



(b) runtimes

Figure 31: In (a) the ROC curve obtained using AdaBoost, adaptive sparse grids, and libsvm is shown. Both sparse grid methods perform better than libsvm. In (b) we illustrate the runtimes of AdaBoost and the adaptive sparse grid classifier. The runtime of the SVM classifier is missing because libsvm has no multi-threading and vectorization support.

8. Case Study: Thermal Conduction Problem and Heat Shield

In the previous section, we have tackled classical learning problems with supervised learning methods based on sparse grids. Now, in this section, we employ supervised learning techniques to construct reduced-order models. We consider two popular thermal conduction problems: a thermal block and a heat shield. In both problems a block with a certain geometry and parametrized thermal conductivity properties is heated. We are then interested in the average temperature in specific parts of the spatial domain Ω.

So far, we focused on classification as one form of supervised learning. We had data sets of the kind $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^M \subset \mathbb{R}^d \times \{1, \dots, k\}$ where the labels $y_i \in \{1, \dots, k\}$ were discrete values. Now, we consider interpolation and regression as another form of supervised learning where the data sets are now subsets of $\mathbb{R}^d \times \mathbb{R}$ and thus the labels $y_i \in \mathbb{R}$ are real numbers.

In Sec. 8.1 we employ sparse grid interpolation to learn the output function $s : \mathcal{D} \to \mathcal{Y}$ of the thermal block and the heat shield problem. If we do not require the state vector $\boldsymbol{u}(\boldsymbol{\mu})$, this is a cheap, simple, and non-intrusive way to compute the output of interest. It has been shown in [47, 45, 44] that it is computationally feasible to extend this sparse grid interpolation approach to interpolate not only the output of interest but the whole state vector. However, the problem with the interpolation method is that it forces us to compute the snapshots for specific, pre-defined parameters. To be more precise, we have to provide the snapshots where the parameters coincide with the sparse grid points. Of course, this limits the scope of the method because in many situations we are given a set of snapshots and cannot influence for which parameters they have been computed. Therefore, in Sec. 8.2, we combine POD and sparse grid regression, to handle pre-computed snapshots for which the parameters do not necessarily have to coincide with sparse grid points.

8.1. Sparse Grid Interpolation of Output of Interest

We employ sparse grid interpolation to learn the input-output relationship of the problem $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$. We have already discussed that when we solve the problem $s : \mathcal{D} \to \mathcal{U} \to \mathcal{Y}$, we are usually only interested in the output of interest $s(\boldsymbol{\mu}) \in \mathcal{Y}$ and not in the state variable $u(\boldsymbol{\mu}) \in \mathcal{U}$, see Sec. 2.3. Nevertheless, most projection-based model reduction methods make the detour around the state variable $u(\boldsymbol{\mu})$ to compute the output of interest. We follow a different approach and directly interpolate the function $s : \mathcal{D} \to \mathcal{Y}$ without explicitly considering the state. Note that this is also the difference to the sparse grid methods presented in [47, 45, 44], were the full state vector $\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ is interpolated.

Interpolation of output of interest We have $\mathcal{D} \subset [0, 1]^d$ and $\mathcal{Y} \subset \mathbb{R}$. Let $\mathcal{V}_\ell^{(1)}$ be a sparse grid space of level ℓ and dimension d with N grid points. We construct the sparse grid interpolant $s_N \in \mathcal{V}_\ell^{(1)}$ of the full model $s^{\mathcal{N}}$ with the hierarchisation procedure, see Sec. 2.4. For higher parameter dimensions ($d > 3$), this is not feasible with ordinary tensor product grids, but with sparse grids we can cope with higher dimensions to some extent. After this, we only have to evaluate the sparse grid function s_N at a parameter $\boldsymbol{\mu} \in \mathcal{D}$ to obtain an approximation of the corresponding output of interest. In particular, we do not have a Galerkin projection into a space spanned by a reduced basis. Hence, we do not need to implement a solver for such a projection. This is a clear advantage over projection-based methods such as RBM and POD because for many (real-world) applications the implementation of the solver for the high-fidelity solution is a challenge on its own, let alone the development of a solver for the reduced system. Overall, this means our method is non-intrusive. We treat the full model $s^{\mathcal{N}}$ as a black box and do not need to know anything about the governing equations[2]. Of course, this lack of knowledge comes at a price. Usually, it is distinctly harder to achieve similar accuracies with non-intrusive as with intrusive methods.

We want to compare the reduced basis method to our sparse grid interpolation with respect to the number of operations needed for one evaluation of the reduced output function in the Online phase. For the following problems with d parameters, the reduced basis method with a reduced basis space of dimension n needs roughly $\mathcal{O}(n^3 + d \cdot n^2)$ operations, i.e., $\mathcal{O}(d \cdot n^2)$ to form the d stiffness matrices corresponding to the d parameters and $\mathcal{O}(n^3)$ to solve the corresponding system of linear equations [142]. In case of our sparse grid method, we only have to evaluate the sparse grid interpolant which is linear in the number of sparse grid points N. We do not consider Offline costs in detail here but we want to remark that the construction of a sparse grid interpolant is very expensive because for each grid point we have to compute the corresponding high-fidelity solution. However, it is also important to emphasize that the fast construction of the reduced basis space with the greedy strategy is only possible in very restricted settings where *a posteriori* error estimators are available, see Sec. 2.3.2.

We compute the maximum relative output error over a test set $\mathcal{T} \subset \mathcal{D}$

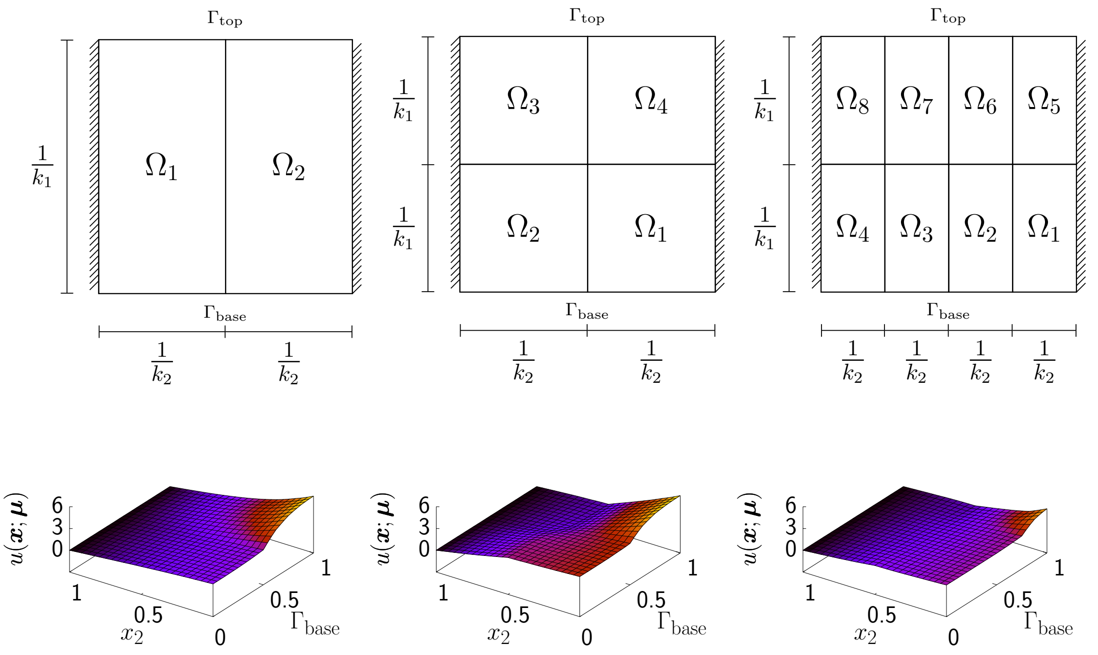
$$\max_{\boldsymbol{\mu} \in \mathcal{T}} \frac{|\tilde{s}(\boldsymbol{\mu}) - s^{\mathcal{N}}(\boldsymbol{\mu})|}{|s^{\mathcal{N}}(\boldsymbol{\mu})|} \tag{73}$$

where $\tilde{s}$ is either the reduced basis approximation or the sparse grid interpolant.

In the following, we compare our non-intrusive sparse grid interpolation method with the intrusive reduced basis method on two thermal conduction problems. We refer to [144, 149, 120] for more examples.

Thermal block The thermal block problem has become a very popular benchmark problem for model order reduction since it has been used in [142] to demonstrate the RBM. This is a (steady) heat conduction problem with conductivities as parameters. We have a square domain $\Omega = (0,1) \times (0,1)$ which is split into $k_1 \times k_2$ rectangular subdomains $\Omega_i, i = 1, \ldots, k_1 \cdot k_2$ with a different thermal conductivity in each subdomain. Thus, a parameter $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_d) \in \mathcal{D} = [0.1, 10]^d$ is a d-dimensional vector with $d = k_1 \cdot k_2$. We consider three different versions of the thermal block with two, four, and eight

[2]Additional knowledge about the properties of the problem and the equations can be used to improve the sparse grid interpolant. For instance, special refinement strategies or transformations might then be applicable.



(a) TB2 (b) TB4 (c) TB8

Figure 32: The geometries and surface plots of exemplary solution functions of three versions of the thermal block problem [149] are shown.

regions, respectively. Their geometry is shown in Fig. 32. The state variable $u^e(\boldsymbol{\mu})$ is the temperature which satisfies Laplace's equation in Ω. Following the geometry and notation in Fig. 32, we impose homogeneous Dirichlet conditions on the top boundary Γ_{top}, homogeneous Neumann conditions on the two sides, and Neumann 1 conditions on the bottom or base Γ_{base}. The output of interest

$$s^e(\boldsymbol{\mu}) = \int_{\Gamma_{\text{base}}} u^e(\boldsymbol{x};\boldsymbol{\mu})\,\mathrm{d}\boldsymbol{x} \tag{74}$$

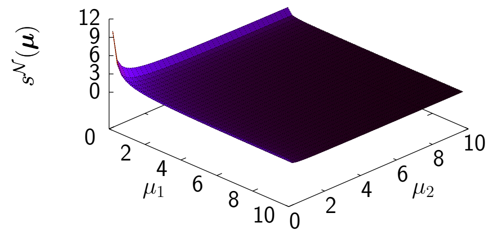
is the average temperature over the base Γ_{base}. We obtain an approximation $u^{\mathcal{N}}(\boldsymbol{\mu})$ of $u^e(\boldsymbol{\mu})$ by solving Laplace's equation for the parameter $\boldsymbol{\mu}$ in a finite element space $\mathcal{U}^{\mathcal{N}}$ with dimension $\mathcal{N}$. The state variable $u^{\mathcal{N}}(\boldsymbol{\mu})$ is then used to approximate (74) which we denote with $s^{\mathcal{N}}$. We refer to [149] for more details on the bilinear forms, involved spaces, and the finite element approximation.

Let $\tau : [0,1]^d \to [0.1, 10]^d$ with

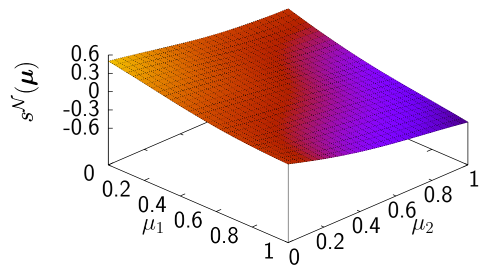
$$\tau(\boldsymbol{\mu}) = \left[0.1 \cdot \left(\frac{10}{0.1}\right)^{\mu_1}, \ldots, 0.1 \cdot \left(\frac{10}{0.1}\right)^{\mu_d}\right]^T. \tag{75}$$

We do not directly interpolate $s^{\mathcal{N}} : \mathcal{D} \to \mathbb{R}$ but $\hat{s}^{\mathcal{N}} : [0,1]^d \to \mathbb{R}$ with

$$\hat{s}^{\mathcal{N}}(\boldsymbol{\mu}) = \tau^{-1}(s^{\mathcal{N}}(\tau(\boldsymbol{\mu}))).$$



(a) linear scale



(b) logarithmic scale

Figure 33: We plot in (a) the interpolant of the output function of TB2 on the sparse grid of level 10 and in (b) the same output function but transformed with τ.

The transformation τ removes the peak of $s^{\mathcal{N}}$ near the corner $(0.1, 0.1, \dots, 0.1)$, see Fig. 33. For a motivation and further details on the transformation (75), we refer, again, to [149].

Let us first consider the thermal block with four subdomains, see Fig. 32b. Thus, we need a four-dimensional sparse grid to interpolate the transformed output function $\hat{s}^{\mathcal{N}}$. In Fig. 34a we compare the sparse grid interpolant with the reduced basis approximation. The interpolants are computed on regular sparse grids from level four to nine. The dimensions of the RBM spaces reach from five to 16. In the beginning, the sparse grid interpolation performs better than the RBM approximation. However, the error of the sparse grid interpolants decreases slower than the error of the RBM approximation. We have a very similar situation for the thermal block with eight subdomains, see Fig. 34b. Overall, if high-fidelity responds are required then the reduced basis method seems to be a better choice. However, the sparse grid interpolation is well suited if a rough approximation is sufficient. For instance, this is the case for visualization and computational steering where similar methods have already been successfully employed [47, 45, 44]. We would like to emphasize once more that our sparse grid interpolation method is non-intrusive in contrast to the reduced basis method. Especially for applications with a system of several governing equations usually only a solver for the high-fidelity solution is available and it is not feasible to add another solver for the reduced basis.

Heat Shield In Fig. 35a we show the geometry of a heat shield. It is a modified version of the problem introduced in [82]. The domain is given by a square with two holes

$$\Omega = [0,1] \times [0,1] \setminus (1/8, 3/8) \times (1/4, 3/4) \setminus (5/8, 7/8) \times (1/4, 3/4).$$

The domain describes a profile of a heat shield. An air flow runs through the holes and cools the block. The top, bottom, and right boundaries are insulated (Neumann 0) and the left boundary is exposed to a high temperature (Dirichlet). The parameters of the problem are the Biot numbers (thermal conductivities) on the left boundary Γ_{out} and on

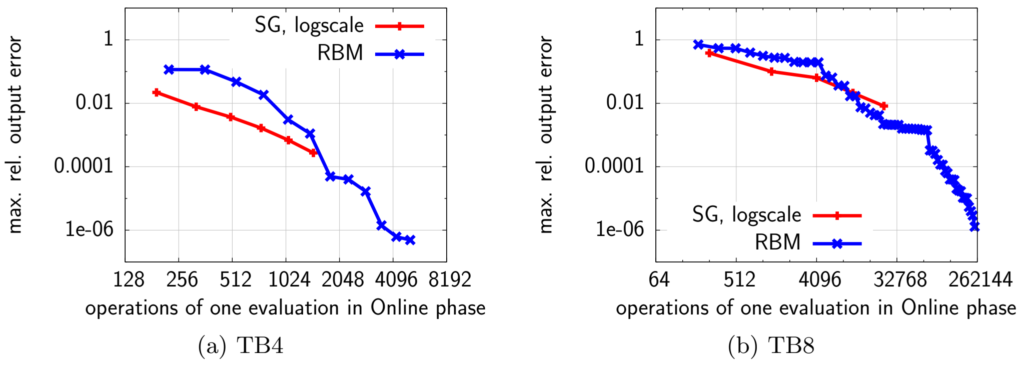


Figure 34: Comparison of the error of sparse grid interpolation and reduced basis approximation: For rough approximations the sparse grid interpolant performs slightly better than the reduced basis method.

the interior boundaries of the holes Γ_{in}. The heat field is defined by the (time-dependent) heat equation [120]. The output of interest is the average temperature of the block

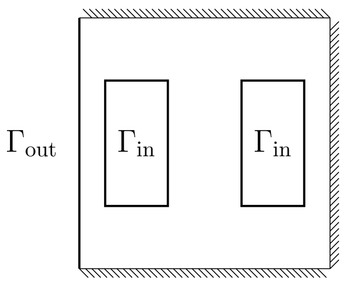
$$s^e(\boldsymbol{\mu}) = \int_\Omega u^e(\boldsymbol{x};\boldsymbol{\mu})\,\mathrm{d}\boldsymbol{x},$$

where $u(\boldsymbol{\mu})$ is the solution of the heat equation after sufficiently many time steps, see Fig. 35b for an example. The parameter domain is $\mathcal{D} = (0.01, 0.001) \times (0.5, 0.1)$.

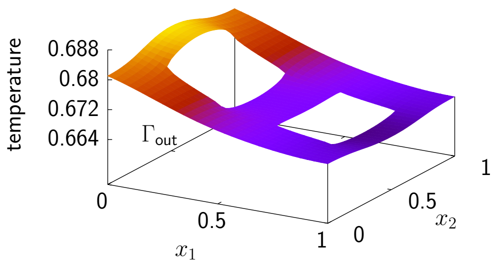
We have a time-dependent problem with two parameters. We can consider time just as any other parameter and perform the interpolation not on a two-dimensional sparse grid but on a three-dimensional one. Or we construct a two-dimensional interpolant for each time step. Both approaches are compared to the reduced basis method in Fig. 36. We used 1024 time steps with step size 1/64 and adaptive sparse grids with the usual surplus-based refinement criterion. Again we plot the error (73) versus the number of operations for one evaluation of the reduced output function in the Online phase. Whereas the three-dimensional sparse grid interpolant (time-space SG), which incorporates time and the two parameters at once, clearly performs worse than the reduced basis method, we obtain competitive results with the separate two-dimensional interpolants for each time step.

8.2. Regression and Pre-Computed Data Repositories

Now, instead of only interpolating the output of interest, we reconstruct the whole state vector $\boldsymbol{u}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$ for an arbitrary parameter $\boldsymbol{\mu} \in \mathcal{D}$. This is useful, for example, to visually explore simulation results. Sparse grid interpolation has already been studied in this context [47, 45, 44]. As mentioned above, a drawback of the interpolation approach is that the parameters corresponding to snapshots have to coincide with sparse grid points. The obvious solution is to employ regression instead of interpolation. Whereas



(a) Geometry of the heat shield



(b) Solution for $\boldsymbol{\mu} = (0.05, 0.005)$

Figure 35: The heat shield problem: The geometry is sketched in (a) and the surface of a sample solution is plotted in (b).

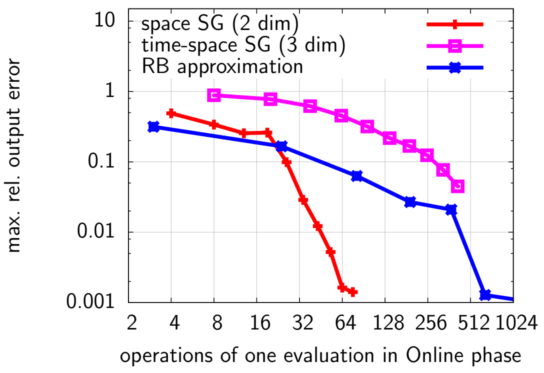


Figure 36: Comparison of sparse grid interpolation with reduced basis approximation for the heat shield problem.

we can simply replace sparse grid interpolation with sparse grid regression if we want to compute only the output of interest, this is not straightforward if we are interested in the whole state vector because it becomes computationally infeasible. That is why we combine POD and sparse grid regression [46].

In the following, we give details on this model reduction method based on sparse grid regression and POD, show that it indeed can handle pre-computed data repositories, and, finally, evaluate it on thermal conduction problems.

Pre-computed data repositories Most interpolation methods which are still applicable in higher-dimensional settings (e.g., kriging, splines) have to solve a system of linear equations to compute the weights for the basis or kernel functions [102]. In contrast, the construction of a sparse grid interpolant is linear in the number of sparse grid points, see Sec. 2.4. That is why it is possible to interpolate not only the output of interest but the whole state vector, i.e., we construct $\mathcal{N}$ sparse grid interpolants, one for each node of the state vector $\boldsymbol{u}(\boldsymbol{\mu})$. The drawback is that the parameter configurations (sampling points) have to coincide with the sparse grid points. This means that we have to know the sparse grid points at the time we compute the snapshots. However, a common scenario is that we already have many snapshots stored in a data repository and want to explore the corresponding simulation at arbitrary parameter configurations (visualization, computational steering). An obvious solution is to employ sparse grid regression instead of sparse grid interpolation. Regression can handle scattered data points without a pre-defined structure such as a sparse grid. Unfortunately, regression is distinctly more expensive than interpolation. We have to solve a system of linear equations with as many unknowns as we have sparse grid points, see Sec. 2.4.2. Thus, we cannot afford to compute a regression function for each node of the state $\boldsymbol{u}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}}$.

Regression of POD coefficients To reduce the number of required regression functions, we first compute a POD basis for our snapshots and then learn the coefficients with sparse grid regression. We do not use the POD basis for Galerkin projection because we want to have a non-intrusive method, see Sec. 2.3.2. Similar methods have already been introduced in [129, 16, 17] with either cubic splines in only one-dimensional settings or with kernel ridge regression.

Let $\boldsymbol{U} \in \mathbb{R}^{\mathcal{N} \times M}$ be the matrix of M snapshots $\boldsymbol{u}(\boldsymbol{\mu}_i)$ with $\mathcal{N}$ nodes each and with parameters $\mathcal{P} = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_M\}$. Let $\boldsymbol{V} = [\boldsymbol{v}_1, \dots, \boldsymbol{v}_n] \in \mathbb{R}^{\mathcal{N} \times n}$ be the first n POD basis vectors with $n \ll \mathcal{N}$, see Sec. 2.3.2. We can approximate a snapshot

$$\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}_j) \approx \sum_{i=1}^{n} \beta_{i,j} \boldsymbol{v}_i,$$

as a linear combination of the POD basis vectors $[\boldsymbol{v}_1, \dots, \boldsymbol{v}_n]$ and the coefficients $\boldsymbol{U}^T \boldsymbol{V} = [\boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_n] \in \mathbb{R}^{M \times n}$. Recall that $\beta_{i,j}$ is the j-th component of vector $\boldsymbol{\beta}_i$. In general, we can say a solution $\boldsymbol{u}(\boldsymbol{\mu})$ is approximated as

$$\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}) \approx \sum_{i=1}^{n} g_i(\boldsymbol{\mu}) \boldsymbol{v}_i,$$

where the functions $g_1, \dots, g_n : \mathcal{D} \to \mathbb{R}$ determine the coefficient of the respective basis vector $\boldsymbol{v}_i$ for a parameter $\boldsymbol{\mu} \in \mathcal{D}$. From the POD of the M snapshots, we can construct the training data sets

$$\mathcal{S}_i = \{(\boldsymbol{\mu}_j, \beta_{i,j})\}_{j=1}^{M},$$

for each function g_i and POD basis vector $\boldsymbol{v}_i$. We then approximate g_i with sparse grid regression as discussed in Sec. 2.4.2. Instead of $\mathcal{N}$, we only have to compute n regression functions. Of course, we assume the problem has a low-dimensional structure and thus $n \ll \mathcal{N}$, see Sec. 2.3.1.

In the Offline phase, we compute the POD of the snapshots $\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu}_M)$ and the sparse grid regression functions $g_1, \dots, g_n \in \mathcal{V}_\ell^{(1)}$ in a sparse grid space with N basis functions (sparse grid points). In the Online phase, we only have to evaluate the coefficient functions $g_1, \dots, g_n$ and form the linear combination

$$\boldsymbol{u}_n(\boldsymbol{\mu}) = \sum_{i=1}^{n} g_i(\boldsymbol{\mu}) \boldsymbol{v}_i, \tag{76}$$

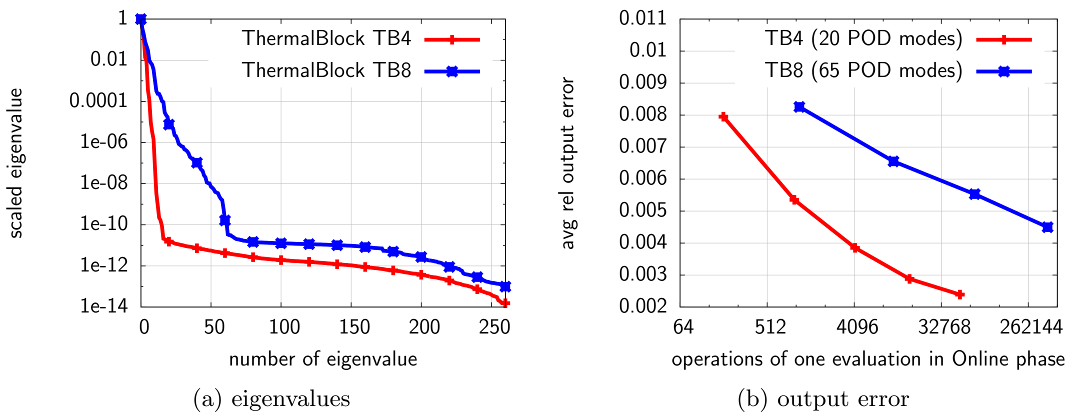


Figure 37: The eigengap in (a) indicates that 20 and 65 POD modes for the thermalblock with four and eight parameters, respectively, are a good choice. In (b) we see the average relative error of the output of interest. The accuracy is sufficient for applications such as visualization and computational steering.

to get an approximate of $\boldsymbol{u}^{\mathcal{N}}(\boldsymbol{\mu})$.

Steady-state heat conduction To evaluate the regression-based model order reduction method, we again consider the thermal block problem with four and eight parameters as introduced in the previous section. We compute randomly $2,769$ snapshots for the four-dimensional and $6,401$ snapshots for the eight-dimensional problem. Let us first consider the eigenvalues of the covariance matrix, see Fig. 37a. The clear bend in the curve at about 20 POD modes for the thermal block problem with four parameters and at about 65 POD modes for the problem with eight parameters indicates that we should use 20 and 65 modes, respectively. With these 20 and 65 POD modes, respectively, we construct the reduced-order model (76) with the procedure described in the previous paragraph. The corresponding error curves are shown in Fig. 37b. Note that we used the average instead of the maximum relative output error over a test set $\mathcal{T}$ because their are a few outliers which lead to high maximum errors even though most of the solutions are fine. The regularization parameter λ has been chosen with cross validation. For both problems, the method shows a reasonable error behavior. When we increase the number of sparse grid points, i.e., the number of operations needed for one evaluation in the Online phase, the error decreases. The obtained accuracy is competitive with sparse grid interpolation and sufficient for visualization and computational steering, see [46].

9. Remarks

Both, the model reduction method based on sparse grid interpolation of the output function s and the method based on sparse grid regression of the POD coefficient functions, are non-intrusive, cf. Sec. 2.3.2. Therefore, we can treat the solver for the full model as black box and do not have to rely on any additional knowledge of the governing equations.

With the direct interpolation of the output function, we avoided the detour around the state variable $u(\boldsymbol{\mu})$. We only have to evaluate one sparse grid function to compute the output of interest. Sparse grid interpolation is necessary because we usually have a multi-dimensional interpolation problem due to multiple parameters $\boldsymbol{\mu} = (\mu_1, \dots, \mu_d) \in \mathcal{D}$. We have seen that even though we have a non-intrusive method, the sparse grid interpolant of the output function s is competitive with the reduced basis approximation of s if only a rough response is required. This is the case for visualization, exploration, and computational steering.

With the regression of the POD coefficients, we can also handle pre-computed data where the parameter configurations do not necessarily coincide with the sparse grid points. Because sparse grid regression is distinctly more expensive than interpolation, we first transform the data into its POD representation and then learn the POD coefficient functions only. Note that such a transformation is not necessary, if interpolation can be employed because the construction of a sparse grid interpolant is only linear in the number of grid points.

Overall, the reduced-order models obtained with our non-intrusive sparse grid methods have an accuracy sufficient for visualizing and exploring simulations, and we do not have to know anything about the underlying governing equations or rewrite the full model solvers.

Besides these applications in the context of model reduction, we considered several classical problems of supervised learning in Sec. 7, and tackled them with sparse grid methods.

A fundamental building block for many learning methods is density estimation. We introduced a sparse grid density estimation method and developed standard algorithms to work with these sparse grid densities. The results have shown that the method is not very sensitive to the regularization parameter and that it is well suited for large data sets. We could easily tackle data sets with up to 500,000 data points in up to nine dimensions where common implementations of kernel density estimation already failed. We also introduced a sampling method especially adapted to sparse grid density functions. With it, we achieved better results than with a standard sampling method with respect to accuracy and runtime. Overall, we introduced a whole toolbox to work with sparse grid density functions.

We presented a probabilistic, generative model for classification based on sparse grid density estimation. Whereas the classification method based on sparse grid regression is a non-probabilistic approach, i.e., it directly constructs the map from the data points to the class labels, we learn $p(\boldsymbol{x}|Y_i)$ and compute $p(Y_i|\boldsymbol{x})$ with Bayes' theorem. Synthetic and real-world data sets have shown that our density-based approach is competitive with the classical, non-probabilistic approach with respect to the classification accuracy. Additionally, the system matrix of the system of linear equations underlying our density-based approach is independent from the training data points. This allows us to split the classification procedure into an Offline and an Online phase. With this splitting, we solve the classification problem in $\mathcal{O}(N^2)$ rather than in $\mathcal{O}(N^3)$. In our examples, we observed speed ups up to 500. Future work might include an extension of the density-based approach to regression [193] and an Offline/Online splitting of the computational procedure for adaptive sparse grids.

Due to their dynamic structure, adaptive sparse grids are not only problematic for the Offline/Online splitting of the density-based classification method but also for modern hardware [108]. We employed AdaBoost to train sparse grid classifiers on a data set with dynamic weights and put them into a team. With classifiers on regular sparse grids, the team achieved a similar classification accuracy as an adaptive sparse grid classifier. The advantage is that due to the regular sparse grids, the complexity of the implementation is dramatically reduced and it can be fitted easier to modern hardware leading to a shorter runtime of the classification process by up to a factor of 2.5.

Part III.
MOR III: Post Analysis Model Order Reduction

So far we discussed *a priori* and *a posteriori* model order reduction. In the former, no data is given and the reduced model is derived from knowledge obtained by theoretic considerations alone. In the latter, the problem has already been solved for a set of parameters in the past and the resulting pile of data is used to build a reduced-order model. We go one step further now: We analyze the data with machine learning methods first and then construct reduced-order models from the pre-processed data.

The general idea of model reduction is to approximate the solution sub-manifold $\{u^{\mathcal{N}}(\boldsymbol{\mu}) \,|\, \boldsymbol{\mu} \in \mathcal{D}\}$ with a low-dimensional subspace $\mathcal{U}_n$ of the high-dimensional space $\mathcal{U}^{\mathcal{N}}$. We have seen many examples where the fundamental assumption that the solution manifold can be well approximated by the space $\mathcal{U}_n$ with only a few dimensions holds. However, we can also think of as many examples where the solutions corresponding to the parameters in $\mathcal{D}$ are scattered all over $\mathcal{U}^{\mathcal{N}}$ and model order reduction fails because the solution manifold has the same dimension as the space $\mathcal{U}^{\mathcal{N}}$.

Besides these two very extreme cases where model reduction works very well and where it fails completely, a third alternative is that the solutions for the parameters in $\mathcal{D}$ are indeed distributed all over $\mathcal{U}^{\mathcal{N}}$ but they form locally low-dimensional clusters. Thus, if we would try to approximate all of these clusters at once, we would need a reduced space spanned by many basis vectors even though each cluster considered individually lies in a low-dimensional space. One example are the trajectories of a time-dependent problem which pass through many regions of the state space $\mathcal{U}^{\mathcal{N}}$ but in each region they form a low-dimensional manifold. Other examples are problems where the parameters determine the characteristics of the underlying system. For instance, the activation energy parameter in a chemical reaction simulation decides if the reaction occurs or not. Depending on that, the solutions certainly occupy different parts of the space $\mathcal{U}^{\mathcal{N}}$ even though the solutions corresponding to one specific characteristic behavior of the system might be approximated well in a low-dimensional subspace. Hence, in the following, to efficiently treat such problems, we first analyze the data to find out about such local clusters and then incorporate this information into the reduced-order model. We call this *post analysis* model order reduction.

A first step towards this direction is adaptivity. We already have extensively used this in the form of adaptive sparse grids with the surplus-based refinement criterion in the previous sections. In the context of model reduction and the reduced basis method we refer to, e.g., [97, 149, 36] for adaptive and goal-oriented greedy algorithms, and to [98] for a method to adaptively select the dimension n of the reduced basis space in the Online phase. In [131] many basis vectors are pre-computed but only a few of those are selected online. Another approach to exploit the locality of the problem is presented in [57, 43]. There, a POD basis is built for each cell of the Voronoi tessellation of the set of snapshots. These bases are then combined in one global basis to construct the reduced-order model.

This can be further improved by building not one global but multiple local reduced-order models and then choosing a suitable one during the Online phase to perform the actual approximation. Hence, besides a way to construct the local reduced-order models, we need a strategy to decide which of the local models to use. An easy but very efficient solution is to decide with respect to the parameter. In [96, 56] the parameter- and time-domain are split recursively into subdomains and for each subdomain a separate reduced-order model is built. In the Online phase, when a new parameter is given, the corresponding subdomain of the parameter- and time-domain is determined, and the associated reduced-order model is employed for the approximation. A similar approach is followed in [61] for elliptic and in [60] for parabolic PDEs. All these methods have in common, that they recursively split the parameter domain and that their selection procedure relies on the parameter. The recursive splitting of the domain might lead to a large number of subdomains because a poor division in the beginning cannot be corrected later on. Furthermore, the parameter is not always a good indicator for the selection of the local model. This is especially the case for time-dependent problems and for nonlinear problems during Newton iterations.

In [13] the local reduced-order models are derived from clusters of the set of snapshots obtained with k-means and a local model is selected with respect to the current state vector in the Online phase. Thus, a recursive splitting of the parameter domain is avoided and the selection procedure does not rely on the parameter anymore. However, in particular if the number of clusters is large, clustering methods do not always produce a reasonable and stable cluster assignment compared to, e.g., simply splitting the parameter- and time-domain. This issue is not addressed in [13] because only up to four clusters are used. Furthermore, the procedure in [13] to select a local reduced-order model scales quadratically with the number of clusters. Note that this might be reduced to a log-linear growth but no details are given in [13]. In [183], the approach in [13] is extended to allow (costly) updates of the local reduced-order models during the Online phase. With a somewhat different motivation but also with the idea of local reduced-order models in mind, interpolated reduced bases are considered in [11] and interpolated reduced-order models in [128, 10, 54, 141, 12].

Recall that *a priori* model order reduction does not involve learning from data and that *a posteriori* model order reduction can be related to supervised learning. Most of the *post analysis* methods group similar snapshots together and derive local reduced-order models from these groups. This is an unsupervised learning problem. We have data points and want to cluster them with respect to a certain clustering criterion. That is why we introduce two non-convex clustering methods based on sparse grids in the following two sections — the first clustering methods based on sparse grids to the author's knowledge. We start with an out-of-sample extension for spectral clustering in Sec. 10.1 and continue with a density-based clustering method in Sec. 10.2. We discuss the properties of these two methods and assess both on benchmark and real-world data mining problems.

Having introduced these two clustering methods, we propose two localization approaches in the context of model order reduction. In Sec. 11, a workflow is introduced to analyze car crash simulation data. Machine learning techniques are employed to quickly detect simulation runs with exceptional or extreme behavior. For that purpose, we cluster the nodes of the finite element model of the car and use these clusters to spot regions where the car behaves unusual during the crash. Furthermore, the clusters are the input for (nonlinear) dimensionality reduction methods to find a low-dimensional embedding of the data which still reflects the characteristics of the simulation runs. In Sec. 12 we consider nonlinear PDEs and extend the discrete empirical interpolation method (DEIM) to the localized discrete empirical interpolation method (LDEIM). We cluster the snapshots corresponding to the nonlinear term to derive local DEIM interpolants. We employ feature extraction on the state vector to decide which local interpolant to use during the Online phase. To demonstrate LDEIM, we consider a reacting flow problem.

10. Unsupervised Learning with Sparse Grids

In this section, we show how to tackle clustering and other unsupervised learning problems with sparse grid techniques. We present two fundamentally different clustering methods and demonstrate them on benchmark and real-world problems. In Sec. 10.1 we start with an out-of-sample extension for spectral clustering. Since the computational complexity of spectral clustering is in $\mathcal{O}(M^3)$ where M is the number of data points, such an extension becomes necessary very soon. We consider image segmentation where out-of-sample extensions are inevitable. We continue in Sec. 10.2 with a density-based clustering method. It is especially well suited for large data sets and does not only give a cluster assignment of the data but also distinguishes between dense (strong) and sparse (weak) clusters. We discuss this feature and employ it to detect dominating modes in the crash behavior of cars.

It is important to emphasize that the two clustering methods are not meant to compete with each other. On the contrary, they have very different properties which, combined, give a hybrid clustering method as we show in the real-world examples in Sec. 11.

10.1. Sparse-Grid-Based Out-of-Sample Extension for Spectral Clustering

Spectral methods such as Laplacian eigenmaps are among the most popular algorithms for clustering and dimensionality reduction [24, 125, 179]. However, since the runtime of these methods usually scales with $\mathcal{O}(M^3)$ in the number of data points M and they only provide a clustering or an embedding of the training data, they become computationally infeasible for large data sets very soon. Therefore, to deal with data points not included in the training set, so-called out-of-sample extensions are necessary. In this section, we propose to use sparse grid functions to approximate the eigenfunctions of the Laplace-Beltrami operator [147]. This provides us with an explicit mapping between the high-dimensional data points and their low-dimensional representation which is required to cluster out-of-sample points.

In Sec. 10.1.1 we briefly present spectral clustering based on Laplacian eigenmaps as introduced in [24] before we discuss the sparse-grid-based out-of-sample extension in Sec. 10.1.2. Benchmark problems follow in Sec. 10.1.3 and image segmentation in Sec. 10.1.4.

10.1.1. Laplacian Eigenmaps and Spectral Clustering

Spectral methods represent the data as a similarity graph and use the eigenvectors of a modified adjacency matrix of the graph, the so-called graph Laplacian, to derive a clustering or low-dimensional embedding. They can be seen as relaxed graph partition algorithms with the objective to minimize the so-called cut (or "flow") between the different partitions. Spectral methods have been shown to perform well where other clustering algorithms such as k-means fail [179].

Graph Laplacian Let $\mathcal{S} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_M\} \subset \mathbb{R}^d$ be our data set with M data points. We construct a weighted graph $G = (\mathcal{S}, \mathcal{E})$ where the weight $W_{ij} \geq 0$ describes the similarity between the points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. In many cases, the well-known Gaussian kernel

$$W_{ij} = \exp\left(-\frac{\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}{\sigma^2}\right) \tag{77}$$

is used. The matrix $\boldsymbol{W}$ with the weights (77) is the weighted adjacency matrix of the graph. We define the degree d_i of the i-th data point or vertex as $d_i = \sum_j W_{ij}$ and the degree matrix $\boldsymbol{D}$ as diagonal matrix with $d_1, \dots, d_M$ on the diagonal. Since fully connected graphs become too expensive, other graph types such as ϵ-neighborhood or nearest neighbor graphs are used. Note, however, that we always assume that the graph is connected. In the following, the so-called (unnormalized) graph Laplacian

$$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W} \tag{78}$$

plays a crucial role. It is a discrete approximation of the Laplace-Beltrami operator corresponding to the manifold described by the data $\mathcal{S}$, see [24] and the references therein. Extensive studies about the operator $\boldsymbol{L}$ can be found in, e.g., [53, 179]. For us it is only important that the smallest eigenvalue of $\boldsymbol{L}$ is 0 and that the corresponding eigenvector is the constant vector $\mathbf{1}$.

Bipartitioning of graphs The intuition of clustering is that it groups data points into different clusters according to their similarities. If we are given the data in form of a weighted graph where the weight of an edge between two vertices corresponds to their similarity, we have to partition the graph such that the edges connecting the partitions have a very small weight. Thus, we partition the set of vertices $\mathcal{S}$ of our similarity graph $G = (\mathcal{S}, \mathcal{E})$ into two sets $\mathcal{A}, \mathcal{B} \subset \mathcal{S}$ with $\mathcal{A} \cup \mathcal{B} = \mathcal{S}$ and $\mathcal{A} \cap \mathcal{B} = \emptyset$ where the sum of weights of the edges connecting the vertices in $\mathcal{A}$ to $\mathcal{B}$ should be small. More formally, this means we minimize the cut

$$\operatorname{cut}(\mathcal{A}, \mathcal{B}) = \sum_{\boldsymbol{x}_i \in \mathcal{A}, \boldsymbol{x}_j \in \mathcal{B}} W_{ij}. \tag{79}$$

Directly minimizing (79) would lead to partitions which just cut off outliers. To avoid that, we consider the Normalized Cut between $\mathcal{A}$ and $\mathcal{B}$,

$$\operatorname{Ncut}(\mathcal{A}, \mathcal{B}) = \operatorname{cut}(\mathcal{A}, \mathcal{B}) \left(\frac{1}{\operatorname{vol}(\mathcal{A})} + \frac{1}{\operatorname{vol}(\mathcal{B})}\right), \tag{80}$$

where $\operatorname{vol}(\mathcal{A}) = \sum_{\boldsymbol{x}_i \in \mathcal{A}} d_i$ is the so-called volume of the set $\mathcal{A}$.

The minimization of Ncut is a well-studied problem. The result is a vector $\boldsymbol{y} \in \{0, 1\}^M$ where each component indicates if the corresponding vertex (data point) either belongs to subset $\mathcal{A}$ or $\mathcal{B}$. Unfortunately, the minimization of the Ncut is NP-complete [181]. However, the minimum of (80) in its relaxed form with $\boldsymbol{y} \in \mathbb{R}^M$ can be found in polynomial time by minimizing the Rayleigh quotient

$$\frac{\boldsymbol{y}^T \boldsymbol{L} \boldsymbol{y}}{\boldsymbol{y}^T \boldsymbol{D} \boldsymbol{y}}, \tag{81}$$

where the matrix $\boldsymbol{L}$ is the graph Laplacian and $\boldsymbol{D}$ the degree matrix of the underlying graph. We add the constraint $\boldsymbol{y}^T\boldsymbol{D}\mathbf{1} = 0$ to exclude the constant vector $\mathbf{1}$ with eigenvalue 0 as solution. Therefore, we have to solve the optimization problem

$$\underset{\substack{\boldsymbol{y} \\ \boldsymbol{y}\boldsymbol{D}\mathbf{1}=0}}{\arg\min} \frac{\boldsymbol{y}^T\boldsymbol{L}\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y}}. \tag{82}$$

If we assume an ascending order of the eigenvalues, the solution to (82) is the eigenvector to the second eigenvalue of the generalized eigenvalue problem

$$\boldsymbol{L}\boldsymbol{y} = \gamma\boldsymbol{D}\boldsymbol{y}\,. \tag{83}$$

Because the eigenvectors computed in (83) are the eigenfunctions of the corresponding Laplace-Beltrami operator evaluated at the data points in $\mathcal{S}$, this procedure is also called Laplacian eigenmaps. To finally achieve a partitioning of the graph, we apply k-means on the components of $\boldsymbol{y}$ and so derive a partitioning of the vertices of the graph into the subsets $\mathcal{A}$ and $\mathcal{B}$.

Clustering and dimensionality reduction It seems strange to solve the relaxed minimization problem (82) only to apply k-means eventually. However, due to the properties of the graph Laplacian, the components $\{y_1, \ldots, y_M\}$ of $\boldsymbol{y}$ exhibit a clearer cluster structure than the original data points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ [24]. For example, by first computing the embedding $\{y_1, \ldots, y_M\}$ of the data points $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ of the two moons data set and then applying k-means, we can separate the two moons whereas this is not possible if k-means is applied directly to $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$, see Sections 2.2.2 and 10.1.3.

We can also consider the components of $\boldsymbol{y}$ as a low-dimensional embedding of the points in $\mathcal{S}$. This embedding preserves locality with respect to the similarity measure defined by the weights W_{ij} in the graph. Thus, if data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are similar, i.e., if the weight of the edge connecting these two points is high, then their respective embeddings y_i and y_j should be close together as well. From this point of view, we compute a low-dimensional representation of the data points in $\mathcal{S}$. Instead of only the eigenvector $\boldsymbol{y}_2$ corresponding to the second smallest eigenvalue of (83), we can also compute the eigenvectors $\boldsymbol{y}_2, \ldots, \boldsymbol{y}_{r+1}$ corresponding to the 2nd, $\ldots, r+1$-th eigenvalue, respectively. Then, the i-th row of the matrix $[\boldsymbol{y}_2, \ldots, \boldsymbol{y}_{r+1}] \in \mathbb{R}^{M \times r}$ is an r-dimensional representation of the data point $\boldsymbol{x}_i$.

Computational procedure and costs Spectral clustering with Laplacian eigenmaps requires us to build a similarity graph of the data and then solve the generalized eigenproblem (83). Whereas constructing a similarity graph is a standard task in data mining and easily parallelizable, the eigenproblem poses the computationally most expensive part of the method. It scales in $\mathcal{O}(M^3)$. Therefore, large amounts of data cannot be processed with spectral clustering without further ado.

Out-of-sample extensions We have already seen that the complexity to solve the eigenproblem (83) is in $\mathcal{O}(M^3)$ where M is the number of data points. In case of spectral clustering this is particularly unfortunate because there is no natural way to treat out-of-sample points. This means, we cannot assign data points to clusters if they were not included in the graph Laplacian $\boldsymbol{L}$ during the computation of the eigenvectors. However, with so-called out-of-sample extensions, we can learn an explicit map $g : \mathbb{R}^d \to \mathbb{R}^r$ between data points in $\mathbb{R}^d$ and their low-dimensional embedding in $\mathbb{R}^r$. We can process large amounts of data by first partitioning the data into a small training data set and a large test set and computing the Laplacian eigenmaps with the out-of-sample extension on the training set only. Then, we evaluate g at all points in the test data set. If an evaluation of g is cheap, this is computationally feasible for large amounts of data.

In [103, 111] extensions based on linear projections are introduced. However, linear projection leads to poor results for already simple problems such as the Swiss roll example, see Sec. 10.1.3. An approach based on polynomials is presented in [157]. This approach achieves better results than the one based on the linear projection but it is also distinctly more expensive. If the degree of the polynomials is increased to achieve a better accuracy, the computational costs become quickly infeasible.

Nyström-based out-of-sample extension The most common out-of-sample extension for Laplacian eigenmaps and other spectral methods is based on the Nyström method [140, 21]. In the context of machine learning, the Nyström method first appeared in [186]. It has been proposed as an out-of-sample extension for spectral methods in [25]. Since then it has been applied to a wide range of problems, see, e.g., [72, 187, 8]. It is based on the assumption that the similarity measure can be represent as a kernel function K. These kernel functions are then used to build the Gram matrix for the given training data. Let $\boldsymbol{y}_1, \dots, \boldsymbol{y}_M$ and $\gamma_1, \dots, \gamma_M$ be the eigenvectors and eigenvalues, respectively, of the Gram matrix. Then the Nyström method uses

$$g_i(\boldsymbol{x}) = \frac{\sqrt{M}}{\gamma_i} \sum_{j=1}^{M} y_{j,i} K(\boldsymbol{x}, \boldsymbol{x}_j)\,, \tag{84}$$

as approximation for the i-th eigenfunction at point $\boldsymbol{x}$. Here $y_{j,i}$ is the i-th component of the j-th eigenvector $\boldsymbol{y}_j$. It has been shown in [25] that this approximation is in essence the kernel PCA projection [166]. We clearly see that the sum in (84) iterates over all data points in $\mathcal{S}$. Therefore, we have to run over all data points to obtain the embedding for an out-of-sample point.

10.1.2. Sparse-Grid-Based Out-of-Sample Extension

In this section we show how to approximate the eigenfunctions of the Laplace-Beltrami operator with sparse grid functions in $\mathcal{V}_\ell^{(1)}$. The approximations are computed from the graph Laplacian $\boldsymbol{L}$. We derive the corresponding generalized eigenvalue problem and conclude with remarks on the computational procedure.

Approximation of eigenfunctions on sparse grids Let $\mathcal{S}$ be our data set and $f \in \mathcal{V}_\ell^{(1)}$ be a sparse grid function. We want $f(\boldsymbol{x}) = y$ to be the Laplacian eigenmaps embedding of the data point $\boldsymbol{x}$. The function f is a linear combination of the hierarchical basis in Φ_ℓ with the coefficients $\boldsymbol{\alpha} = [\alpha_1, \dots, \alpha_N]^T$

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \alpha_i \phi_i(\boldsymbol{x}). \tag{85}$$

We define the vector $\boldsymbol{f} = [f(\boldsymbol{x}_1), \dots, f(\boldsymbol{x}_M)]^T \in \mathbb{R}^M$ with the function value $f(\boldsymbol{x}_i)$ at the i-th component. To find the coefficients of (85), we solve the minimization problem

$$\underset{\substack{f \in \mathcal{V}_\ell^{(1)} \\ \boldsymbol{f}^T \boldsymbol{D}\boldsymbol{1} = 0}}{\arg\min} \frac{\boldsymbol{f}^T \boldsymbol{L} \boldsymbol{f}}{\boldsymbol{f}^T \boldsymbol{D} \boldsymbol{f}}, \tag{86}$$

where $\boldsymbol{L}$ and $\boldsymbol{D}$ are the graph Laplacian and degree matrix of our similarity graph, respectively. With the matrix $\boldsymbol{B}$ of Sec. 2.4.2 with entries $B_{ij} = \phi_j(\boldsymbol{x}_i)$ we have $\boldsymbol{f} = \boldsymbol{B}\boldsymbol{\alpha}$ and can rewrite (86) as

$$\underset{\substack{f \in \mathcal{V}_\ell^{(1)} \\ \boldsymbol{\alpha}^T \boldsymbol{B}^T \boldsymbol{D}\boldsymbol{1} = 0}}{\arg\min} \frac{\boldsymbol{\alpha}^T \boldsymbol{B}^T \boldsymbol{L} \boldsymbol{B} \boldsymbol{\alpha}}{\boldsymbol{\alpha}^T \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \boldsymbol{\alpha}}. \tag{87}$$

The minimum of (87) is the function $f \in \mathcal{V}_\ell^{(1)}$ with the coefficient vector $\boldsymbol{\alpha}$ which is the eigenvector of the second smallest eigenvalue of the generalized eigenproblem

$$\boldsymbol{B}^T \boldsymbol{L} \boldsymbol{B} \boldsymbol{\alpha} = \gamma \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \boldsymbol{\alpha}. \tag{88}$$

Condition of generalized eigenproblem The generalized eigenproblem (88) is called singular, if

$$\det(\boldsymbol{B}^T \boldsymbol{L} \boldsymbol{B} - \gamma \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B}) = 0$$

for all values of γ [20]. If the matrix $\boldsymbol{B}$ is singular, the eigenproblem (88) is singular as well. This is indeed the case because with $\det(\boldsymbol{B}) = 0$ and the multiplicativity of the determinant we obtain

$$\det\left(\boldsymbol{B}^T(\boldsymbol{L} - \gamma \boldsymbol{D})\boldsymbol{B}\right) = \det\left(\boldsymbol{B}^T\right) \det\left(\boldsymbol{L} - \gamma \boldsymbol{D}\right) \det\left(\boldsymbol{B}\right) = 0.$$

However, we really have to gear the data set and the sparse grid to get a singular matrix $\boldsymbol{B}$. For example, if there exists a basis function ϕ_i which evaluates to zero at all data points, then column i of $\boldsymbol{B}$ contains only zero entries. The eigenproblem solver available in the GNU Scientific Library does not directly solve (88) but the problem

$$\eta \boldsymbol{B}^T \boldsymbol{L} \boldsymbol{B} \boldsymbol{\alpha} = \nu \boldsymbol{B}^T \boldsymbol{D} \boldsymbol{B} \boldsymbol{\alpha} \tag{89}$$

where $\gamma = \nu / \eta$. A singular problem is signaled with $\eta = \nu = 0$ which allows us to catch these cases [14].

Even though we can avoid singular eigenproblems, the condition of the problem might still be bad. Therefore, and to impose a certain smoothness constraint on the solution function, we add a regularization operator $\boldsymbol{C}$ to the minimization problem (87) and get

$$\underset{\substack{f\in\mathcal{V}_\ell^{(1)}\\ \boldsymbol{\alpha}^T\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{1}=0}}{\arg\min}\ \frac{\boldsymbol{\alpha}^T(\lambda\boldsymbol{C}+\boldsymbol{B}^T\boldsymbol{L}\boldsymbol{B})\boldsymbol{\alpha}}{\boldsymbol{\alpha}^T\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}\boldsymbol{\alpha}} \tag{90}$$

and the corresponding eigenproblem

$$\left(\lambda\boldsymbol{C}+\boldsymbol{B}^T\boldsymbol{L}\boldsymbol{B}\right)\boldsymbol{\alpha}=\gamma\boldsymbol{B}^T\boldsymbol{D}\boldsymbol{B}. \tag{91}$$

The balance between fidelity and smoothness is controlled with the regularization parameter λ. In the following, the matrix $\boldsymbol{C}$ has either the entries $C_{ij}=(\nabla\phi_i,\nabla\phi_j)_{L^2}$ or is the identity matrix. We refer to Sec. 2.4.2 for more details and a discussion why these are good choices for $\boldsymbol{C}$.

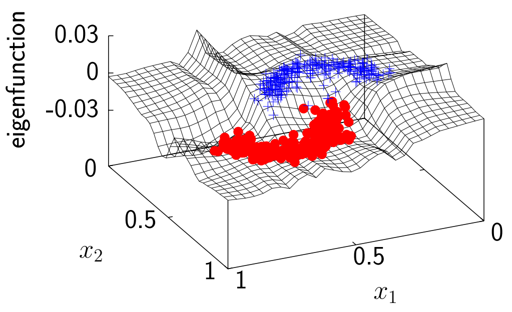
Savings of computational costs Let us first consider the eigenproblem. In the previous section, we have seen that the computational costs of spectral clustering with Laplacian eigenmaps scales in $\mathcal{O}(M^3)$ because we have to solve an eigenproblem of size $M\times M$. If we consider the eigenproblem (91) of spectral clustering with the sparse-grid-based out-of-sample extension, we see that it is of size $N\times N$ only where N is the number of grid points. Thus, the dimension of the eigenproblem is independent from the number of data points M. This leads to huge savings for large data sets where $M\gg N$.

The same holds for the out-of-sample extension. When we compute the cluster assignment or the low-dimensional embedding of a data point, we have to evaluate an approximation of the eigenfunction. Whereas in the case of the Nyström method (84), one such function evaluation scales with the number M of data points in the training data set $\mathcal{S}$, an evaluation of a sparse grid function in $\mathcal{V}_\ell^{(1)}$ is in $\mathcal{O}(\ell^d)$, see Sec. 2.4. Since we are interested in large data sets with $M\gg N$ and because due to the sparse grid structure $\ell^d<N$ holds, we can cluster an out-of-sample point with the sparse-grid-based extension faster than with the Nyström method.

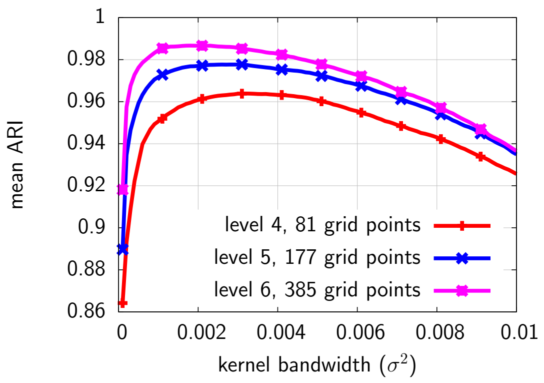
10.1.3. Benchmarks and other Learning Problems

We demonstrate our sparse-grid-based out-of-sample extension on some benchmark examples. The Laplacian eigenmaps embedding of the data points can be used to compute a cluster assignment or it can be directly seen as a low-dimensional representation of the data. We first consider clustering where we approximate the eigenfunctions with the help of a small training data set and cluster the test or out-of-sample data points with our out-of-sample extension. We then show that the sparse-grid-based out-of-sample extension also works for nonlinear dimensionality reduction. With a low-dimensional embedding we analyze the data of a computational fluid dynamics (CFD) simulation.

In the following, we stick to nearest neighbor graphs and employ our own implementation based on the Boost graph library. The generalized eigenproblem is solved with the GNU Scientific Library.



(a) second eigenfunction



(b) mean ARI

Figure 38: The surface of the sparse grid approximation (level five) of the second eigenfunction for the two moons data set is shown in (a). The values of the sparse grid function at the data points clearly reflect the two clusters. In (b) the mean ARI is plotted versus the σ^2 of the Gaussian kernel. The eigenfunction was approximated on a two-dimensional sparse grids of level four, five, and six, respectively.

Two moons data set We generate five training and test sets of the two moons data set with 500 and 5,000 points each, respectively. We compute an approximation of the second eigenfunction with the minimization problem (90) for each of the five training sets and test each on the five test sets. We employ the regularization term $\boldsymbol{C}$ with $C_{ij} = (\nabla\phi_i, \nabla\phi_j)_{L_2}$ corresponding to the Laplacian and set the regularization parameter λ to 0.01. The second eigenfunction is plotted in Fig. 38a. The mean over all ARI values is shown in Fig. 38b. Recall that an ARI of 1 means perfect clustering. Depending on the kernel bandwidth of the Gaussian kernel, we obtain an ARI above 0.96 with only 81 sparse grid points. If we increase the number of sparse grid points, we obtain even better results.

Iris data set Let us consider the four-dimensional iris flower data set. We only use the 100 data points corresponding to the non-linearly separable classes. Because we have very few data points we apply ten-fold cross validation. Therefore, the data set is split into ten randomly chosen subsets with ten points each. We then take the union of nine subsets as training set and the remaining subset as test set. This is repeated ten times with each of the subsets used exactly once as test set. In Fig. 39, the mean ARI is reported for our out-of-sample extension with a sparse grid of level four and the Nyström method. The regularization term $\boldsymbol{C}$ corresponding to the Laplacian with $\lambda = 0.0001$ is used. Both methods reach about the same mean ARI for the training set. However, this is not true for the test data set. Whereas the Nyström method reaches only a mean ARI of 0.38, our method performs better and yields 0.45.

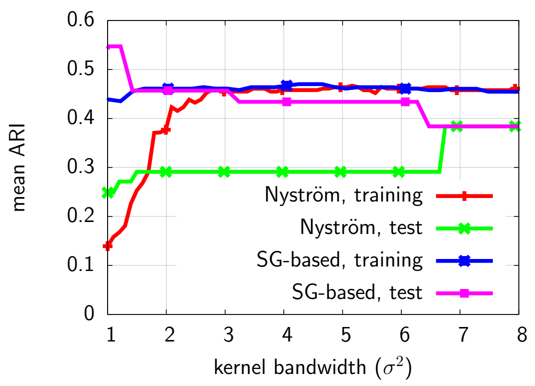
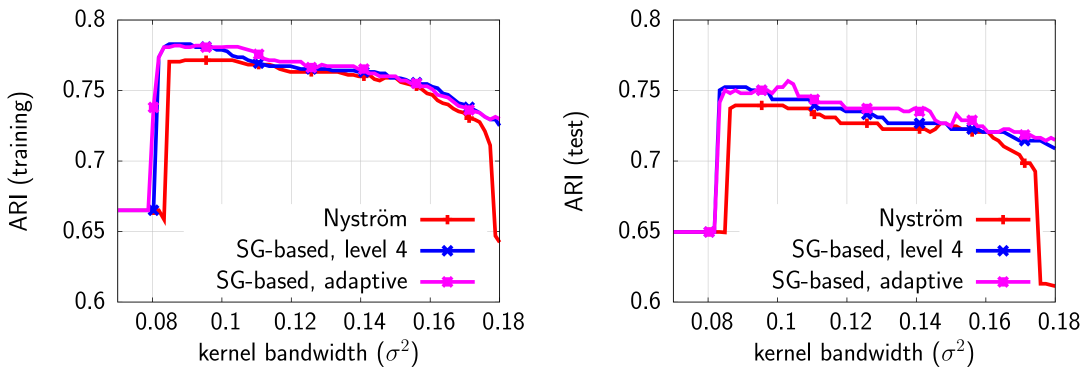


Figure 39: The mean ARI for the iris flower data set is plotted. Our sparse-grid-based out-of-sample extension performs better on the test data than the Nyström method.

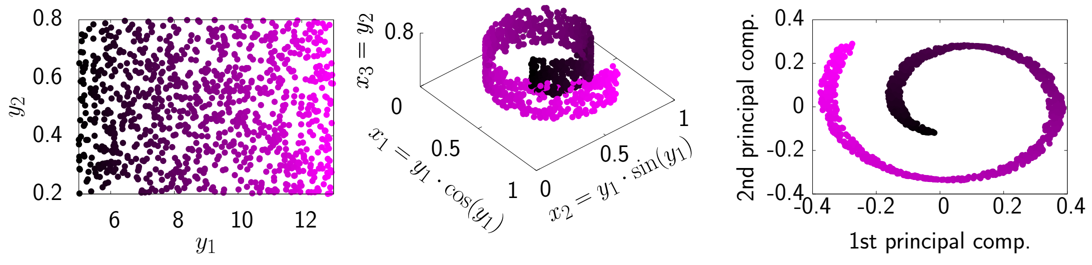


(a) training data set

(b) test data set

Figure 40: The mean ARI for the oil flow data set is shown to compare the Nyström method and our sparse-grid-based extension. Our out-of-sample extension is slightly better than the Nyström method.

Oil flow data set The 12-dimensional oil flow data set consists of three classes of fluids occurring in oil pipelines. We skip the data points corresponding to the "stratified configuration" because they are scattered into several smaller clusters. We cluster the two remaining configurations. Overall, we have a training data set of 1,318 points and a test data set with 687 points. We employ the nonuniform basis functions to resolve the boundary and to be able to cope with the 12 dimensions. For a regular sparse grid of level four with $N = 3,249$ grid points we perform better than the Nyström method, see Fig. 40. We achieve about the same accuracy with adaptive sparse grids but with distinctly fewer grid points. We start with a grid of level two and perform three refinement steps. In each of them, we refine ten percent of the grid points which are selected with respect to the common surplus-based adaptivity criterion, see Sec. 2.4. In the end, we have about $1,200$ grid points for reasonable kernel bandwidths $\sigma^2 \in [0.08, 0.1]$ and are still better than the Nyström method.



(a) original data (b) Swiss roll data set (c) embedding with PCA

Figure 41: A popular task of nonlinear dimensionality reduction is to recover the two-dimensional original data (a) from the three-dimensional Swiss roll data set (b) because common linear dimensionality reduction methods such as PCA fail (c).

Dimensionality reduction of the Swiss roll Let us now consider nonlinear dimensionality reduction with the Laplacian eigenmaps and our sparse-grid-based out-of-sample extension. The famous Swiss roll data set lies on a two-dimensional sub-manifold in $\mathbb{R}^3$. Two thousand data points $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M\} \subset [5, 13] \times [0.2, 0.8]$ (see Fig. 41a) are transformed into a three-dimensional data set $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\} \subset \mathbb{R}^3$ (see Fig. 41b) with the function

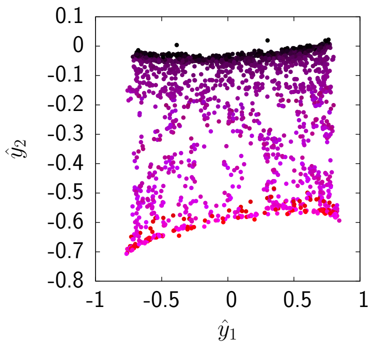
$$f(\boldsymbol{y}) = \begin{bmatrix} y_1 \cdot \cos(y_1) \\ y_1 \cdot \sin(y_1) \\ y_2 \end{bmatrix} .$$

The goal is to recover the two-dimensional data points from the three-dimensional data, i.e., we want to unroll the Swiss roll. In other words, we approximate the inverse function f^{-1} of f. It is well known and becomes immediately clear from the function f that this is not possible with common linear dimensionality reduction methods such as PCA, see Fig. 41c.

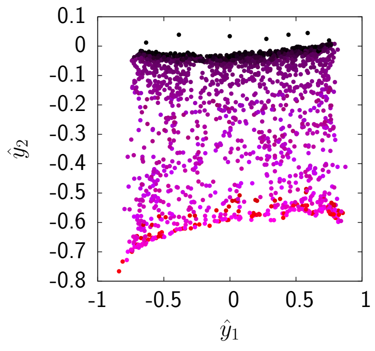
We compute the second and third eigenvector of the eigenproblem (91) where the underlying similarity graph is constructed from the Swiss roll data set with 2,000 data points. The eigenvectors are the sparse grid coefficients determining the three-dimensional sparse grid functions $\hat{f}_1^{-1}, \hat{f}_2^{-1} : \mathbb{R}^3 \to \mathbb{R}$. These lead to the function

$$\hat{f}^{-1}(\boldsymbol{x}) = \begin{bmatrix} \hat{f}_1^{-1}(\boldsymbol{x}) \\ \hat{f}_2^{-1}(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \end{bmatrix} .$$

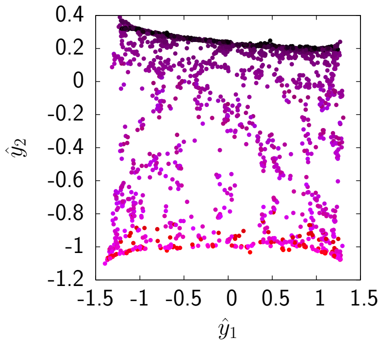
Thus, the two-dimensional embedding of a point $\boldsymbol{x} \in \mathcal{S}$ is $\hat{\boldsymbol{y}} = \hat{f}^{-1}(\boldsymbol{x})$. We plot the embedding of the training and the test data set of the Swiss roll in Fig. 42. We set the bandwidth of the Gaussian kernel to $\sigma^2 = 6$ and the regularization parameter λ to 0.01 for the sparse grid of level five and to 0.036 for the grid of level six. These parameters were hand picked. The Laplacian eigenmaps method with our out-of-sample extension is able to find a valid embedding of the data. We already obtain good results on sparse grids of level five with 705 grid points only.



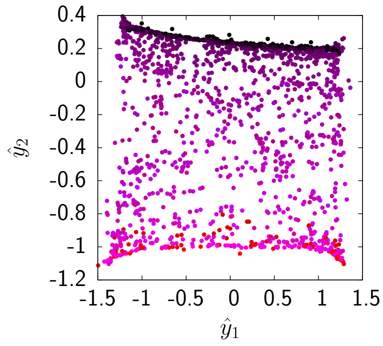
(a) level five, train



(b) level five, test



(c) level six, train



(d) level six, test

Figure 42: Embedding of three-dimensional Swiss roll data set into two dimensions: Eigenfunctions are approximated on a sparse grid of level five (top) and six (bottom), respectively. Whereas PCA fails for this data set, the Laplacian eigenmaps method with our out-of-sample extension is able to unroll the Swiss roll.

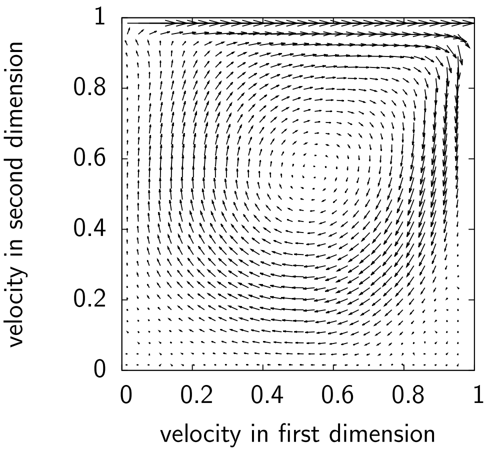


Figure 43: The Navier-Stokes equations are solved for the driven cavity problem. An example flow field is shown. Note that the length of the arrows has been multiplied by five to make them better visible.

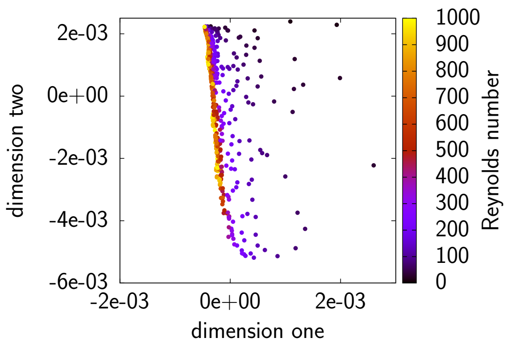
Analysis of flow simulation data With the Swiss roll example we have seen that the Laplacian eigenmaps can indeed detect the structure of the data and that our out-of-sample extension is able to give us a similar embedding for test data. We now apply the method to simulation data to gain insight into the behavior and the structure of the simulation.

We consider a fluid with a certain Reynolds number Re in a cavity where the lid moves with velocity a [89]. This movement of the lid sets the whole fluid in motion. This can be simulated with the Navier-Stokes equations. We discretize them on a 64×64 (full) grid and solve them with the method described in [89]. The flow field for $a = 0.88$ and $Re = 912$ is shown in Fig. 43.

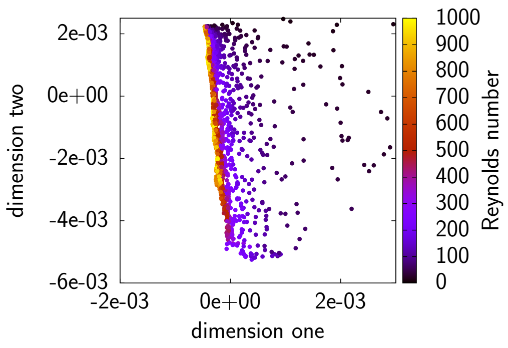
We compute 500 solution vectors for different Reynolds numbers and lid velocities and transform them with PCA into a five-dimensional space. These data points $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\} \subset \mathbb{R}^5$ are the input for the Laplacian eigenmaps method with our out-of-sample extension. We compute another $5,000$ points $\mathcal{T} = \{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_{M'}\} \subset \mathbb{R}^5$ as test data. In Fig. 44 we plot the two-dimensional embedding of the training and the test data where the eigenfunctions have been approximated on a sparse grid of level four, the regularization parameter is $\lambda = 0.001$, and the bandwidth of the kernel is $\sigma^2 = 60$. In the first row of Fig. 44, we color the data points with respect to the Reynolds number and in the second row with respect to the lid velocity. It is obvious that dimension one corresponds to the Reynolds number and dimension two to the lid velocity. Thus, the Laplacian eigenmaps method detects these two parameters and embeds the simulation data accordingly. With our sparse-grid-based out-of-sample extension, we are able to easily embed the $5,000$ test points in $\mathcal{T}$, see the second column in Fig. 44.

10.1.4. Real-World Example: Image Segmentation

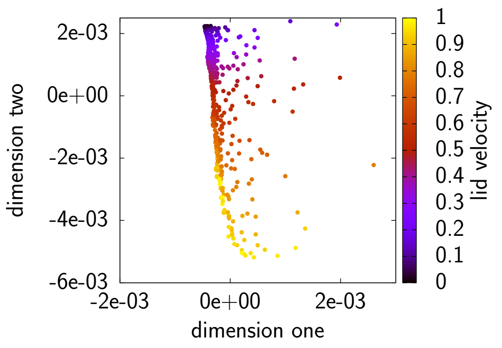
Usually, humans see in a picture more than just pixels. In an image showing a country landscape, they might recognize the sky, the sun, a meadow, animals, and so on. The task of image segmentation is to partition the image with respect to these objects. More precisely, the goal is to group pixels which are similar with respect to a specific



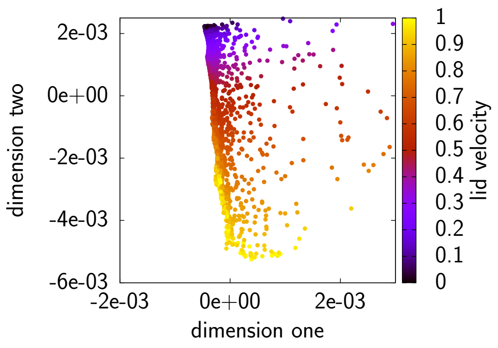
(a) colored w.r.t. *Re*, training



(b) colored w.r.t. *Re*, test



(c) colored w.r.t. velocity, training



(d) colored w.r.t. velocity, test

Figure 44: Two-dimensional embedding of high-dimensional flow field vectors: Each point corresponds to a flow field with a specific Reynolds number and lid velocity. The Laplacian eigenmaps method arranges the data points such that dimension one is the Reynolds number and dimension two the lid velocity.

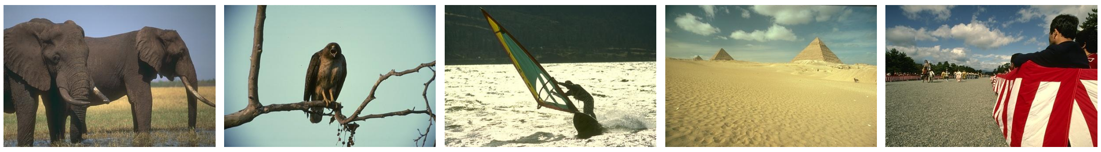

(a) elephant (b) bird (c) surfer (d) pyramids (e) parade

Figure 45: Five images of the Berkeley segmentation data set 300: The goal is to partition the images with respect to objects and visual appearance.

similarity measure [71]. In the following, we formulate this clustering problem in the context of spectral clustering and discuss that an out-of-sample extension is inevitable because an already small image with 200×200 pixels would lead to an eigenproblem of size $40,000 \times 40,000$. We then employ our sparse-grid-based out-of-sample extension to segment images of the Berkeley image data set [133].

Image segmentation as a clustering problem We follow the approach presented in [72]. The image is reduced to eight colors with minimum variance color quantization available in MATLAB. For each pixel, a local color histogram with a 5×5 window centered at the pixel is computed. Thus, for the i-th pixel, we have the histogram $\boldsymbol{h}_i \in \mathbb{R}^8$ where the j-th component $h_{i,j}$ of histogram $\boldsymbol{h}_i$ is the number of pixels in the 5×5 window around the i-th pixel with color index j after the color quantization. As proposed in [72] the distance between the i-th and the j-th pixel is given by the χ^2 test

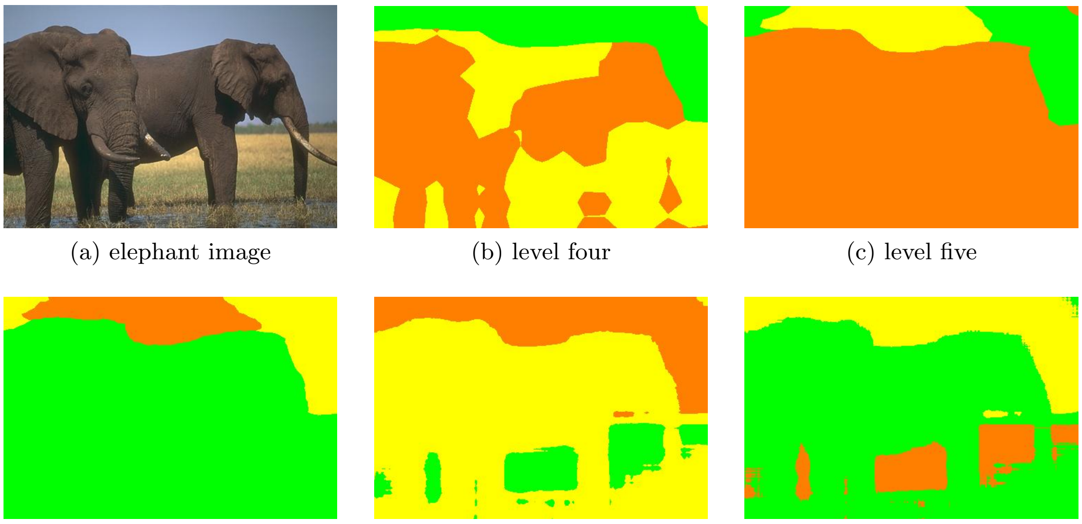
$$\chi^2_{ij} = \frac{1}{2} \sum_{b=1}^{8} \frac{(h_{i,b} - h_{j,b})^2}{h_{i,b} + h_{j,b}} \tag{92}$$

where we replaced zero entries of the histograms with 10^{-10} to avoid divisions by zero. To employ spectral clustering, we construct a similarity graph of the pixels where the edge connecting the i-th and j-th pixel has the weight

$$W_{ij} = \exp\left(\frac{-\chi^2_{ij}}{\sigma^2}\right) . \tag{93}$$

Note that we only consider colors to define the similarity measure and do not include information about textures or other properties of the image [71].

The images We consider the images available in the Berkeley Segmentation data set 300 which have also been used for the evaluation in related work [72, 8, 7]. In particular, we select the elephant (#296059), bird (#42049), pyramids (#260058), surfer (#62096), and parade (#145086) pictures shown in Fig. 45. Each of them has 481×321 pixels. Without an out-of-sample extension, we would have to solve an eigenproblem of size $154,401 \times 154,401$. This is clearly computationally infeasible. Therefore, we cluster with our sparse-grid-based out-of-sample extensions.



(a) elephant image (b) level four (c) level five

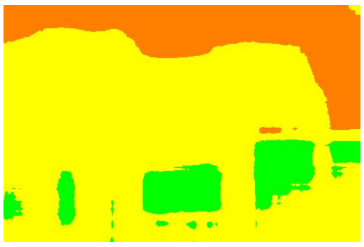
(d) level six (e) level seven (f) level eight

Figure 46: The elephant images segmented with our out-of-sample extension with regular sparse grids of level four (b) to eight (f): Sparse grids of level four to six are not sufficient to resolve the cluster boundaries. However, starting with level seven (833 grid points), we can clearly recognize the sky, the animals, and the steppe [3].

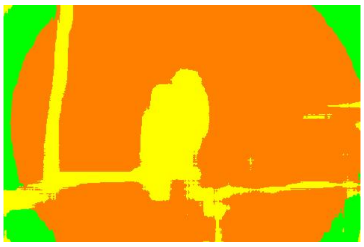
Results with regular sparse grids Besides the number of sparse grid points, the number of training pixels M should be kept small. Even though the size of the eigenproblem does not dependent on M, we have to multiply with the graph Laplacian $\boldsymbol{L}$ during the setup of the matrix $\boldsymbol{B}^T\boldsymbol{L}\boldsymbol{B} + \lambda\boldsymbol{C}$, cf. Sec. 10.1.2 and in particular (91). In the following examples, only $8,000$ randomly selected pixels (about five percent of all pixels) are put into the training set. The regularization parameter λ and the bandwidth of the Gaussian kernel are hand picked [3]. Let us first consider the elephant image, see Fig. 46. We set the number of clusters k to three and approximate the second and third eigenfunction on regular sparse grids including boundary points of level four to eight. The final cluster assignment is obtained by evaluating the approximated eigenfunctions on all pixels. It is clear that sparse grids of level four to six cannot resolve the boundaries of the clusters. But already with a sparse grid of level seven (833 grid points) we obtain a reasonable clustering which partitions the image into the sky, the animals, and the steppe. The segmentation of the other four images is shown in Fig. 47. Recall that we cluster only with respect to the color. Thus, for example, the pixels corresponding to the black clothes of the human in the parade picture are in the same cluster as the pixels corresponding to the black hair, see Figures 47i and 47j. The same is true for the white strips at the barrier and the white clouds in the sky.

Results with adaptive sparse grids Let us now reduce the number of sparse grid points with the help of adaptivity. Most images demand a high resolution only near the cluster boundaries and require only few grid points in regions that are put into one segment. Let

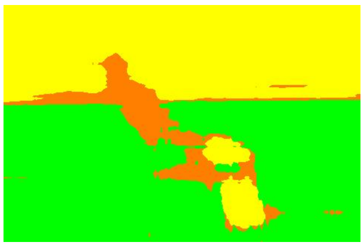
(a) elephant


(b) elephant, level 7

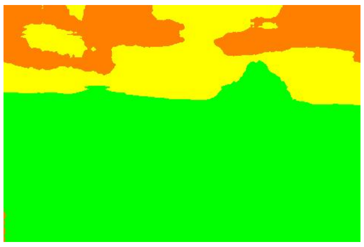
(c) bird

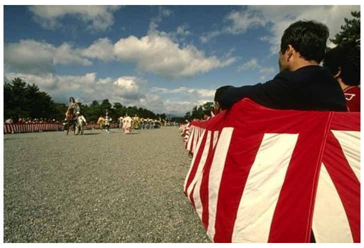

(d) bird, level 8

(e) surfer


(f) surfer, level 7

(g) pyramids


(h) pyramids, level 7


(i) parade

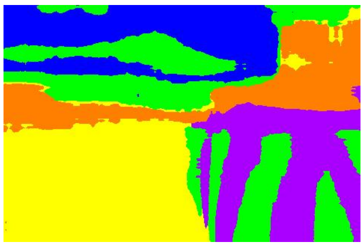
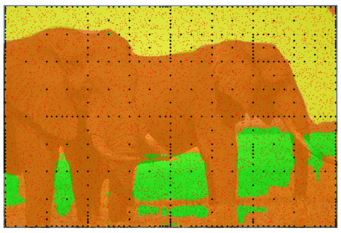
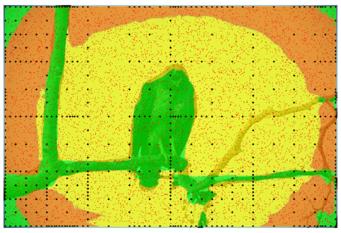
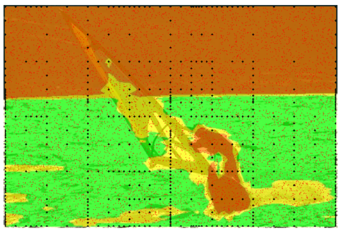
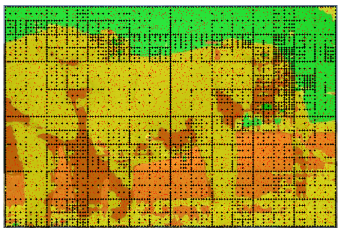

(j) parade, level 7

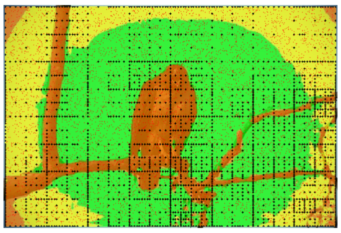
Figure 47: Segmentations of the five images computed with the sparse-grid-based out-of-sample extension on regular sparse grids are shown [3].


(a) elephant, 499 grid points


(b) bird, 497 grid points


(c) surfer, 514 grid points


(d) elephant, 4,180 grid points


(e) bird, 2,349 grid points

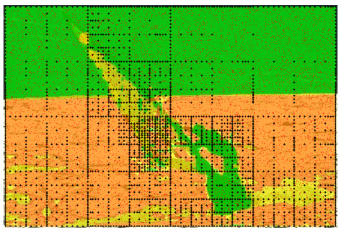

(f) surfer, 2,499 grid points

Figure 48: Elephant, bird, and surfer image segmented with adaptive sparse grids: Already with ≈ 500 grid points, we obtain a rough but very cheap segmentation of the images. To resolve the details, we have to increase the number of sparse grid points. It can be seen that most grid points are spent near cluster boundaries [3].

us first consider the top row of Fig. 48. It shows the segmentation of the elephant, bird, and surfer image where we show that only about 500 sparse grid points are necessary to capture the most important regions of the pictures. With more refinement steps, we obtain distinctly better results and can resolve most details, see the bottom row of Fig. 48. The adaptivity criterion leads to many grid points only near the cluster boundaries and few grid points in homogeneous regions.

10.2. Clustering with Sparse Grid Density Estimation

In this section we develop another clustering method based on sparse grids and on density estimation. We already mentioned the density-based notion of a cluster in the preliminaries. According to it a cluster is a dense region ("where many data points are") surrounded by a region of low-density ("where few data points are"). With our sparse grid density estimation method we want to find such regions. We first present the idea and the computational procedure of the density-based clustering method and then show some benchmark examples where we compare to other widely-used clustering methods. Finally, we consider a real-world problem where we cluster the finite element data of car crash simulations to gain insight into the crash behavior of the vehicle.

10.2.1. Clustering with Estimated Densities

Our method can be considered as a density-based clustering method, cf. Sec. 2.2.2. The data is represented as a similarity graph. Nonparametric density estimation is employed to detect regions of high density. These are then split off from the graph and define k connected components. These components of the graph represent the cluster centers and lead to the final cluster assignment.

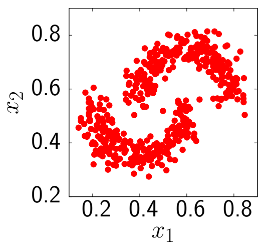
Clustering algorithm The steps of our density-based clustering method are visualized for the two moons data set in Fig. 49. In the first step, we compute the estimated density function $\hat{p}$ for the data $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ with the sparse grid density estimation method of Sec. 7.1, see Figures 49a and 49b. In step two, we represent the data $\mathcal{S}$ as a similarity graph $G = (\mathcal{S}, \mathcal{E})$ with vertices $\mathcal{S}$ and edges $\mathcal{E}$. The graph for the two moons data set is shown in Fig. 49c. Then (step three) we delete all vertices $\tilde{\mathcal{S}}$ (i.e., data points) of the graph G which lie inside a region of low density. The result is the graph $\hat{G} = (\hat{\mathcal{S}}, \hat{\mathcal{E}}) = (\mathcal{S} \setminus \tilde{\mathcal{S}}, \mathcal{E} \setminus \tilde{\mathcal{E}})$ which is split into, say, k (connected) components, each representing a cluster center (a high-density region). The connected components are shown in Fig. 49d. With these components we can assign cluster labels to the data points $\hat{\mathcal{S}} = \mathcal{S} \setminus \tilde{\mathcal{S}}$ remaining in the graph $\hat{G}$. To do so, we first number the components with $1, \ldots, k$ (descending with the number of vertices in the component) and associate the label j to all data points in component j. Thus, the result of step four is the labels $y_{i_1}, \ldots, y_{i_{\hat{M}}} \in \{1, \ldots, k\}$ of the data points in $\hat{\mathcal{S}} = \{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{\hat{M}}}\}$. We now can either stop here and treat the unlabeled data points in $\tilde{\mathcal{S}}$ as noise and outliers or we proceed to the optional step five and train a classifier $\hat{c}$ on the data

$$\{(\boldsymbol{x}_{i_1}, y_{i_1}), \ldots, (\boldsymbol{x}_{i_{\hat{M}}}, y_{i_{\hat{M}}})\}.$$

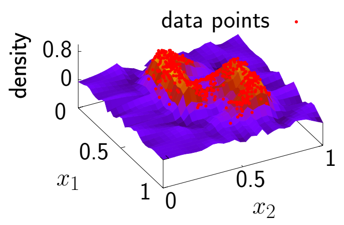
We can employ any classification method to construct $\hat{c}$, cf. Sec. 2.2.1. We summarize the algorithm to cluster the data $\mathcal{S} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_M\}$ in the following five steps:

1. Construct a similarity graph $G = (\mathcal{S}, \mathcal{E})$ to represent the data points in $\mathcal{S}$.
2. Employ sparse-grid-based density estimation to compute probability density function $\hat{p}$.
3. Create graph $\hat{G} = (\hat{\mathcal{S}}, \hat{\mathcal{E}}) = (\mathcal{S} \setminus \tilde{\mathcal{S}}, \mathcal{E} \setminus \tilde{\mathcal{E}})$ with k (connected) components by deleting vertices $\tilde{\mathcal{S}}$ and related edges $\tilde{\mathcal{E}}$ at which the estimated density function $\hat{p}$ evaluates to values below threshold ϵ.
4. Depending on their component, assign labels $y_{i_1}, \ldots, y_{i_{\hat{M}}} \in \{1, \ldots, k\}$ to the remaining vertices (i.e., data points) in $\hat{\mathcal{S}} = \{\boldsymbol{x}_{i_1}, \ldots, \boldsymbol{x}_{i_{\hat{M}}}\}$.
5. Optional: Train classifier $\hat{c}$ on data $\hat{\mathcal{S}}$ with labels $y_{i_1}, \ldots, y_{i_{\hat{M}}}$ and obtain labels for the data points in $\tilde{\mathcal{S}}$.

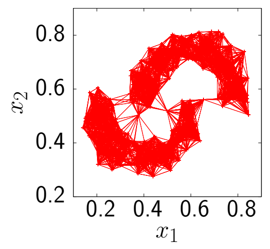
Implementation details We refer to Sec. 7.1 for details on step one and the sparse grid density estimation method. Note that the sparse grid density estimation method is well-suited for large data sets. In step two, we have to build a similarity graph from the



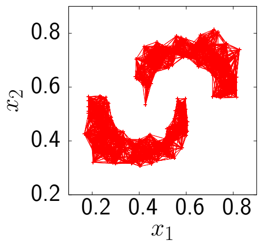
(a) data set



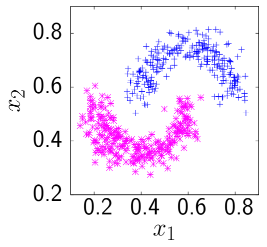
(b) estimated density function



(c) similarity graph



(d) components of graph

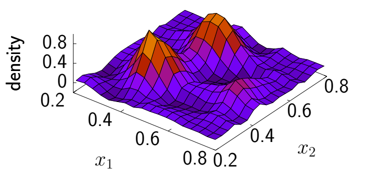


(e) clustered data set

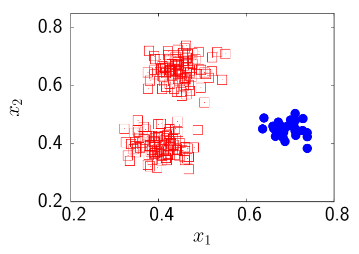
Figure 49: Our density-based clustering method estimates the density function (b) to split the similarity graph (c) of the data into connected components (d). These components correspond to the clusters (e) of the data.

data in $\mathcal{S}$. In the following, this is always a nearest neighbor graph with the Euclidean distance as similarity measure. This means, we compute the Euclidean distances between the data points and connect the data points with their $l \in \mathbb{N}$ nearest neighbors. This is a standard task in machine learning and therefore such procedures are available in many libraries. Our implementation is based on the Boost graph library. Even though the construction of the graph is in $\mathcal{O}(M^2)$, it can be easily parallelized, which makes processing of large amounts of data feasible. To determine connected components of the graph $\hat{G}$ in step three, we again use the routines available in the Boost graph library. For the classification in step five, we stick to a simple but very fast nearest neighbor classifier based on the approximated nearest neighbors library ANN. We tested more sophisticated classification approaches such as SVM (libsvm) and sparse-grid-based classification, but even though the nearest neighbor classifier did not always give the best result, it always worked well enough for our examples and it did not require to fine-tune parameters.

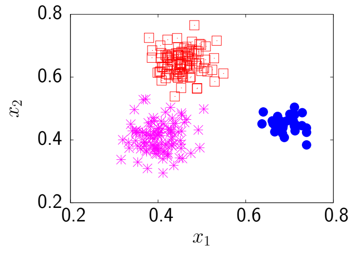
The threshold parameter Our density-based clustering method can be tuned with a couple of parameters. We have the parameters for the density estimation, e.g., the level of the sparse grid and the regularization parameter λ. We refer to Sec. 7.1 for details on them. Then, we need to decide on the number of neighbors l of the nearest neighbor graph. We show with the examples below that a reasonable guess of l is usually sufficient. Finally, the threshold ϵ has to be chosen. It controls what we consider to be a region



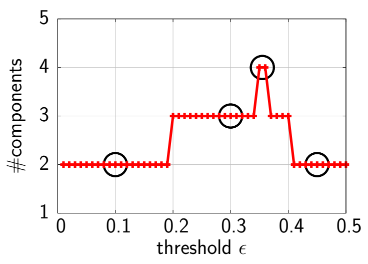
(a) estimated density function



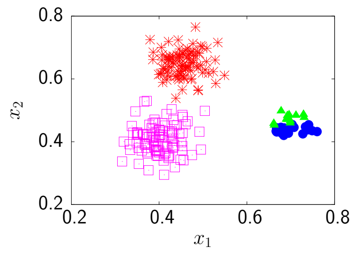
(b) $\epsilon = 0.1$, 2 clusters



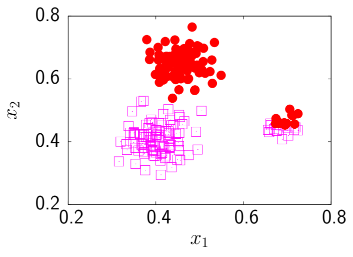
(c) $\epsilon = 0.3$, 3 clusters



(d) nr. of components



(e) $\epsilon = 0.36$, 4 clusters



(f) $\epsilon = 0.45$, 2 clusters

Figure 50: This simple example demonstrates the effect of the threshold ϵ on the clustering. If the threshold is set too low (b) we may miss clusters consisting of many points, if it is set too high (f) we may not be able to capture clusters with just a few points. Note that we only plotted every tenth point of the data set in order to make the symbols better legible.

of low and high density. We demonstrate its effect on the clustering result with the example shown in Fig. 50. In Fig. 50a we plotted the estimated density function of a data set with three clusters. We clearly see that the density function yields higher values near the two clusters on the left than near the cluster on the right. This means, the two clusters on the left are denser or stronger than the one on the right. In Fig. 50d, we plotted the number of connected components of the similarity graph versus the threshold ϵ. If we choose the threshold too low (< 0.2), the two strong clusters on the left are not separated, see Fig. 50b. The reason is that the estimated density function is still above the threshold between the two clusters. If we set the threshold too high (≥ 0.4) then we miss the weak cluster on the right because the density function is not high enough there, see Fig. 50f. However, we also have a large region between 0.2 and 0.4 where our method correctly predicts three clusters. Of course, there might be some outliers as in Fig. 50e.

Indicator for the number of clusters If we are looking for a specific number k of clusters in our data set, we can tune ϵ so that our method indeed gives us k clusters. If we do not know the number of clusters, we can plot the number of components versus the threshold as in Fig. 50d, and search for flat regions in the graph. A flat region means

	dbased clustering					k-means	SC	DBSCAN
	ϵ	l	λ	level	ARI	ARI	ARI	ARI
3S	0.15	10	1e-05	7 with b.	**1.00**	0.26	**1.00**	**1.00**
3Snoise	0.17	13	1e-05	7 with b.	0.73	0.31	**0.87**	0.73
olives	0.02	5	1e-10	5 w/out b.	**0.97**	0.32	0.69	0.62

Table 11: The table lists the ARI (adjusted rand index) of the cluster assignments obtained by our proposed method, k-means, spectral clustering, and DBSCAN. ARI of 1 means perfect clustering. Note that we employed a sparse grid without boundary points for the olives data set.

that the number of cluster centers stays constant for a wide range of the threshold which in turn means that the cluster centers have to correspond to very distinctive peaks in the estimated density function. The number of distinctive peaks is usually a good indicator for the number of clusters. We used this indicator in [29] where we also compared it to other heuristics.

10.2.2. Benchmark Problems

To assess our density-based clustering method, we apply it to several benchmark examples and compare the results with the cluster assignments obtained with k-means, spectral clustering (without an out-of-sample extension), and DBSCAN. Note that DBSCAN is also a density-based clustering method whereas k-means (convex method) and spectral clustering (see Sec. 10.1) pursue a very different approach. We used the scikit-learn implementation of these methods. For all of these methods, several parameters have to be tuned. We determined the parameter configurations by running each method for a reasonable range of parameters and kept the ones where the method yielded the best result. For details on the parameters and the parameter selection procedure, we refer to [148]. In Tab. 11 we show the ARI of the obtained cluster assignments for the 3S, 3Snoise, and olives data sets.

The 3S data set is a three-dimensional data set containing 625 points of three spheres of different sizes. The spheres are tangled with each other and thus the three clusters (three spheres) cannot be linearly separated. That is why k-means fails completely to detect the three clusters. All other methods can perfectly learn the data set, i.e., they achieve an ARI of 1.0. The 3Snoise data set is in principle the 3S data set but with distinctly more noise. This leads to a clear decrease in the ARI corresponding to spectral clustering, DBSCAN, and our density-based method. Still, our method is as good as DBSCAN.

The eight-dimensional olives data set is another famous benchmark problem where fatty acids of Italian olive oils (572 data points) have to be clustered according to the region of Italy where they come from. The number of clusters is three, i.e., the oils come from three different regions. Our density-based clustering method achieves an ARI of 0.97 which is distinctly better than 0.69 of spectral clustering and 0.62 of DBSCAN. The

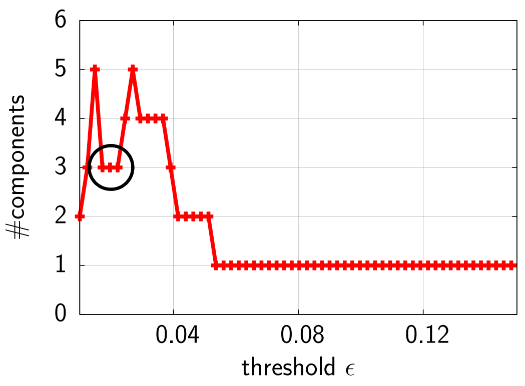


Figure 51: The number of connected components of the similarity graph of the olives data set versus the threshold ϵ: It shows that our density-based method correctly predicts three clusters at a very narrow range around $\epsilon \approx 0.02$. This indicates that one cluster is very weak and thus hard to capture.

reasons why our method performs so well here is that the data contains one very weak cluster which is found by our method. We accomplished that by setting the regularization parameter λ of the density estimation method to only 10^{-10}. Therefore, we almost overfitted the density function to our data but we also captured all the tiny details. Then, we plotted the number of connected components versus the threshold, see Fig. 51. The plot clearly shows that our method correctly predicts three clusters in a very narrow range near $\epsilon \approx 0.02$ only. As we have discussed above, this means that the value of the estimated density function at one cluster is very low compared to the other two clusters. This indicates that one of the three clusters is very weak.

10.2.3. Real-World Example: Analysis of Car Crash Data

In automotive industry, virtual product development and numerical computer simulations play a crucial role. We consider car crash tests where it is of particular advantage if the crash can be simulated rather than to build a prototype and conduct a live test. It requires accurate finite element models and robust solvers to compute solutions reflecting the physical behavior of the car during the crash. Just as the beams and parts of the car are crushed in reality so are the nodes of the finite element model moved accordingly in the simulation. The result of a simulation run is a huge pile of data. Only after a careful analysis, one is able to draw conclusions about the crash behavior of the car. Our task is to find groups of nodes with similar moving patterns and moving intensity. This helps to quickly spot parts of the car which behave unexpectedly. Furthermore, to detect noise, we distinguish between weak and strong clusters.

Let us consider a finite element model of a Chevrolet C2500 pick-up truck with $M = 65,975$ nodes, see Fig. 52. Let $\boldsymbol{x}_i^t \in \mathbb{R}^3$ be the position of the i-th node at time step t. We compute the displacement $\boldsymbol{d}_i = \boldsymbol{x}_i^{t_{17}} - \boldsymbol{x}_i^{t_1} \in \mathbb{R}^3$ between the beginning of the simulation (t_1) and the end of the crash (t_{17}) for all $i = 1, \ldots, M$. The displacements are then put into our data set $\mathcal{S} = \{\boldsymbol{d}_1, \ldots, \boldsymbol{d}_M\}$. Note that we consider here a single simulation. Later, in Sec. 11, we cluster the nodes of many simulation runs at once.

Four beams At first, we cluster only the $\approx 7,000$ nodes of four beams in the front of the car model, see Fig. 52. These beams distinctly influence the whole crash behavior.

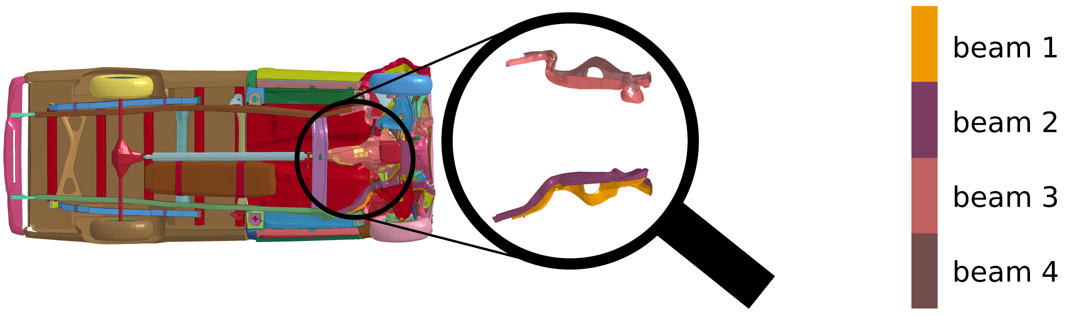


Figure 52: The Chevrolet C2500 pick-up truck from below and the beams number 1 to 4 in the front of the car: The displacement of these four beams is responsible for the crash behavior of the whole car.

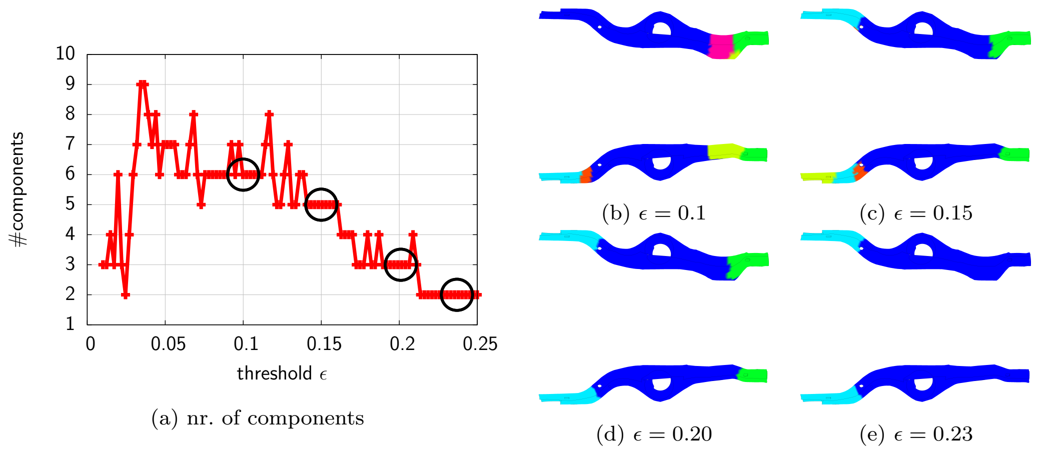


Figure 53: The plots show the cluster assignments of the four selected beams (b)-(e) with respect to different thresholds ϵ (a).

For the density estimation we employ a sparse grid of level five with boundary points (1,505 grid points) and set the regularization parameter λ to our standard choice 10^{-5}, cf. Sec. 7.1. Note that we have distinctly fewer grid points than data points. The number of nearest neighbors of the similarity graph is set to 10. Whereas the clustering result is fairly stable with respect to these parameters and thus no special tuning is necessary, the threshold ϵ can heavily influence the cluster assignment. We plot the number of components for a wide range $[0.01, 0.25]$ of the threshold ϵ in Fig. 53a and observe four intervals $[0.08, 0.11]$, $[0.14, 0.16]$, $[0.19, 0.21]$, and $[0.22, 0.25]$ where the curve is flat. These regions correspond to six, five, three, and two components, respectively. Hence, our method predicts between six and two clusters for a threshold $\epsilon \in [0.01, 0.25]$. The intervals are marked by circles in Fig. 53a.

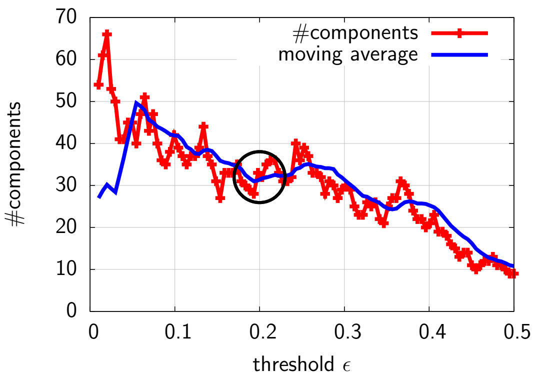
The four cluster assignments corresponding to those four intervals are plotted in Figures 53b to 53e. Overall, we see that for small ϵ we have many clusters in the front of the beams, but if we increase ϵ, they disappear. This suggests that the clusters in the front are not very strong. Even though the estimated density function has local maxima there, they are rather low compared to other local maxima of the function. Thus, if we set the threshold too high, we cannot capture them anymore, cf. Fig. 50. We have the opposite situation for the clusters in the rear where we can only separate the peaks corresponding to the clusters after we have increased the threshold to at least 0.15. This explains why new clusters appear from Figures 53b to 53c. The clusters in the rear seem to be more stable than the clusters in the front. They can be observed only after ϵ is set above 0.1 and remain until the end of the range of the threshold parameter.

This information helps to analyze the crash from an automotive engineering point of view. On the one hand, we only have weak clusters in the front and thus more or less noise. We do not find distinct moving patterns there. This means, we cannot expect to predict how the beams are crushed in the front. On the other hand, in the rear, we find quite strong clusters and thus the crash behavior can be predicted quite well there. Such information might help to make design decision on the material properties [29]. By visualizing the four beams at all time steps during the crash, we can gain similar insight into the behavior of the crash. However, this is more tedious than to run our clustering algorithm.

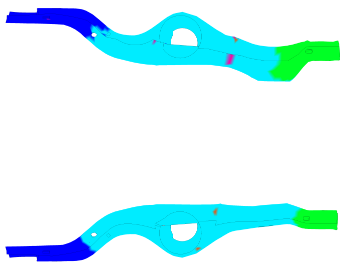
Whole car We now cluster all $M = 65,975$ nodes of the car model. In the case of the four beams, we could observe two effects in the crash behavior. We had weak clusters in the front, and strong clusters in the rear. When we cluster the whole car, we can expect to have many of these effects leading to many different clusters scattered all over the model. Therefore, we need a sparse grid with many grid points to capture all the peaks corresponding to these clusters. We set the sparse grid level to nine ($18,943$ grid points) without grid points at the boundary. We do not need boundary points because we have so many noisy points and outliers that nothing important is happening near the boundary. Note that we still have fewer grid points than data points as was also the case when we clustered the nodes of the four beams in the previous paragraph. As we have so many data points and expect many sharp peaks, we set the regularization parameter λ to 10^{-8} to better fit the data. All other parameters are kept as in the case of the four beams.

We plot the number of components versus the threshold in Fig. 54a. Because the curve wobbles around heavily, we cannot easily recognize a flat region. Therefore, we also plot the moving average of the previous ten points and find a flat region near $\epsilon = 0.2$. We plot the cluster assignment of the nodes of the four beams in Fig. 54a. It consists of three clusters and a few obviously not correctly clustered points. The clustering we find if we cluster the whole car with all $65,975$ points is very similar to the one where we cluster the four parts only.

Runtime The clusterings were performed on an Intel Core i7 870 with 8GB RAM on which the clustering of all nodes took 37 seconds. Thus, an interactive analysis of crash data with our clustering method on a common desktop machine is feasible. The computationally most expensive part of the clustering procedure is to construct the



(a) nr. of components



(b) four beams, $\epsilon = 0.2$

Figure 54: In (a) we show the number of components versus the threshold ϵ for the whole car, and in (b) the clustering of the four beams with threshold $\epsilon = 0.2$.

similarity graph and to estimate the density function. If we only change the threshold ϵ, we do not have to recompute the graph or the density function. Therefore, a new cluster assignment can be obtained and visualized within a few seconds. This allows us to quickly investigate cluster assignments corresponding to different threshold parameters which is especially useful to analyze the crash behavior of the model.

11. Case Study: Car Crash Simulation – Clustering of Nodes

In Sec. 10.2.3 we have already seen that the analysis of car crash simulation data is a challenging machine learning problem. We have shown how information about the crash behavior of a car can be extracted from the data of one simulation run with our density-based clustering method. This is certainly an interesting machine learning task in the context of simulation data analysis, however, we go one step further now.

The design process of a car usually does not involve only one simulation run but many runs with different design parameters, e.g., materials, plate thicknesses, or positions of bevel seams. One is then interested in assessing the different design parameters with respect to specific objectives such as the intrusion of the fire wall into the passenger compartment. Of course, every simulation run could be analyzed on its own, but this is a very tedious task. A natural step is to automatize this process with machine learning methods. Therefore, the goal is now to detect simulation runs with exceptional or extreme behavior from a whole bunch of simulation runs and to relate their behavior to the parameters.

To foster research in that direction, the German Federal Ministry of Education and Research (BMBF) funded the SIMDATA-NL project with the aim to create and implement such a workflow based on clustering and nonlinear machine learning methods [29]. The SIMDATA-NL project is a research consortia of academic groups and industry partners. Five German universities and research groups are involved: Institute for Numerical Simulation[3], U Bonn; Scientific Computing Group[4], TUM; Mathematical Statistics Group[5], TUM; Numerical Analysis of Partial Differential Equations Research Group[6], TU Berlin; and Fraunhofer SCAI[7]. The industry partners Audi[8], PDTec[9], and Volkswagen[10] provide car crash models and data to test and validate the algorithms and methods.

In the following, we first introduce the SIMDATA-NL workflow, discuss what role clustering of nodes plays during the workflow, and present results for data of a frontal crash of the Chevrolet pick-up truck we have already seen in Sec. 10.2.3 and of a Ford Taurus.

11.1. The SIMDATA-NL Simulation Data Analysis Workflow

The following workflow has been developed as part of the SIMDATA-NL project. We present the workflow, show in detail how to simultaneously cluster the nodes of many simulation runs at once, and propose an evaluation criterion for the obtained cluster assignments.

[3]http://wissrech.ins.uni-bonn.de/main/
[4]http://www5.in.tum.de/
[5]http://www-m4.ma.tum.de/
[6]http://page.math.tu-berlin.de/~garcke/
[7]http://www.scai.fraunhofer.de/
[8]http://www.audi.com/
[9]http://www.pdtec.de/
[10]http://www.volkswagen.com/

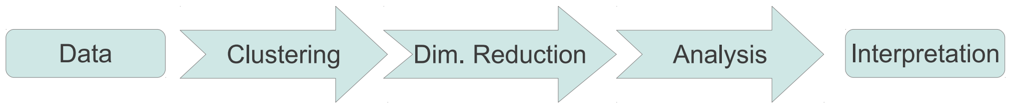


Figure 55: The SIMDATA-NL workflow for the analysis of car crash simulation data consists of a clustering and a nonlinear dimensionality reduction step [29].

SIMDATA-NL workflow The goal of the SIMDATA-NL workflow is to analyze simulation data with machine learning techniques in order to detect trends in the simulation runs (e.g., clusters, bifurcations), to find simulation runs with an exceptional behavior (e.g., outliers), and to map specific crash behavior to parameter configurations (e.g., slow variables). Of course, this is not the first attempt to automatically analyze car crash simulation data. In [2, 113] the focus is on linear (PCA) and nonlinear dimensionality reduction (diffusion maps). Clustering with k-means is treated in [174, 134].

Figure 55 illustrates the three steps of the SIMDATA-NL workflow. Input is the data stemming from computer simulations of car crash tests which have been performed for different parameter configurations. The crash is simulated with the LS-DYNA software which represents a car as a finite element model. For each simulation run and each time step, we obtain a vector with the displacements of the finite element nodes. We assume the data has already been cleaned and pre-processed accordingly, i.e., there are no missing or invalid values, there are no double records, and the data is available in a format which can be processed easily. In the first step (clustering), nodes of the car model with similar moving behavior during the crash are grouped together into clusters. These clusters reflect the characteristics of the crash of all simulation runs. By clustering the nodes, we can easily spot regions where the car is deformed most or at which point in time a specific beam starts to bend. Additionally, the clustering can also be seen as a pre-processing step for the dimensionality reduction in step two of the workflow. The crash data exhibits many effects simultaneously. Applying the nonlinear dimensionality reduction methods not on the nodes of the whole car but on each cluster separately allows us to obtain more detailed information about the crash. In the final step, the reduced data for each cluster is, for example, visualized to detect so-called slow variables which can be mapped onto the parameters. This reveals which parameters are important with respect to the bending behavior of the beams. The reduced data also quickly allows us to determine outliers. The workflow has been applied to a Chevrolet pick-up truck in [29]. Note that even though this workflow is only considered in the context of car crash simulation data here and in [29], it can be applied to other simulation data as well. In the following, we do not process the whole SIMDATA-NL workflow but consider only step number one where we cluster the nodes of the simulations.

Clustering of nodes Let $\boldsymbol{X}^t(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times 3}$ be the simulation run with parameter $\boldsymbol{\mu} \in \mathcal{D}$ at time step $t \in \mathbb{R}$. The i-th row of $\boldsymbol{X}^t(\boldsymbol{\mu})$ contains the position of the i-th node in $\mathbb{R}^3$. Let

$$\tilde{\boldsymbol{X}}^{t_i,t_j}(\boldsymbol{\mu}) = \boldsymbol{X}^{t_j}(\boldsymbol{\mu}) - \boldsymbol{X}^{t_i}(\boldsymbol{\mu}) \in \mathbb{R}^{\mathcal{N}\times 3} \tag{94}$$

denote the matrix of the displacements with respect to two time steps $t_i, t_j \in \mathbb{R}$. We have M simulation runs and thus M matrices $\tilde{\boldsymbol{X}}^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, \tilde{\boldsymbol{X}}^{t_i,t_j}(\boldsymbol{\mu}_M)$. We derive two different data sets for the clustering. In the first data set $\mathcal{S}_{\max}^{t_i,t_j}$, we only keep the maximum displacements, i.e.,

$$\begin{aligned} \mathcal{S}_{\max}^{t_i,t_j} = \Big\{ \Big[&\max\left\{ \tilde{X}_{l1}^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, \tilde{X}_{l1}^{t_i,t_j}(\boldsymbol{\mu}_M) \right\}, \\ &\max\left\{ \tilde{X}_{l2}^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, \tilde{X}_{l2}^{t_i,t_j}(\boldsymbol{\mu}_M) \right\}, \\ &\max\left\{ \tilde{X}_{l3}^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, \tilde{X}_{l3}^{t_i,t_j}(\boldsymbol{\mu}_M) \right\} \Big]^T \,|\, l = 1, \dots, \mathcal{N} \Big\} \subset \mathbb{R}^3 . \end{aligned} \tag{95}$$

The data set $\mathcal{S}_{\max}^{t_i,t_j}$ is a subset of $\mathbb{R}^3$ and contains $\mathcal{N}$ points. For the second data set $\mathcal{S}_{\text{euc}}^{t_i,t_j}$, we compute the Euclidean norm of the rows of each matrix $\tilde{\boldsymbol{X}}^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, \tilde{\boldsymbol{X}}^{t_i,t_j}(\boldsymbol{\mu}_M)$ and obtain the vectors $\boldsymbol{d}^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, \boldsymbol{d}^{t_i,t_j}(\boldsymbol{\mu}_M) \in \mathbb{R}^{\mathcal{N}}$. The data set $\mathcal{S}^{t_i,t_j}$ is then defined as

$$\mathcal{S}_{\text{euc}}^{t_i,t_j} = \left\{ \left[d_l^{t_i,t_j}(\boldsymbol{\mu}_1), \dots, d_l^{t_i,t_j}(\boldsymbol{\mu}_M) \right]^T \,|\, l = 1, \dots, \mathcal{N} \right\} \subset \mathbb{R}^M , \tag{96}$$

where, for example, $d_l^{t_i,t_j}(\boldsymbol{\mu}_1)$ is the l-th component of $\boldsymbol{d}^{t_i,t_j}(\boldsymbol{\mu}_1)$. Thus, we obtain a data set with $\mathcal{N}$ points and each data point is in $\mathbb{R}^M$. If it does not matter or if it is clear from the context if we mean $\mathcal{S}_{\max}^{t_i,t_j}$ or $\mathcal{S}_{\text{euc}}^{t_i,t_j}$, we simply write $\mathcal{S}^{t_i,t_j}$ or even $\mathcal{S}$ instead.

Having constructed either (95) or (96), the data set is clustered into k subsets $\mathcal{S}_1, \dots, \mathcal{S}_k$. The finite element model of the car is already divided into so-called parts, i.e., a partition of the set $\mathcal{S}$ is already given. These parts are defined by the engineering process and do not take the behavior of the crash into account. Compared to them, the clustering has two main advantages. First, the number of clusters can be freely chosen whereas the number of parts is fixed. Second, the nodes of one part might show a very different bending behavior. For example in a frontal crash a beam reaching all over the car is certainly more crushed in the front than in the rear of the car. The clustering allows us to further divide such parts into multiple clusters or merge a couple of smaller parts together.

Finally, we emphasize that clustering the data sets (95) and (96) is very different from the clustering task in Sec. 10.2.3. Whereas in Sec. 10.2.3 the nodes of only one simulation run were clustered, we consider now all simulation runs at once. Thus, the clustering reflects the crash behavior of all simulation runs with all parameter configurations.

Evaluation of clustering results in the context of the SIMDATA-NL workflow The purpose of the clustering in the SIMDATA-NL workflow is on the one hand to obtain a partition of the model to derive information about the crash behavior and on the other hand to provide the input data for the dimensionality reduction methods. These two objectives also determine how we assess the quality of a clustering $\mathcal{S}_1^{t_i,t_j}, \dots, \mathcal{S}_k^{t_i,t_j}$ of the car model.

We first check by visual inspection if the nodes in one cluster $\mathcal{S}_l^{t_i,t_j}$ show the same moving pattern. Therefore, we plot the positions of the nodes and color them according to their cluster assignments. With this simple criterion, we can quickly decide if a clustering is reasonable and if it captures the crash behavior.

Besides this qualitative criterion, we also want to have a quantitative one. Recall that the SIMDATA-NL workflow contains a dimensionality reduction step where a low-dimensional embedding of the displacements is constructed. The embedding is obtained with nonlinear dimensionality reduction methods to achieve reduced data with a low dimension but which still contains the relevant information of the crash. However, the nonlinear dimensionality reduction methods are not directly applied to all data points at once but to each cluster separately.

Now, let $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k = \mathcal{S}'_1 \uplus \cdots \uplus \mathcal{S}'_k$ be two different clusterings. We compare the two clusterings by computing the reconstruction error corresponding to $\mathcal{S}_1, \dots, \mathcal{S}_k$ and $\mathcal{S}'_1, \dots, \mathcal{S}'_k$. Using the example of the Chevrolet pick-up truck, it is shown in [29] that a good clustering with respect to the POD reconstruction error is also a good clustering with respect to the reconstruction error obtained with a nonlinear dimensionality reduction method. Hence, we leave the nonlinear dimensionality reduction methods to the respective partners in the SIMDATA-NL project and consider only POD here. The reconstruction procedure based on POD is summarized in Alg. 6. The data points in $\mathcal{S}$ are clustered into $\mathcal{S}_1, \dots, \mathcal{S}_k$. For each cluster $\mathcal{S}_i$, the data matrix $\boldsymbol{T}_i^{\text{train}}$ containing the displacements of the nodes corresponding to the cluster $\mathcal{S}_i$ is created. A POD basis $\boldsymbol{V}_i$ is constructed and the test data matrix $\boldsymbol{T}_i^{\text{test}}$ is reconstructed with $\hat{\boldsymbol{T}}_i^{\text{train}} = \boldsymbol{V}_i(\boldsymbol{V}_i^T \boldsymbol{T}_i^{\text{test}})$. All matrices $\hat{\boldsymbol{T}}_1^{\text{test}}, \dots, \hat{\boldsymbol{T}}_k^{\text{test}}$ are combined in one matrix $\hat{\boldsymbol{T}}^{\text{test}}$. With $\boldsymbol{T}^{\text{test}}$ and $\hat{\boldsymbol{T}}^{\text{test}}$, we compute the error

$$\frac{1}{|\mathcal{T}|} \sum_{j=1}^{|\mathcal{T}|} \frac{\sum_{i=1}^{3\mathcal{N}} \left(T_{ij}^{\text{test}} - \hat{T}_{ij}^{\text{test}} \right)^2}{\sum_{i=1}^{3\mathcal{N}} T_{ij}^2} . \tag{97}$$

The same procedure but with nonlinear dimensionality reduction methods is used in [29].

11.2. Chevrolet C2500 Pick-Up Truck

A frontal crash of a Chevrolet C2500 pick-up truck is analyzed with the clustering step of the SIMDATA-NL workflow. We consider simulation runs with large parameter changes leading to simulation data with distinct characteristics. First, it is shown that our sparse-grid-based clustering methods yield a cluster assignment reflecting the behavior of the car during the crash whereas k-means fails. Second, the clustered crash data is reconstructed with the procedure described in Alg. 6 to demonstrate that the clustering of the nodes indeed improves the reconstruction error.

Chevrolet C2500 pick-up truck We consider here the frontal crash of a Chevrolet C2500 pick-up truck, see Fig. 57. A finite element model of the truck is available from the National Crash Analysis Center[11] (NCAC). If we ignore the obstacle, the model consists of 65,975 finite element nodes and 239 parts. For the SIMDATA-NL project,

[11] http://www.ncac.gwu.edu/

Algorithm 6 Reconstruction of car crash data with POD

```
 1: procedure reconPOD($n, k, M, \mathcal{S}, \mathcal{T}, \tilde{\boldsymbol{X}}(\boldsymbol{\mu}_1), \dots, \tilde{\boldsymbol{X}}(\boldsymbol{\mu}_M)$)
 2:     initialize $\boldsymbol{T}^{\text{test}}, \hat{\boldsymbol{T}}^{\text{test}} \in \mathbb{R}^{3\mathcal{N} \times |\mathcal{T}|}$ with zeros
 3:     cluster data in $\mathcal{S}$ into $\mathcal{S}_1, \dots, \mathcal{S}_k$
 4:     derive clustering $\mathcal{J}_1, \dots, \mathcal{J}_k$ of set $\{1, \dots, 3\mathcal{N}\}$ according to clustering $\mathcal{S}_1, \dots, \mathcal{S}_k$
 5:     for $i = 1 : k$ do
 6:         form matrices ${}^i\tilde{\boldsymbol{X}}(\boldsymbol{\mu}_j)$ with rows of $\tilde{\boldsymbol{X}}(\boldsymbol{\mu}_j)$ with indices in $\mathcal{J}_i$ for $j = 1, \dots, M$
 7:         let $\mathcal{N}_i = |\mathcal{S}_i|$ and assemble matrix
                $$\boldsymbol{T}_i = \begin{bmatrix} {}^i\tilde{X}_{1,1}(\boldsymbol{\mu}_1) & \dots & {}^i\tilde{X}_{1,1}(\boldsymbol{\mu}_M) \\ \vdots & \ddots & \vdots \\ {}^i\tilde{X}_{\mathcal{N}_i,3}(\boldsymbol{\mu}_1) & \dots & {}^i\tilde{X}_{\mathcal{N}_i,3}(\boldsymbol{\mu}_M) \end{bmatrix} \in \mathbb{R}^{3\mathcal{N}_i \times M}$$
 8:         store the columns of $\boldsymbol{T}_i$ with indices in $\{1, \dots, 3\mathcal{N}\} \setminus \mathcal{T}$ in $\boldsymbol{T}_i^{\text{train}}$
 9:         store the columns of $\boldsymbol{T}_i$ with indices in $\mathcal{T}$ in $\boldsymbol{T}_i^{\text{test}}$
10:         compute $n$ POD basis vectors $\boldsymbol{V} \in \mathbb{R}^{3\mathcal{N}_i \times n}$ for data in $\boldsymbol{T}_i^{\text{train}}$
11:         compute $\hat{\boldsymbol{T}}_i^{\text{test}} = \boldsymbol{V}_i(\boldsymbol{V}_i^T \boldsymbol{T}_i^{\text{test}})$
12:         store rows of $\boldsymbol{T}_i^{\text{test}}, \hat{\boldsymbol{T}}_i^{\text{test}}$ in rows of $\boldsymbol{T}^{\text{test}}, \hat{\boldsymbol{T}}^{\text{test}}$ with indices in $\mathcal{J}_i$
13:     end for
14: return $\boldsymbol{T}^{\text{test}}, \hat{\boldsymbol{T}}^{\text{test}}$
15: end procedure
```

the Fraunhofer SCAI[12] computed 126 simulation runs where nine parameters were varied. These parameters correspond to the plate thicknesses of specific beams. Each of the 126 simulation runs consists of 17 time steps. At time step t_3 the truck hits the obstacle, at step t_7 the crash has happened and the rebound starts. In the following, we restrict ourselves to the displacements from time step t_3 to t_4, from t_6 to t_7, and from t_1 to t_{17}.

Even though the car model consists of 239 parts, especially four beams determine the crash behavior of the truck. These four beams are shown in Figures 57e and 57f. In Fig. 58 the four beams are plotted again but for three different simulation runs at the last time step, i.e., after the crash. In the rear (left), a different bending behavior in each simulation run can be recognized. This shows that our data exhibits phenomena such as buckling and high sensitivity to parameter changes and it also shows that a clustering of the nodes is badly needed because the nodes in the rear of the parts behave very differently from the nodes in the front.

We employ the clustering methods introduce in Sections 10.1 and 10.2, and compare with k-means. With k-means, we cluster the set $\mathcal{S}_{\text{euc}}^{t_i,t_j}$ and with the density-based clustering method the set $\mathcal{S}_{\max}^{t_i,t_j}$. The density function for the density-based clustering method is estimated on a sparse grid of level six (3,713 grid points). The number of neighbors for the nearest neighbor similarity graph, the threshold ϵ, and the regularization parameter λ for the density estimation method are all determined with respect

[12]http://www.scai.fraunhofer.de

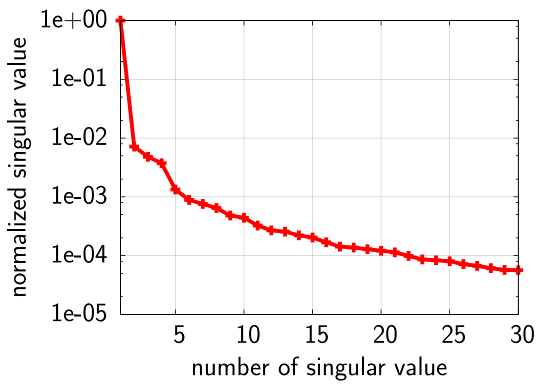


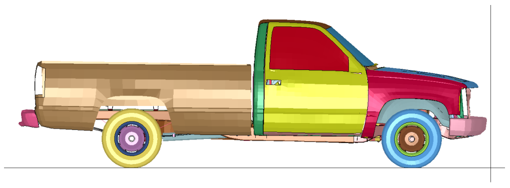
Figure 56: The 30 largest singular values of the data matrix corresponding to $\mathcal{S}_{\text{euc}}^{t_6,t_7}$: The first seven principal components contain most of the information in $\mathcal{S}_{\text{euc}}^{t_6,t_7}$.

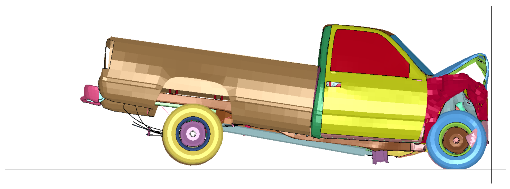
to the exponential density measure, see Sec. 2.2.2. For the spectral clustering with the sparse-grid-based out-of-sample extension, the data points in $\mathcal{S}_{\text{euc}}^{t_i,t_j}$ are projected onto the first seven principal components. Figure 56 shows the normalized singular values corresponding to the data matrix of $\mathcal{S}_{\text{euc}}^{t_6,t_7}$. It is clear that the first seven components capture most information of the data. A sparse grid of level four (799 grid points) is used for the out-of-sample extension. The regularization parameter and the bandwidth of the kernel for the graph are determined with respect to the exponential density measure.

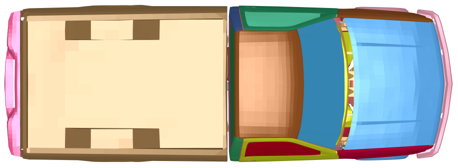
Clustering of four beams Let us first have a look at four longitudinal chassis beams of the truck, see Figures 57e and 57f. As already discussed above, these beams are essential for the crash behavior. With 6,674 nodes they contain more than ten percent of the nodes of the whole truck. Our density-based clustering method predicts eight clusters. This is verified by a statistical method in [29]. Recall that in Sec. 10.2.3 our density-based method predicted between two and six clusters but for a single simulation run and thus for a different data set.

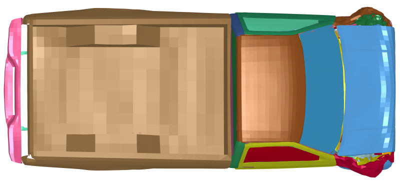
The cluster assignment of the nodes of the four beams for $\mathcal{S}^{t_3,t_4}, \mathcal{S}^{t_6,t_7}$, and $\mathcal{S}^{t_1,t_{17}}$ is shown in Fig. 59. The parameters for the spectral and density-based clustering were determined for each data set individually, see above. In case of spectral clustering, an eigenproblem of size 799×799 instead of $6,674 \times 6,674$ was solved due to our sparse-grid-based out-of-sample extension with level four (799 grid points). We approximated the first seven eigenvectors.

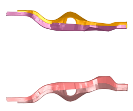
Let us first consider the results for the displacements from time step t_3 to t_4. The cluster assignments yielded by spectral and density-based clustering look very similar. Both find one big cluster in the rear and many rather small clusters in the front. This suggests that all nodes from the rear of the beams behave similar from time step t_3 to t_4. Indeed, this fits to what we mentioned above, namely, that at time step t_4 the obstacle is hit and the front of the car is crushed. In contrast, k-means divides the rear of the beams further and thus its cluster assignment does not capture this behavior. Let us now come to time steps t_6 and t_7. There we have the opposite situation. Only the rear of the beams is crushed. This is reflected in the cluster assignments. We have multiple clusters in the rear but only very few in the front. All clustering methods were able


(a) t_1, Truck, side view


(b) t_{17}, Truck, side view


(c) t_1, Truck, aerial view


(d) t_{17}, Truck, aerial view


(e) t_1, four beams, aerial view

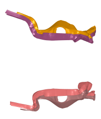
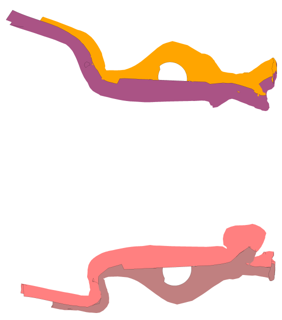
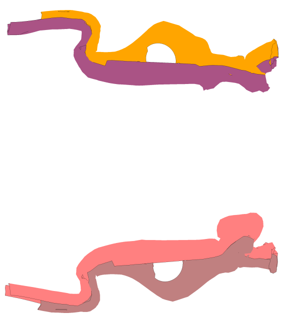
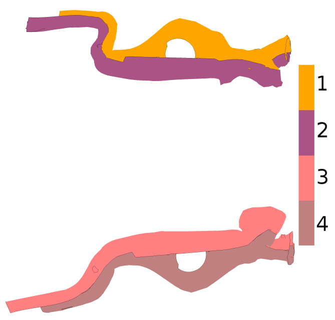

(f) t_{17}, four beams, aerial view

Figure 57: The Chevrolet C2500 pick-up truck before and after the crash: Figures (e) and (f) show the position of the four beams with respect to the truck. We refer to Fig. 58 for enlarged plots of the beams.

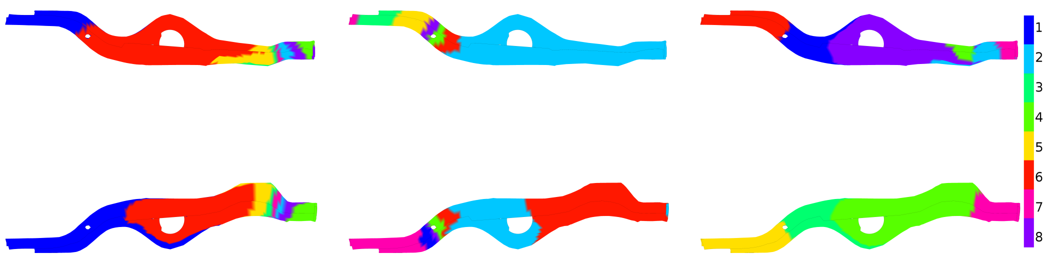

(a) simulation run one


(b) simulation run two

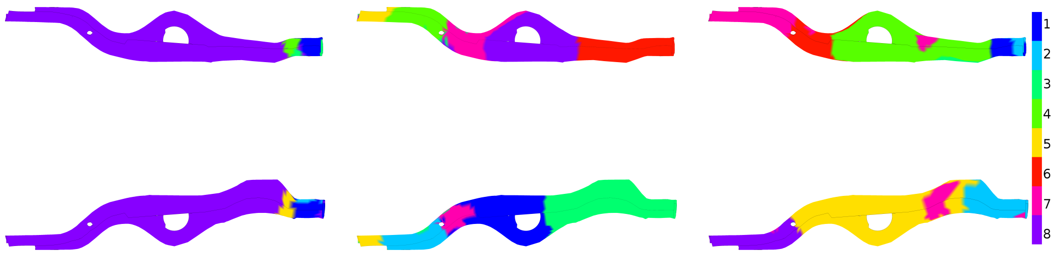


(c) simulation run three

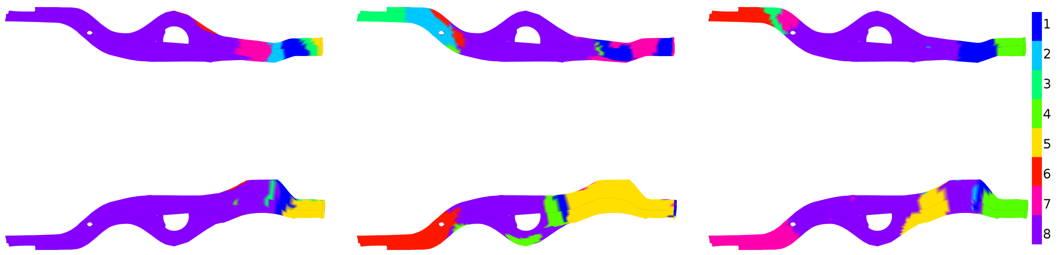
Figure 58: Four specific beams of the Chevrolet truck after the crash for three different simulation runs: A different bending behavior in the rear can be recognized. This demonstrates that the parts given by the engineering process are not a good partition of the nodes of the car model with respect to the crash behavior.



(a) $\mathcal{S}^{t_3,t_4}$, k-means (b) $\mathcal{S}^{t_6,t_7}$, k-means (c) $\mathcal{S}^{t_1,t_{17}}$, k-means

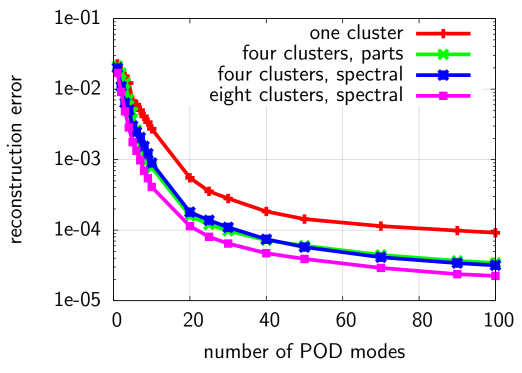


(d) $\mathcal{S}^{t_3,t_4}$, spec. clustering (e) $\mathcal{S}^{t_6,t_7}$, spec. clustering (f) $\mathcal{S}^{t_1,t_{17}}$, spec. clustering

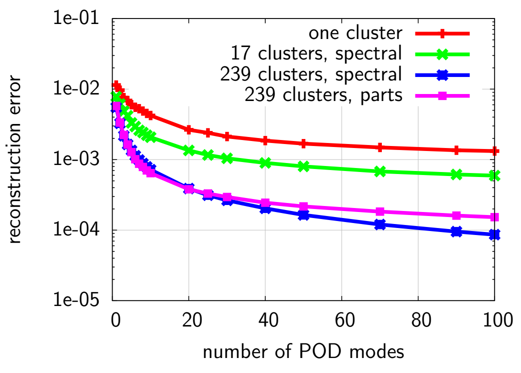


(g) $\mathcal{S}^{t_3,t_4}$, density-based (h) $\mathcal{S}^{t_6,t_7}$, density-based (i) $\mathcal{S}^{t_1,t_{17}}$, density-based

Figure 59: Shows the cluster assignments of the nodes of the four beams obtained with the k-means method, spectral clustering with our sparse-grid-based out-of-sample extension, and our density-based clustering method [29]. From t_3 to t_4 only the front of the beams is crushed. This is recognized by spectral and density-based clustering. From t_6 to t_7 only the rear of the beam bends but the front does not distinctly change anymore. Thus, many clusters are found in the rear but only one cluster in the front. Both of these effects are reflected in the clustering of the displacements from time step t_1 to t_{17}.



(a) four beams



(b) whole truck

Figure 60: Comparison of cluster methods with respect to the reconstruction error of the displacements: On the left, the reconstruction error corresponding to the four beams is shown, and on the right, the reconstruction error for the whole truck. As we increase the number of clusters the reconstruction error gets reduced.

to detect this behavior. Finally, if we consider the displacements from time step t_1 to time step t_{17}, the whole crash should be captured in the cluster assignment. Indeed, all methods have multiple clusters in the rear and in the front and only one cluster in the middle. This is in good agreement with what we have seen so far where the front got crushed from time step t_3 to t_4, and the rear from t_6 to t_7.

With the procedure in Alg. 6 we compute the reconstruction error (97) of the displacements of the nodes of the four beams from time step t_6 to t_7 for different numbers of POD basis vectors, see Tab. 12. We report results for one, four, and eight clusters. Let us first consider four clusters where we compare the error corresponding to the four parts of the beams with the error corresponding to the cluster assignment obtained with k-means, spectral clustering, and density-based clustering. Recall that the four beams correspond to four parts of the car model. In most cases, spectral clustering gives the best results. However, just reconstructing each part separately is almost as good. Now, as we said above, one advantage of the clustering is that we can easily increase the number of clusters. Therefore, let us consider the results with eight clusters. Spectral clustering with the sparse-grid-based out-of-sample extension clearly outperforms the other clustering methods. In Fig. 60a the results are summarized. Fig. 60a also confirms that the separation into multiple clusters is worthwhile because the reconstruction error is highest with only one cluster, can be reduced if we have four clusters, and gets even better with eight clusters.

Clustering of the whole truck Let us now consider the whole truck. The corresponding reconstruction errors are shown in Tab. 13. Our density-based clustering method predicts 17 clusters. We cluster all 65,975 finite element nodes into 17, 50, 100, and 239 clusters. Recall that the truck has 239 parts. With the sparse-grid-based out-of-sample extension, we had to solve an eigenproblem with size 799×799 instead of one with size $65,975 \times$

	1 cluster	4 clusters				8 clusters		
n	-	parts	k-means	spectral	dbased	k-means	spectral	dbased
1	2.28e-02	2.13e-02	2.85e-02	**1.99e-02**	2.20e-02	2.36e-02	**1.70e-02**	1.95e-02
2	1.83e-02	1.69e-02	1.41e-02	**1.09e-02**	1.31e-02	1.22e-02	**9.09e-03**	9.76e-03
3	1.52e-02	1.21e-02	8.60e-03	**6.37e-03**	8.79e-03	7.88e-03	**4.84e-03**	6.38e-03
4	1.22e-02	7.23e-03	**4.72e-03**	5.02e-03	4.93e-03	4.29e-03	**2.86e-03**	3.80e-03
5	6.80e-03	4.70e-03	3.12e-03	3.00e-03	**2.71e-03**	2.25e-03	**1.77e-03**	2.35e-03
6	6.03e-03	2.63e-03	2.29e-03	2.47e-03	**2.11e-03**	1.58e-03	**1.33e-03**	1.61e-03
7	5.03e-03	**1.60e-03**	1.75e-03	2.06e-03	1.68e-03	1.06e-03	**9.77e-04**	1.17e-03
8	4.10e-03	1.24e-03	**1.14e-03**	1.54e-03	1.26e-03	7.68e-04	**6.95e-04**	9.69e-04
9	3.40e-03	9.90e-04	**9.61e-04**	1.21e-03	9.93e-04	6.20e-04	**5.45e-04**	7.79e-04
10	2.73e-03	**7.76e-04**	7.90e-04	8.95e-04	8.25e-04	5.12e-04	**4.09e-04**	5.75e-04
20	5.52e-04	**1.60e-04**	2.41e-04	1.80e-04	2.14e-04	1.62e-04	**1.14e-04**	1.42e-04
25	3.55e-04	**1.20e-04**	1.84e-04	1.38e-04	1.45e-04	1.22e-04	**8.04e-05**	9.96e-05
30	2.81e-04	**9.79e-05**	1.38e-04	1.10e-04	1.08e-04	9.69e-05	**6.46e-05**	7.30e-05
40	1.84e-04	**7.19e-05**	1.03e-04	7.42e-05	8.09e-05	7.25e-05	**4.72e-05**	5.04e-05
50	1.43e-04	6.02e-05	8.70e-05	**5.75e-05**	6.56e-05	6.10e-05	**3.90e-05**	3.92e-05
70	1.14e-04	4.45e-05	7.05e-05	**4.13e-05**	5.08e-05	4.62e-05	2.92e-05	**2.87e-05**
90	9.86e-05	3.67e-05	6.00e-05	**3.41e-05**	4.22e-05	3.81e-05	2.38e-05	**2.34e-05**
100	9.19e-05	3.43e-05	5.59e-05	**3.17e-05**	3.99e-05	3.57e-05	2.24e-05	**2.18e-05**

Table 12: Reconstruction error of the displacements of the four beams of the Chevrolet truck with n POD modes: In case of four and eight clusters, spectral clustering performs best. In any case, clustering the data and reconstructing each cluster separately is better than reconstructing the displacements of all four beams at once.

$65,975$. We approximated the first 30 eigenvectors. In Tab. 13, we see that the spectral method works well for a small number of clusters, whereas k-means seems to be a good choice if we have a large number of clusters. It is known that spectral clustering might not perform well if many clusters are sought. Even though there is no solution working in all settings, there are a couple possibilities and workarounds, see, e.g., [8]. The density-based method does not perform very well at all. The results are summarized in Fig. 60b. Again, clustering does distinctly improve the reconstruction error. We also see that the clustering gives better results than the parts.

11.3. Ford Taurus

A frontal crash of a Ford Taurus is simulated. A car model with more than ten times as many nodes as in the Chevrolet truck model is used. The simulation runs are computed for only slightly different parameters leading to data where it is hard to detect characteristics. We show that our algorithms can still cope with this large data set by analyzing the corresponding simulation data with our clustering methods.

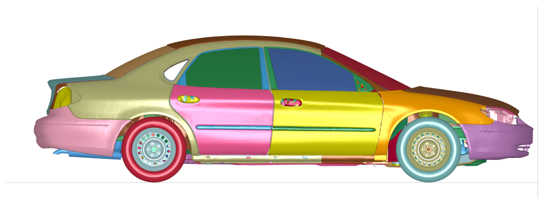
Ford Taurus The finite element model of the Ford Taurus is also available from the National Crash Analysis Center (NCAC), see Fig. 61. It has $874,728$ nodes which form 741 parts. The Fraunhofer SCAI computed 251 simulation runs with minor changes in the parameters. Each simulation run consists of 152 time steps. We are interested in time step t_{75} and thus in the data set $\mathcal{S}^{t_1,t_{75}}$. Time step t_{75} is after the frontal crash but before the rebound. Just as for the truck, we can identify 19 important parts, see Figures 61e and 61f.

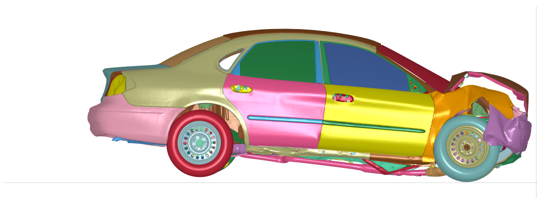
We compare k-means, spectral, and density-based clustering. The k-means method is directly applied to the set $\mathcal{S}^{t_1,t_{75}}_{\text{euc}}$, the density-based method to $\mathcal{S}^{t_1,t_{75}}_{\max}$. The density is estimated on a sparse grid of level eight. All other parameters are determined with respect to the exponential density measure. For the spectral clustering, the data $\mathcal{S}^{t_1,t_{75}}_{\text{euc}}$ is again projected onto the first seven principal components. They contain enough information for a good clustering. The out-of-sample extension approximates the eigenfunctions on sparse grids of level four. The other parameters are found with the exponential density measure.

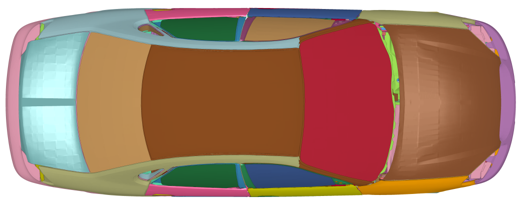
Clustering of 19 selected parts Let us first consider the nodes of the 19 parts shown in Figures 61e and 61f. These parts are essential for the crash behavior of the car. Our density-based clustering predicts 22 clusters. We compare the reconstruction error corresponding to the cluster assignments obtained with k-means, spectral clustering, and density-based clustering in Tab. 14. We only show the results for up to nine POD modes. Since the parameters of the simulation runs differ only slightly, the snapshots are very similar, and we achieve already very high accuracies with only nine POD modes. Clearly, in case of 19 clusters, the 19 parts perform best. If we increase the number of clusters to 22, as predicted by the density-based method, our density-based method performs best compared to spectral clustering and k-means. This is again summarized in Fig. 62a where we again see that first clustering and then reconstructing each cluster individually

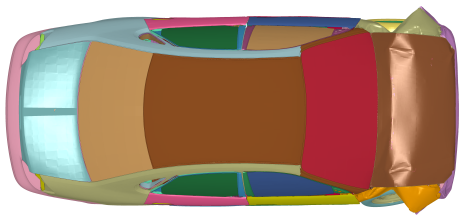
POD	1 cluster	17 clusters			50 clusters		100 clusters		239 clusters		
	-	*k*-means	spectral	dbased	*k*-means	spectral	*k*-means	spectral	*k*-means	spectral	parts
1	1.14e-02	8.03e-03	**7.82e-03**	9.07e-03	6.55e-03	**6.38e-03**	5.92e-03	**5.90e-03**	**5.09e-03**	5.49e-03	5.77e-03
2	9.65e-03	6.49e-03	**6.15e-03**	7.24e-03	4.54e-03	**4.19e-03**	**3.60e-03**	3.65e-03	**2.66e-03**	3.27e-03	3.35e-03
3	7.57e-03	4.94e-03	**4.86e-03**	5.52e-03	3.60e-03	**3.10e-03**	**2.61e-03**	2.62e-03	**1.68e-03**	2.18e-03	2.24e-03
4	6.89e-03	**4.16e-03**	4.18e-03	4.70e-03	2.97e-03	**2.52e-03**	**2.06e-03**	2.08e-03	**1.19e-03**	1.64e-03	1.64e-03
5	6.01e-03	3.53e-03	**3.31e-03**	4.24e-03	2.50e-03	**2.16e-03**	**1.69e-03**	1.73e-03	**9.25e-04**	1.35e-03	1.28e-03
6	5.53e-03	3.02e-03	**2.94e-03**	3.53e-03	2.20e-03	**1.88e-03**	**1.47e-03**	1.50e-03	**7.58e-04**	1.14e-03	1.01e-03
7	5.28e-03	2.72e-03	**2.60e-03**	3.12e-03	1.94e-03	**1.68e-03**	**1.29e-03**	1.33e-03	**6.50e-04**	9.89e-04	8.74e-04
8	4.88e-03	2.51e-03	**2.42e-03**	2.85e-03	1.76e-03	**1.50e-03**	**1.16e-03**	1.18e-03	**5.64e-04**	8.81e-04	7.87e-04
9	4.52e-03	2.31e-03	**2.22e-03**	2.57e-03	1.60e-03	**1.36e-03**	**1.05e-03**	1.07e-03	**5.00e-04**	7.91e-04	7.03e-04
10	4.22e-03	2.14e-03	**2.08e-03**	2.45e-03	1.45e-03	**1.25e-03**	**9.66e-04**	9.88e-04	**4.45e-04**	7.20e-04	6.43e-04
20	2.65e-03	**1.28e-03**	1.35e-03	1.41e-03	8.35e-04	**7.28e-04**	**5.39e-04**	5.55e-04	**2.21e-04**	3.89e-04	3.80e-04
25	2.40e-03	**1.12e-03**	1.16e-03	1.22e-03	7.02e-04	**6.11e-04**	**4.50e-04**	4.72e-04	**1.78e-04**	3.12e-04	3.31e-04
30	2.13e-03	**1.00e-03**	1.04e-03	1.09e-03	6.17e-04	**5.40e-04**	**3.88e-04**	4.12e-04	**1.47e-04**	2.65e-04	2.95e-04
40	1.86e-03	**8.36e-04**	8.99e-04	9.04e-04	5.01e-04	**4.49e-04**	**3.06e-04**	3.36e-04	**1.07e-04**	2.04e-04	2.45e-04
50	1.69e-03	**7.41e-04**	8.01e-04	7.97e-04	4.31e-04	**3.87e-04**	**2.56e-04**	2.81e-04	**8.38e-05**	1.64e-04	2.17e-04
70	1.49e-03	**6.19e-04**	6.79e-04	6.60e-04	3.46e-04	**3.12e-04**	**1.96e-04**	2.20e-04	**5.66e-05**	1.20e-04	1.83e-04
90	1.36e-03	**5.56e-04**	6.16e-04	5.89e-04	2.99e-04	**2.68e-04**	**1.64e-04**	1.86e-04	**4.07e-05**	9.54e-05	1.61e-04
100	1.32e-03	**5.36e-04**	5.94e-04	5.62e-04	2.83e-04	**2.53e-04**	**1.53e-04**	1.74e-04	**3.56e-05**	8.63e-05	1.53e-04

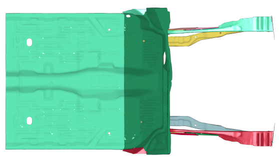
Table 13: Reconstruction error of the displacements of the whole truck with n POD modes: To cluster first and to reconstruct afterward is always better than to reconstruct the truck at once. The clustering gives also a better partition of the nodes with respect to the reconstruction error than the parts. Spectral clustering seems to work well for a small number of clusters, and k-means performs best for a large number of clusters.


(a) t_1, Taurus, side view


(b) t_{152}, Taurus, side view


(c) t_1, Taurus, aerial view


(d) t_{152}, Taurus, aerial view


(e) t_1, 19 important parts, aerial view

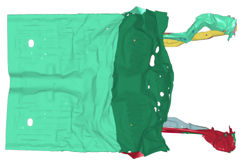

(f) t_{152}, 19 important parts, aerial view

Figure 61: The Ford Taurus before and after the crash.

	1 cluster	19 clusters			22 clusters		
n	-	k-means	spectral	parts	k-means	spectral	dbased
1	1.33e-06	1.03e-06	1.08e-06	**8.23e-07**	1.03e-06	1.07e-06	**8.78e-07**
2	7.46e-07	5.79e-07	6.83e-07	**4.89e-07**	5.70e-07	6.80e-07	**4.83e-07**
3	5.67e-07	3.89e-07	5.04e-07	**3.16e-07**	3.81e-07	5.01e-07	**3.23e-07**
4	4.83e-07	2.88e-07	4.02e-07	**2.44e-07**	2.83e-07	4.00e-07	**2.25e-07**
5	3.62e-07	2.21e-07	3.22e-07	**1.77e-07**	2.16e-07	3.21e-07	**1.66e-07**
6	3.36e-07	1.66e-07	2.69e-07	**1.36e-07**	1.64e-07	2.64e-07	**1.35e-07**
7	2.89e-07	1.34e-07	2.42e-07	**1.12e-07**	1.31e-07	2.34e-07	**1.10e-07**
8	2.75e-07	1.12e-07	2.04e-07	**9.31e-08**	1.10e-07	2.02e-07	**9.89e-08**
9	2.43e-07	9.37e-08	1.75e-07	**8.04e-08**	9.29e-08	1.75e-07	**8.82e-08**

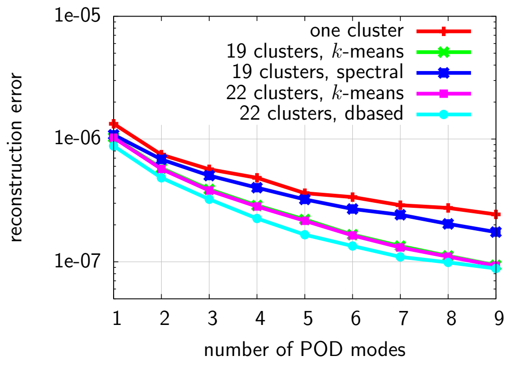
Table 14: Reconstruction error of the displacements of 19 parts of the Ford Taurus with n POD modes: For 19 clusters, the parts are the best clustering of the nodes. However, for 22 clusters, the number predicted by our density-based method, the density-based method performs best.

gives distinctly better results than just reconstructing all 19 parts at once. Fig. 62a also shows that the cluster assignment obtained with the density-based clustering leads to the lowest reconstruction error.

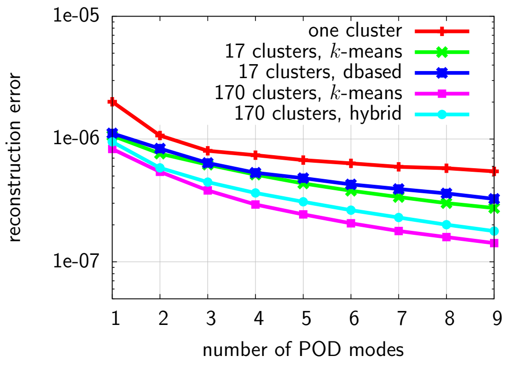
Clustering the whole Ford Taurus We now cluster all $874,728$ nodes of the Ford Taurus. It is not reasonable to cluster all nodes in one step because the data exhibits many overlapping effects. Therefore, we introduce a two-step hybrid clustering scheme where we merge density-based clustering with spectral clustering.

In the first step, the nodes of the Taurus are grouped into only a few clusters with the density-based method. As we have discussed in Sec. 10.2, the density-based clustering is well suited for large data sets. For the Taurus, the method predicts only 17 clusters. Hence, the method predicts fewer clusters for the whole car than for the 19 parts. This is not unusual because it only means that this is a distinctly rougher clustering than we have obtained for the 19 parts. Therefore, in the second step, each of these 17 clusters is further divided into 10 clusters by the spectral clustering method with our out-of-sample extension. In total, we obtain 170 clusters. Again, all parameters are selected with respect to the exponential density measure.

We report the reconstruction errors of the Taurus data for several cluster assignments in Tab. 15. The results for the hybrid method are slightly worse than the ones for the k-means method. However, we also see that first clustering and then reconstructing is again better in terms of accuracy than reconstructing the whole car at once. This is also verified in Fig. 62b. As we increase the number of clusters, the reconstruction error decreases.



(a) 19 parts

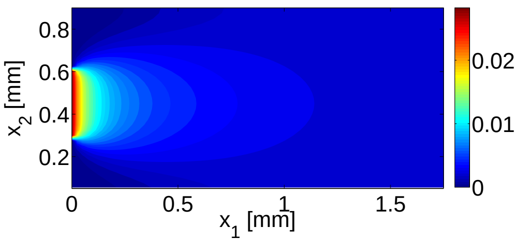


(b) whole car

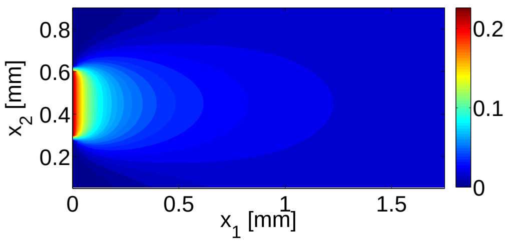
Figure 62: Reconstruction error of the displacements of the Ford Taurus for different clustering methods: In (a) only the 19 important parts are reconstructed where the density-based method gives the best results. In (b) the reconstruction error corresponding to the whole car is shown where k-means performs best.

	1 cluster	17 clusters		170 clusters		741 clusters	
n	-	k-means	dbased	k-means	hybrid	k-means	parts
1	2.01e-06	**1.07e-06**	1.11e-06	**8.30e-07**	9.49e-07	6.73e-07	**6.27e-07**
2	1.07e-06	**7.61e-07**	8.37e-07	**5.41e-07**	5.83e-07	3.82e-07	**3.62e-07**
3	8.02e-07	**6.19e-07**	6.39e-07	**3.82e-07**	4.46e-07	2.49e-07	**2.26e-07**
4	7.39e-07	**5.13e-07**	5.31e-07	**2.93e-07**	3.64e-07	1.80e-07	**1.59e-07**
5	6.74e-07	**4.35e-07**	4.80e-07	**2.43e-07**	3.08e-07	1.42e-07	**1.19e-07**
6	6.34e-07	**3.79e-07**	4.27e-07	**2.06e-07**	2.63e-07	1.16e-07	**9.67e-08**
7	5.94e-07	**3.37e-07**	3.92e-07	**1.78e-07**	2.30e-07	9.72e-08	**7.89e-08**
8	5.78e-07	**3.01e-07**	3.61e-07	**1.59e-07**	2.01e-07	8.32e-08	**6.69e-08**
9	5.45e-07	**2.74e-07**	3.25e-07	**1.42e-07**	1.78e-07	7.30e-08	**5.65e-08**

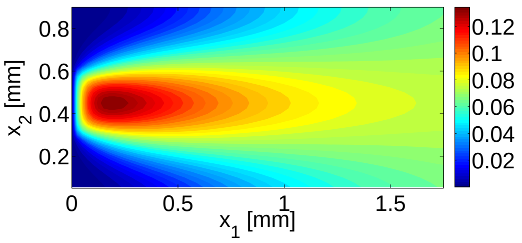
Table 15: Reconstruction error of the displacements of the whole Ford Taurus with n POD modes: In the hybrid method, each cluster obtained with the density-based method is again clustered with spectral clustering. In most cases, k-means is better than the other clustering methods.



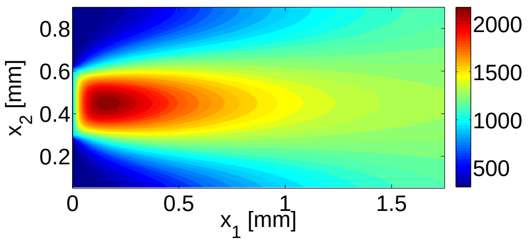
(a) H_2 mass fraction



(b) O_2 mass fraction



(c) H_2O mass fraction



(d) temperature

Figure 63: Shown are the mass fractions of H_2 (fuel), O_2 (oxidizer), and H_2O (product) and the temperature of the high-fidelity model solutions with pre-exponential factor $7.775 \cdot 10^{12}$ and activation energy $5.5 \cdot 10^3$.

12. Case Study: Reacting Flow – Clustering of Snapshots

We consider a model of a steady premixed H_2-Air flame underlying the one-step reaction mechanism

$$2H_2 + O_2 \to 2H_2O$$

where H_2 is the fuel, O_2 the oxidizer, and H_2O the product, see Fig. 63. The evolution of the flame is given by a nonlinear advection-diffusion-reaction equation [35]. It gives rise to a computationally expensive simulation and that is why we want to develop a reduced-order model based on POD. In Sec. 2.3.2 it was already mentioned that POD-Galerkin reduced-order models for nonlinear PDEs include at least one step which scales with the dimension $\mathcal{N}$ of the full model. That is why in [35] the so-called discrete empirical interpolation method (DEIM) is employed. In this section, we replace DEIM with the localized discrete empirical interpolation method (LDEIM). It clusters the (nonlinear) snapshots and computes a local DEIM approximation for each cluster.

We briefly recapitulate DEIM, identify its limitations, and so motivate the development of LDEIM. We discuss LDEIM in a general setting and then introduce a specific variant. Finally, our method is demonstrated on the reacting flow example. Note that we cannot cover all aspects of LDEIM here. We refer to [146] for more LDEIM variants, detailed descriptions of the involved machine learning task, and further applications. The LDEIM method has been developed in close collaboration with Daniel Butnaru [44].

12.1. POD-DEIM-Galerkin Reduced-Order Models

We discuss POD-Galerkin in the context of nonlinear PDEs, illustrate the arising problems, and quickly recapitulate the DEIM method. The section is closed with a paragraph on the limitations of DEIM that motivates the development of the localized DEIM.

POD-Galerkin Consider a discrete system of nonlinear equations

$$\boldsymbol{A}\boldsymbol{u}(\boldsymbol{\mu}) + \boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) = 0 \tag{98}$$

stemming from the discretization of a parametrized nonlinear PDE. It consists of the linear operator $\boldsymbol{A} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ and the nonlinear term $\boldsymbol{F} : \mathbb{R}^{\mathcal{N}} \to \mathbb{R}^{\mathcal{N}}$ with $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) = [F(u_1(\boldsymbol{\mu})), \dots, F(u_{\mathcal{N}}(\boldsymbol{\mu}))]^T$. The Jacobian of the system (98) is given by

$$\boldsymbol{J}(\boldsymbol{u}(\boldsymbol{\mu})) = \boldsymbol{A} + \boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{u}(\boldsymbol{\mu}))\,,$$

where $\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{u}(\boldsymbol{\mu})) = \operatorname{diag}\{F'(u_1(\boldsymbol{\mu})), \dots, F'(u_{\mathcal{N}}(\boldsymbol{\mu}))\} \in \mathbb{R}^{\mathcal{N}\times\mathcal{N}}$ because $\boldsymbol{F}$ is evaluated at $\boldsymbol{u}(\boldsymbol{\mu})$ component-wise. With the POD basis $\boldsymbol{V} \in \mathbb{R}^{\mathcal{N}\times n}$ computed from the snapshots in $\{\boldsymbol{u}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}(\boldsymbol{\mu}_M)\}$ we obtain the POD-Galerkin reduced-order system

$$\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu}) + \boldsymbol{V}^T\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu})) = 0\,, \tag{99}$$

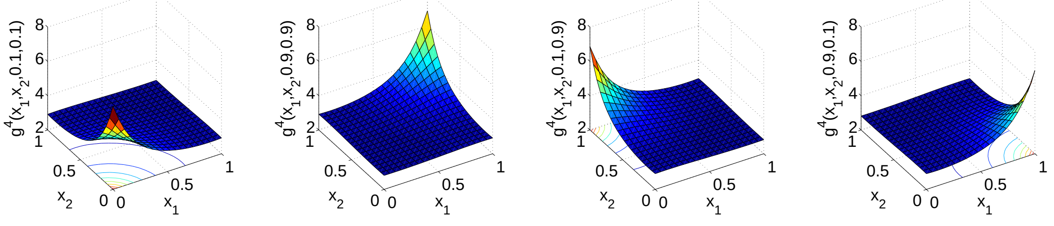
where $\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu})$ replaces the state vector $\boldsymbol{u}(\boldsymbol{\mu})$. In the same way as (99), we can derive the reduced Jacobian.

From (99) it becomes clear that the reduced linear operator $\tilde{\boldsymbol{A}} = \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} \in \mathbb{R}^{n\times n}$ can be pre-computed in the Offline phase but that this is not possible for the nonlinear term. Thus, the nonlinear term $\boldsymbol{F}$ has to be evaluated at all $\mathcal{N}$ components of $\boldsymbol{u}(\boldsymbol{\mu})$ when the system (99) is solved in the Online phase. This quickly becomes the dominating part of the solution time of a POD-Galerkin reduced-order system.

Discrete empirical interpolation method When POD-Galerkin is applied to nonlinear PDEs with a general nonlinearity, it still requires computations that scale with the dimension of the high-fidelity model. Many solutions have been proposed to overcome this drawback of POD-Galerkin, see, e.g., [161, 15, 83, 23, 48]. Here we consider the discrete empirical interpolation method (DEIM) introduced in [51].

Let $\boldsymbol{F}$ be the nonlinear term and $\mathcal{S} = \{\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}_1)), \dots, \boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}_M))\}$ be the set of nonlinear snapshots. A POD basis $\boldsymbol{Q} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_m] \in \mathbb{R}^{\mathcal{N}\times m}$ is computed from the nonlinear snapshots in $\mathcal{S}$. DEIM combines projection and interpolation to approximate $\boldsymbol{F}$ in the space spanned by $\boldsymbol{Q}$, i.e., the coefficients $\boldsymbol{\rho}(\boldsymbol{\mu}) \in \mathbb{R}^m$ have to be determined from the system $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) \approx \boldsymbol{Q}\boldsymbol{\rho}(\boldsymbol{\mu})$. Unfortunately, the system $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) \approx \boldsymbol{Q}\boldsymbol{\rho}(\boldsymbol{\mu})$ is overdetermined and thus DEIM selects only m rows of $\boldsymbol{Q}$ to compute the coefficients. Therefore, a matrix $\boldsymbol{P} = [\boldsymbol{e}_{p_1}, \dots, \boldsymbol{e}_{p_m}] \in \mathbb{R}^{\mathcal{N}\times m}$ is introduced, where $\boldsymbol{e}_{p_i}$ is the p_i-th column of the identity matrix. The interpolation points $p_1, \dots, p_m$ are selected with a greedy algorithm. Assuming $\boldsymbol{P}^T\boldsymbol{Q}$ is nonsingular, the coefficients $\boldsymbol{\rho}(\boldsymbol{\mu})$ can be determined from $\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) = (\boldsymbol{P}^T\boldsymbol{Q})\boldsymbol{\rho}(\boldsymbol{\mu})$ leading to

$$\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) \approx \boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})). \tag{100}$$



(a) $\boldsymbol{\mu} = (0.1, 0.1)$ (b) $\boldsymbol{\mu} = (0.9, 0.9)$ (c) $\boldsymbol{\mu} = (0.1, 0.9)$ (d) $\boldsymbol{\mu} = (0.9, 0.1)$

Figure 64: Depending on the parameter $\boldsymbol{\mu}$, the function g^4 has a sharp peak in one of the four corners of the spatial domain.

The approximation $\boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ interpolates the function $\boldsymbol{F}$ at the interpolation indices $p_1, \dots, p_m$ given in $\boldsymbol{P}$ because

$$\boldsymbol{P}^T(\boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))) = (\boldsymbol{P}^T\boldsymbol{Q})(\boldsymbol{P}^T\boldsymbol{Q})^{-1}\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu})) = \boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))\,.$$

Thus, we can speak of a DEIM interpolant which we denote with a tuple $(\boldsymbol{Q}, \boldsymbol{P})$. Overall, we obtain the POD-DEIM-Galerkin reduced-order system

$$\underbrace{\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}}_{n\times n}\tilde{\boldsymbol{u}}(\boldsymbol{\mu}) + \underbrace{\boldsymbol{V}^T\boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}}_{n\times m}\boldsymbol{P}^T\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu})) = 0\,, \tag{101}$$

and the reduced Jacobian

$$\tilde{\boldsymbol{J}}_{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\boldsymbol{\mu})) = \underbrace{\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}}_{n\times n} + \underbrace{\boldsymbol{V}^T\boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}}_{n\times m}\underbrace{\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{P}^T\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu}))}_{m\times m}\underbrace{\boldsymbol{P}^T\boldsymbol{V}}_{m\times n}\,. \tag{102}$$

With POD-DEIM-Galerkin the nonlinear term $\boldsymbol{F}$ has to be evaluated at m components only. Just as $n \ll \mathcal{N}$ for POD we assume $m \ll \mathcal{N}$ for DEIM and thus expect savings in computational costs. We refer to [51] for more details on DEIM.

Limitations of POD-DEIM-Galerkin reduced-order models DEIM approximates the nonlinear term $\boldsymbol{F}$ in the subspace spanned by $\boldsymbol{Q} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_m]$. If the behavior of $\boldsymbol{F}$ changes heavily when evaluated at different state vectors, the number m of basis vectors has to be large to accurately approximate $\boldsymbol{F}$ in all situations. We demonstrate this on an interpolation example.

Consider the spatial domain $\Omega = [0,1]^2$ and the parameter domain $\mathcal{D} = [0,1]^2$. We define the function $g^1 : \Omega \times \mathcal{D} \to \mathbb{R}$ as

$$g^1(\boldsymbol{x};\boldsymbol{\mu}) = \frac{1}{\sqrt{((1-x_1)-(0.99\cdot\mu_1-1))^2 + ((1-x_2)-(0.99\cdot\mu_2-1))^2 + 0.1^2}} \tag{103}$$

such that the parameter $\boldsymbol{\mu} = (\mu_1, \mu_2)$ controls the gradient of the peak near the corner $(1,1)$ of the spatial domain Ω. Furthermore, let us consider the function

$$\begin{aligned} g^4(\boldsymbol{x};\boldsymbol{\mu}) = {} & g^1(\boldsymbol{x};\boldsymbol{\mu}) + g^1(1-x_1, 1-x_2; 1-\mu_1, 1-\mu_2) \\ & + g^1(1-x_1, x_2; 1-\mu_1, \mu_2) + g^1(x_1, 1-x_2; \mu_1, 1-\mu_2). \end{aligned} \tag{104}$$

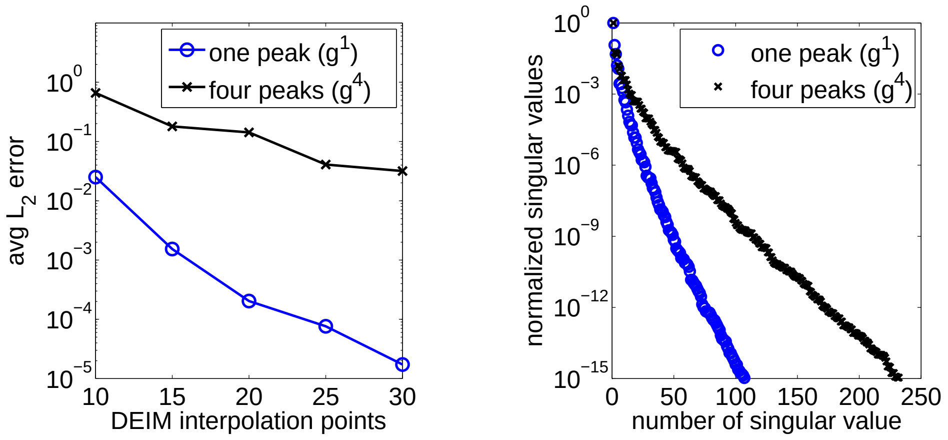


Figure 65: In (a) the average L_2 error is plotted versus the number m of DEIM interpolation points corresponding to the function with one (g^1) and four (g^4) peaks, respectively. The more peaks, the worse the result. This is reflect in the decay of the singular values shown in (b).

The parameter $\boldsymbol{\mu}$ of g^4 controls the gradient of the peaks in all four corners, i.e., depending on the parameter $\boldsymbol{\mu}$, the function g^4 has a sharp peak in one of the four corners of Ω, see Fig. 64. The functions g^1 and g^4 are discretized on a 20×20 equidistant grid in Ω and we randomly sample them on a 25×25 equidistant grid in $\mathcal{D}$. From these 625 snapshots, the DEIM basis $\boldsymbol{Q} = [\boldsymbol{q}_1, \dots, \boldsymbol{q}_m]$ is built. In Fig. 65 the average L_2 error of the approximations over a set of test samples $\{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_{M'}\}$ is shown. The results for g^4 are distinctly worse than for g^1. This is reflected in the slower decay of the singular values of the snapshot matrix corresponding to g^4. Even though g^4 is just a sum of g^1 functions, and, depending on the parameter $\boldsymbol{\mu}$, only one of the summands determines the behavior of g^4, we still get worse results than for g^1. The reason is that the DEIM basis has to be able to approximate all four peaks. It cannot focus on only one (local) peak as is possible when we consider one g^1 function only. This also means that if we choose the parameter $\boldsymbol{\mu} = (0.1, 0.9)$ which leads to only one sharp peak of g^4 near the corner $(0.1, 0.9)$ of Ω, just a few of the DEIM basis vectors are relevant for the approximation and all others are ignored. This is a clear waste of resources and motivates our development of the localized discrete empirical interpolation method.

12.2. Localized Discrete Empirical Interpolation Method

For all parameters $\boldsymbol{\mu}$ in the parameter domain $\mathcal{D}$, DEIM approximates the nonlinear term on the same subspace. Our localized discrete empirical interpolation method (LDEIM) computes several local DEIM interpolants, each of them adapted to only a particular region of characteristic system behavior. The two building blocks of LDEIM are the construction of the local subspace and the selection procedure to choose a good local subspace for the approximation of the nonlinear term.

In this section, we introduce the idea of LDEIM in a general setting, match clustering and classification tasks to its two building blocks, and make some considerations about the error and the computational costs.

LDEIM In the Offline phase, LDEIM clusters the nonlinear snapshots in $\mathcal{S}$ into k subsets $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$. Snapshots should be grouped together if they can be approximated well by the same set of DEIM basis vectors and interpolation points. For each subset $\mathcal{S}_i$ a DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ is constructed. Thus, each interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ is adapted to the snapshots in the corresponding set $\mathcal{S}_i$ only and does not need to capture the behavior of all other snapshots in $\mathcal{S}$. The partition of $\mathcal{S}$ into the subsets $\mathcal{S}_1, \dots, \mathcal{S}_k$ is a clustering task. Input is the set $\mathcal{S}$, output the subsets $\mathcal{S}_1, \dots, \mathcal{S}_k$.

Also in the Offline phase, a classifier $\hat{c} : \mathcal{Z} \to \{1, \dots, k\}$ is constructed. With $\hat{c}$ a local DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ is selected with respect to the indicator $\boldsymbol{z} \in \mathcal{Z}$ in the Online phase. The indicator $\boldsymbol{z}$ has to contain enough information about $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ to select a good local DEIM interpolant and it has to be cheap to compute. Many different indicators are possible. For example, a simple indicator $\boldsymbol{z}$ is the parameter $\boldsymbol{\mu}$. However, we do not pursue this approach further but refer to [146, 44]. Here we derive the indicator $\boldsymbol{z}$ directly from $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ via feature extraction or feature selection, see Sec. 12.3. The construction of the classifier is a supervised learning task. Inputs are the partition $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$ and the indicators $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_M\}$ corresponding to the nonlinear snapshots in $\mathcal{S}$. Output is the classifier $\hat{c}$. We employ nearest neighbor classifiers because they can track curved classification boundaries and are very cheap to evaluate. This is crucial for us because we have to evaluate the classifier during the Online phase.

The POD-LDEIM-Galerkin reduced-order model is

$$\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu}) + \boldsymbol{V}^T\boldsymbol{Q}_{\hat{c}(\cdot)}(\boldsymbol{P}_{\hat{c}(\cdot)}^T\boldsymbol{Q}_{\hat{c}(\cdot)})^{-1}\boldsymbol{P}_{\hat{c}(\cdot)}^T\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu})) = 0 \tag{105}$$

instead of (101), and the reduced Jacobian is now

$$\tilde{\boldsymbol{J}}_{\boldsymbol{F}}(\tilde{\boldsymbol{u}}(\boldsymbol{\mu})) = \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} + \boldsymbol{V}^T\boldsymbol{Q}_{\hat{c}(\cdot)}(\boldsymbol{P}_{\hat{c}(\cdot)}^T\boldsymbol{Q}_{\hat{c}(\cdot)})^{-1}\boldsymbol{J}_{\boldsymbol{F}}(\boldsymbol{P}_{\hat{c}(\cdot)}^T\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu}))\boldsymbol{P}_{\hat{c}(\cdot)}^T\boldsymbol{V}\,. \tag{106}$$

The DEIM basis $\boldsymbol{Q}$ and the matrix $\boldsymbol{P}$ depend through the classifier $\hat{c}$ on the indicator $\boldsymbol{z}$ and thus on the nonlinear term $\boldsymbol{F}$ evaluated at $\boldsymbol{u}(\boldsymbol{\mu})$.

Instead of one $\boldsymbol{V}\boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}$ matrix, we now pre-compute $i = 1, \dots, k$ matrices $\boldsymbol{V}\boldsymbol{Q}_i(\boldsymbol{P}_i^T\boldsymbol{Q}_i)^{-1}$ from which one is picked according to the classifier $\hat{c}$ in the Online phase. The DEIM approximation is fully decoupled from the POD projection of the state. Hence, switching from one DEIM interpolant to another does not influence the POD basis $\boldsymbol{V}$. Note that switching between local reduced-order models is not always that straightforward [96, 56].

Error bounds and computational costs In [51, 188, 52], error bounds for DEIM are introduced. In principle, they can be carried over to LDEIM because after having selected a local DEIM interpolant, we just have a classical DEIM approximation where these error bounds hold.

Compared to DEIM, the Offline phase of LDEIM is more expensive because we first have to cluster the set $\mathcal{S}$, then construct multiple DEIM interpolants, and finally train the classifier $\hat{c}$. However, in the context of model reduction, we always assume we can cope with an expensive Offline phase in favor of a very cheap Online phase.

In the Online phase, the only additional costs incurred by LDEIM compared to DEIM are the evaluation costs of the classifier $\hat{c}$. To evaluate $\hat{c}$, the indicator $\boldsymbol{z}$ has to be computed first. We show in the next section that $\boldsymbol{z}$ can be computed without additional evaluations of the nonlinear term $\boldsymbol{F}$. The evaluation of $\hat{c}$ itself is very cheap because we have a nearest neighbor classifier. The costs neither depend on the number of snapshots M nor on the number of clusters k.

12.3. Feature Extraction for State-Based LDEIM

In many localization approaches in the context of model reduction, the local reduced-order model is selected with respect to the parameter or time step [62, 60, 61, 56, 96]. In terms of the previous section, this would mean to set the indicator $\boldsymbol{z}$ to the parameter $\boldsymbol{\mu}$ and to construct a classifier $\hat{c} : \mathcal{D} \to \{1, \dots, k\}$ with domain $\mathcal{D}$. In contrast, our method computes the indicator as a low-dimensional representation of $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ with feature selection. Thus, the indicator is directly derived from $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ without the detour around the parameter $\boldsymbol{\mu}$. To clearly distinguish the approach presented here from the two LDEIM variants which we have developed in [146, 44], we might sometimes call it state-based LDEIM.

First, the advantages of our state-based LDEIM compared to other localization approaches with a selection procedure based on the parameter are discussed. Then, a low-dimensional representation of $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ is developed.

Limitations of parameter-based selection procedures In Sec. 12.2 we have seen that we need a classifier $\hat{c} : \mathcal{Z} \to \{1, \dots, k\}$ to determine a local DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ in the Online phase of LDEIM. We mentioned the common choice, where the local interpolant or reduced-order model is selected with respect to the parameter $\boldsymbol{\mu}$ of the system. In terms of LDEIM, this would mean the indicator $\boldsymbol{z}$ of $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ is the parameter $\boldsymbol{\mu}$ and the domain $\mathcal{Z}$ of the classifier $\hat{c}$ is set to the parameter domain $\mathcal{D}$. We do not want to discuss such an approach with all its details here but we want to show why setting $\boldsymbol{z} = \boldsymbol{\mu}$ might not always be a good choice.

When we employ DEIM or LDEIM, we have to deal with a system of nonlinear equations. Such nonlinear systems are usually solved with the Newton method. A parameter-based selection procedure picks a local DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ depending on the parameter $\boldsymbol{\mu}$ before the first Newton iteration starts. Since $\boldsymbol{z} = \boldsymbol{\mu}$ is constant during all forthcoming iterations, the local DEIM interpolant is never switched. In contrast, in case of the state-based LDEIM, the indicator $\boldsymbol{z}$ is computed from $\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}))$ which constantly changes during Newton iterations. Thus, even though the parameter $\boldsymbol{\mu}$ is fixed, the indicator $\boldsymbol{z}$ is updated. Therefore, if the classifier $\hat{c}$ is evaluated after each

Newton iteration, we might get different local DEIM approximations in each iteration. This means that the state-based LDEIM can switch the local interpolant during Newton iterations whereas parameter-based procedures keep one local DEIM interpolant fixed until convergence.

Low-dimensional representation Suppose our discrete system of nonlinear equations (105) is solved with the Newton method and $\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})$ is the reduced state vector after the j-th Newton iteration. Our task is to train the classifier $\hat{c}$ on points $\{\boldsymbol{z}_1, \dots, \boldsymbol{z}_M\}$ corresponding to the snapshots in $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_k$ such that we can decide with $\boldsymbol{z}^j$ corresponding to $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ to which subset $\mathcal{S}_1, \dots, \mathcal{S}_k$ the vector $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ belongs. This gives us the index of the local DEIM interpolant for the DEIM approximation in the $j+1$-th Newton iteration.

Of course, we could simply train a classifier $\hat{c} : \mathbb{R}^{\mathcal{N}} \to \{1, \dots, k\}$ on the snapshots $\mathcal{S}$ and evaluate it at $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$. This would certainly work, but it is too expensive. We would have to evaluate the nonlinear term $\boldsymbol{F}$ at all $\mathcal{N}$ components. Hence, we are looking for a more cost-efficient representation of $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$. Consider the map $\tilde{\boldsymbol{F}} : \mathbb{R}^n \to \mathbb{R}^{\tilde{m}}$ where $\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ behaves similar to $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}(\boldsymbol{\mu}))$ but with $\tilde{m} \ll \mathcal{N}$. The function $\tilde{\boldsymbol{F}}$ evaluated at the reduced state vector becomes our indicator, i.e., $\boldsymbol{z}^j = \tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$. With $\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ we represent the high-dimensional vector $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ in a low-dimensional space $\mathbb{R}^{\tilde{m}}$ but $\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ still has to contain the relevant information to classify the vector $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ correctly. It is important that the purpose of $\tilde{\boldsymbol{F}}(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ is not to approximate $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ but only to indicate which local DEIM interpolant to choose. We have to be able to rapidly compute the low-dimensional representation $\boldsymbol{z}^j$ of $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ because the computation is performed during the Online phase, maybe even in each Newton iteration.

Recently, many linear and nonlinear dimensionality reduction methods have been presented, see the survey [125]. In principle, all of them could be employed to compute the indicator. However, we employ two maps $\tilde{\boldsymbol{F}}$ based on DEIM. Let $(\boldsymbol{Q}_g, \boldsymbol{P}_g)$ be the (global) DEIM approximation computed from the set of nonlinear snapshots $\mathcal{S}$ with $\tilde{m}$ DEIM basis vectors and interpolation points. What we call the DEIM-based feature extraction is defined as

$$\tilde{\boldsymbol{F}}_D(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})) = (\boldsymbol{P}_g^T \boldsymbol{Q}_g)^{-1} \boldsymbol{P}_g^T \boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})) \tag{107}$$

and the point-based feature extraction as

$$\tilde{\boldsymbol{F}}_P(\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})) = \boldsymbol{P}_g^T \boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))\,. \tag{108}$$

The DEIM-based feature extraction (108) maps the nonlinear term evaluated at the state vector $\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})$ at the $\tilde{m}$ coefficients of its DEIM representation. The coefficients are a good representation of $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ because of the POD basis underlying the DEIM basis vectors. The point-based feature extraction (108) is motivated by the DEIM greedy algorithm. This algorithm selects those components of $\boldsymbol{F}$ that play a crucial role in the behavior of the nonlinear term. The point-based feature extraction does not require the matrix-vector product with $(\boldsymbol{P}_g^T \boldsymbol{Q}_g)^{-1} \in \mathbb{R}^{\tilde{m} \times \tilde{m}}$ as does the DEIM-based feature extraction.

Efficient computation of the indicator The two low-dimensional representations (107) and (108) lead to a classifier $\hat{c} : \mathbb{R}^{\tilde{m}} \to \{1, \dots, k\}$ with domain $\mathbb{R}^{\tilde{m}}$. To check in the Online phase if switching the local DEIM interpolant after the j-th Newton iteration is appropriate, the indicator $\boldsymbol{z}^j$ is computed either with (107) or (108) and then the classifier $\hat{c}$ is evaluated at $\boldsymbol{z}^j$. In contrast to parameter-based selection procedures where the indicator is simply the parameter, the evaluation of the map (107) or (108) introduces additional costs. Since $\tilde{m}$ is usually even smaller than the number m of DEIM interpolation points, we ignore the costs of the matrix-vector product with $(\boldsymbol{P}_g^T\boldsymbol{Q}_g)^{-1} \in \mathbb{R}^{\tilde{m}\times\tilde{m}}$ in case of DEIM-based feature extraction. However, the additional evaluation of $\boldsymbol{F}$ at $\tilde{m}$ might become quite high. Therefore, we do not evaluate $\boldsymbol{F}$ to compute $\boldsymbol{z}^j$ but interpolate it at the required components with the local DEIM interpolant of the j-th Newton iteration. Thus, for each local DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$, we additionally store the matrix

$$\boldsymbol{W}_i^D = (\boldsymbol{P}_g^T\boldsymbol{Q}_g)^{-1}\boldsymbol{P}_g^T\boldsymbol{Q}_i(\boldsymbol{P}_i^T\boldsymbol{Q}_i)^{-1} \in \mathbb{R}^{\tilde{m}\times m} \tag{109}$$

if we employ the DEIM-based feature extraction (107), and the matrix

$$\boldsymbol{W}_i^P = \boldsymbol{P}_g^T\boldsymbol{Q}_i(\boldsymbol{P}_i^T\boldsymbol{Q}_i)^{-1} \in \mathbb{R}^{\tilde{m}\times m} \tag{110}$$

in case of the point-based feature extraction (108). Suppose we have the local DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ in the j-th Newton iteration. Then we store the vector

$$\tilde{\boldsymbol{f}}^j = \boldsymbol{P}_i\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))\,. \tag{111}$$

This vector is used for the DEIM approximation in the j-th Newton iteration but it is also used to approximate $\boldsymbol{F}$ at the $\tilde{m}$ components required by the feature extraction to compute $\boldsymbol{z}^j$. Thus, the indicator $\boldsymbol{z}^j$ is computed by multiplying the vector $\tilde{\boldsymbol{f}}^j$ with the matrix (109) or (110), respectively.

We emphasize that this allows us to compute the indicator without additional evaluations of the nonlinear term $\boldsymbol{F}$. The approximation error incurred by this interpolation does not effect the DEIM approximation in the $j + 1$-th Newton iteration. It only influences the selection procedure. However, we will show in Sec. 12.5 that this leads to insignificant differences.

12.4. Offline and Online Procedure of LDEIM

The computational procedure of LDEIM in general and of the state-based LDEIM in particular is decomposed into an Offline phase where the local DEIM interpolants are constructed and into an Online phase where a local interpolant is selected depending on the indicator. In this section, we discuss the two phases in detail.

Offline phase In the first step of the Offline phase, the local DEIM interpolants are constructed, and in the second step, the classifier $\hat{c}$ is trained on the clustered snapshot data.

In Alg. 7 the clustering procedure is summarized. Inputs are the number m of local DEIM basis vectors and interpolation points, the dimension $\tilde{m}$ of the low-dimensional

Algorithm 7 Construction procedure of LDEIM

```
1: procedure conLDEIM($m$, $\tilde{m}$, $k$, $\mathcal{S}$, $\tilde{\boldsymbol{F}}$, $\boldsymbol{V}$)
2:     $\tilde{\mathcal{S}} \leftarrow \{\tilde{\boldsymbol{F}}(\boldsymbol{V}^T\boldsymbol{u}(\boldsymbol{\mu}_1)), \dots, \tilde{\boldsymbol{F}}(\boldsymbol{V}^T\boldsymbol{u}(\boldsymbol{\mu}_M))\}$
3:     $(\tilde{\mathcal{S}}_1, \dots, \tilde{\mathcal{S}}_k) \leftarrow k$-means$(\tilde{\mathcal{S}}, k, \|\cdot\|_2)$
4:     $(\boldsymbol{Q}_g, \boldsymbol{P}_g) \leftarrow \mathrm{DEIM}(\mathcal{S}, \tilde{m})$
5:     $l \leftarrow$ empty list
6:     for $i = 1 : k$ do
7:         $\mathcal{S}_i = \{\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}_j)) \,|\, \tilde{\boldsymbol{F}}(\boldsymbol{V}\boldsymbol{u}(\boldsymbol{\mu}_j)) \in \tilde{\mathcal{S}}_i\}$
8:         $(\boldsymbol{U}_i, \boldsymbol{P}_i) \leftarrow \mathrm{DEIM}(\mathcal{S}_i, m)$
9:         $\boldsymbol{W}_i \leftarrow$ depending on $\tilde{\boldsymbol{F}}$ store either matrix (109) or (110)
10:        $l \leftarrow$ append $((\boldsymbol{U}_i, \boldsymbol{P}_i), \boldsymbol{W}_i)$ to list $l$
11:    end for
12:    $l \leftarrow$ append $(\boldsymbol{U}_g, \boldsymbol{P}_g)$ to list $l$
13:    $\hat{c} \leftarrow$ train classifier on $\tilde{\mathcal{S}}_1 \times \{1\} \cup \dots \cup \tilde{\mathcal{S}}_k \times \{k\}$
14: return $l$
15: end procedure
```

representation, the number of clusters k, the set of nonlinear snapshots $\mathcal{S}$, the feature extraction $\tilde{\boldsymbol{F}}$, and the POD basis $\boldsymbol{V}$. The snapshots in $\mathcal{S}$ are transformed into their low-dimensional representation and are then stored in $\tilde{\mathcal{S}}$. The data in $\tilde{\mathcal{S}}$ is then clustered with k-means with respect to the Euclidean norm. We emphasize that it is distinctly cheaper to cluster the low-dimensional points in $\tilde{\mathcal{S}} \subset \mathbb{R}^{\tilde{m}}$ than the high-dimensional snapshots in $\mathcal{S} \subset \mathbb{R}^{\mathcal{N}}$ as is done in [13]. The result is a partition of $\tilde{\mathcal{S}} = \tilde{\mathcal{S}}_1 \uplus \dots \uplus \tilde{\mathcal{S}}_k$ and $\mathcal{S} = \mathcal{S}_1 \uplus \dots \mathcal{S}_k$ into k subsets. For each subset $\mathcal{S}_i$, a local DEIM interpolant $(\boldsymbol{Q}_i, \boldsymbol{P}_i)$ is built and the matrix $\boldsymbol{W}_i$ is constructed. The local interpolants $(\boldsymbol{Q}_1, \boldsymbol{P}_1), \dots, (\boldsymbol{Q}_k, \boldsymbol{P}_k)$, the corresponding matrices $\boldsymbol{W}_1, \dots, \boldsymbol{W}_k$, and the global DEIM interpolant $(\boldsymbol{Q}_g, \boldsymbol{P}_g)$ are stored in the list l. Finally, a nearest neighbor classifier $\hat{c}$ is trained on the set

$$\tilde{\mathcal{S}}_1 \times \{1\} \cup \dots \cup \tilde{\mathcal{S}}_k \times \{k\} \subset \mathbb{R}^{\tilde{m}} \times \{1, \dots, k\}\,.$$

The k-means clustering method (Lloyd's algorithm) is initialized with a random cluster assignment. Because k-means finds only local optima, the initial clustering influences the final cluster assignment. To cope with this nondeterministic behavior, we split of a small test data set of $\tilde{\mathcal{S}}$ and repeat the whole construction procedure in Alg. 7 several times. In the end, the clustering leading to the smallest DEIM residual on the test data is chosen.

Online phase During the Online phase, a local DEIM interpolant for the $j+1$-th Newton iteration is selected with respect to the indicator $\boldsymbol{z}^j$ computed from $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$.

Consider Alg. 8. Input are the POD basis $\boldsymbol{V}$, the nonlinear term $\boldsymbol{F}$, the list l obtained with Alg. 7, the classifier $\hat{c}$, the index $i^j \in \{1, \dots, k\}$ of the local DEIM interpolant in the j-th Newton iteration, the state vector $\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})$, and the vector $\tilde{\boldsymbol{f}}^j$ containing the components of $\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$ corresponding to the interpolation points defined by $\boldsymbol{P}_{i^j}$,

Algorithm 8 Selection procedure of LDEIM

```
1: procedure selLDEIM($\boldsymbol{V}$, $\boldsymbol{F}$, $l$, $\hat{c}$, $i^j$, $\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu})$, $\tilde{\boldsymbol{f}}^j$)
2:     $\boldsymbol{z}^j \leftarrow \boldsymbol{W}_{i^j}\tilde{\boldsymbol{f}}^j$
3:     $i^{j+1} \leftarrow \hat{c}(\boldsymbol{z}^j)$
4:     $\tilde{\boldsymbol{f}}^{j+1} \leftarrow \boldsymbol{P}_{i^j}\boldsymbol{F}(\boldsymbol{V}\tilde{\boldsymbol{u}}^j(\boldsymbol{\mu}))$
5: return $(i^{j+1}, \tilde{\boldsymbol{f}}^{j+1})$
6: end procedure
```

see (111). With the matrix $\boldsymbol{W}_{i^j}$ contained in the list l, the indicator $\boldsymbol{z}^j$ is computed with the vector $\tilde{\boldsymbol{f}}^j$. With the indicator $\boldsymbol{z}^j$, the classifier $\hat{c}$ can be evaluated to get the index i^{j+1} of the local DEIM interpolant for the $j+1$-th Newton iteration. Finally, the nonlinear term is evaluated at the interpolation points defined by $\boldsymbol{P}_{i^{j+1}}$ and stored in $\tilde{\boldsymbol{f}}^{j+1}$. Returned are the index i^{j+1} and the vector $\tilde{\boldsymbol{f}}^{j+1}$.

We make two remarks on the Online phase. First, if the Newton method has not converged after a couple of iterations, we switch to the classical DEIM approximation with m basis vectors and m interpolation points. This is only necessary in exceptional cases, e.g., when we constantly switch back and forth between two local interpolants. Second, to ensure a smooth transition when we switch from one local interpolant to another, we oversample the local DEIM interpolants by employing the interpolation points corresponding to both clusters [196]. Note that we oversample only in the first iteration after a switch.

12.5. Simulation of an H_2-Air Flame

In this section, we finally consider the reacting flow example of an H_2-Air flame. We briefly describe the problem, discuss the POD-DEIM-Galerkin reduced-order model presented in [35], and then derive the LDEIM reduced-order model. We close the section with numerical results.

The H_2-Air flame model The chemistry behind the reacting flow is modeled by a one-step reaction

$$2H_2 + O_2 \rightarrow 2H_2O,$$

where H_2 is the fuel, O_2 the oxidizer, and H_2O the product, see Fig. 63. If we assume a constant and uniform pressure, a constant and divergence-free velocity field, as well as constant, equal, and uniform molecular diffusivities for all species and temperature, the evolution of the flame in a domain $\Omega \subset \mathbb{R}^2$ with boundary Γ is described by the nonlinear advection-diffusion-reaction equation

$$\kappa\Delta\boldsymbol{u} - v\nabla\boldsymbol{u} + \boldsymbol{f}(\boldsymbol{u}, \boldsymbol{\mu}) = 0 \quad \text{in } \Omega \tag{112}$$

where $\boldsymbol{u} = [u_{H_2}, u_{O_2}, u_{H_2O}, T]^T$ contains the mass fractions of the species H_2, O_2, H_2O and the temperature T, and $\boldsymbol{f}$ is the nonlinear source term. The geometry of Ω is shown in Fig. 66. The constants $\kappa = 2.0\text{cm}^2/\text{sec}$ and $v = 50\text{cm/sec}$ are the molecular

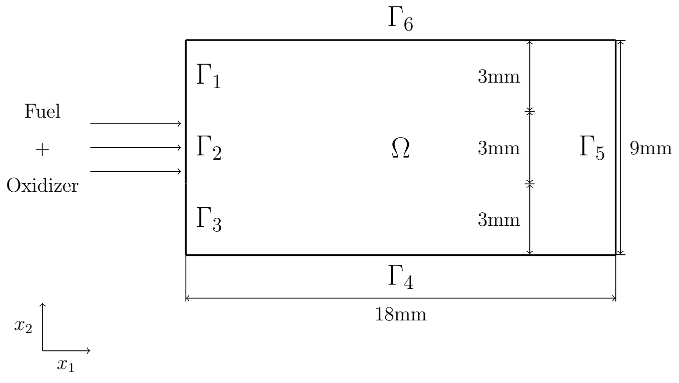


Figure 66: The geometry of the domain for the H_2-Air flame simulation [35].

diffusivity and the velocity of the velocity field in x_1 direction, respectively. With the notation of Fig. 66, we have homogeneous Dirichlet boundary conditions on the mass fractions on Γ_1 and Γ_3, and homogeneous Neumann conditions on temperature and mass fractions on Γ_4, Γ_5, and Γ_6. We have Dirichlet conditions on Γ_2 with $u_{\mathrm{H}_2} = 0.0282, u_{\mathrm{O}_2} = 0.2259, u_{\mathrm{H}_2\mathrm{O}} = 0, u_T = 950\mathrm{K}$, and on Γ_1, Γ_3 with $u_T = 300\mathrm{K}$.

The nonlinear term $\boldsymbol{f} = [f_{\mathrm{H}_2}, f_{\mathrm{O}_2}, f_{\mathrm{H}_2\mathrm{O}}, f_T]^T$ has the components

$$\begin{aligned} f_i(\boldsymbol{u}, \boldsymbol{\mu}) &= -\nu_i \left(\frac{w_i}{\rho}\right)\left(\frac{\rho u_{\mathrm{H}_2}}{w_{\mathrm{H}_2}}\right)^2 \left(\frac{\rho u_{\mathrm{O}_2}}{w_{\mathrm{O}_2}}\right) \mu_1 \exp\left(-\frac{\mu_2}{\iota u_T}\right), \quad i = \mathrm{H}_2, \mathrm{O}_2, \mathrm{H}_2\mathrm{O} \\ f_T(\boldsymbol{u}, \boldsymbol{\mu}) &= f_{\mathrm{H}_2\mathrm{O}}(\boldsymbol{u}, \boldsymbol{\mu})\eta \end{aligned} \tag{113}$$

where μ_1 is the pre-exponential factor, μ_2 is the activation energy, ν_i is the respective stoichiometric coefficient, $w_{\mathrm{H}_2}, w_{\mathrm{O}_2}$, and $w_{\mathrm{H}_2\mathrm{O}}$ are the molecular weights $2.016, 31.9$, and 18gr/mol, respectively. The constant $\rho = 1.39 \cdot 10^{-3}\mathrm{gr/cm}^3$ is the density of the mixture, $\iota = 8.314472$ J/(mol K) is the universal gas constant, and $\eta = 9,800\mathrm{K}$ is the heat of reaction. The parameter $\boldsymbol{\mu} = (\mu_1, \mu_2)$ can vary within the parameter domain $\mathcal{D} = [5.5 \cdot 10^{11}, 1.5 \cdot 10^3] \times [1.5 \cdot 10^{13}, 9.5 \cdot 10^3] \subset \mathbb{R}^2$. All constants are set to the same values as in [35], only the parameter domain $\mathcal{D}$ was slightly extended to allow a larger variation of the solutions.

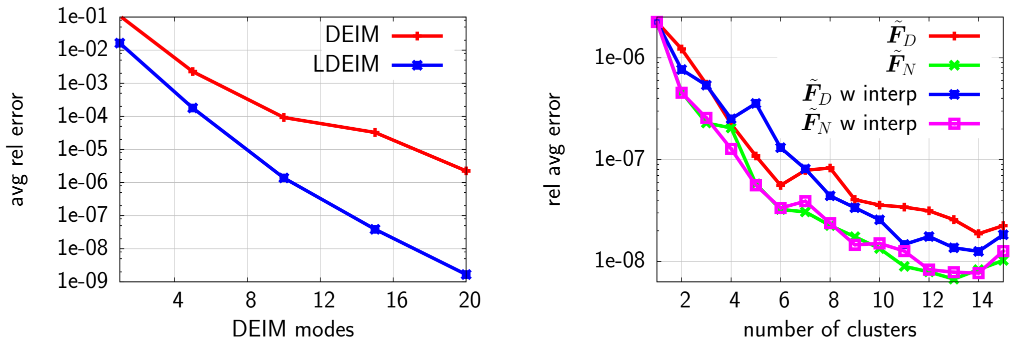
Equation (112) is discretized with finite differences on an equidistant 73×37, leading to the system of nonlinear algebraic equations

$$\boldsymbol{A}\boldsymbol{u} + \boldsymbol{F}(\boldsymbol{u}, \boldsymbol{\mu}) = 0 \tag{114}$$

with $\mathcal{N} = 10,804$ degrees of freedom. The matrix $\boldsymbol{A} \in \mathbb{R}^{\mathcal{N} \times \mathcal{N}}$ corresponds to the linear operators and the function $\boldsymbol{F} : \mathbb{R}^{\mathcal{N}} \to \mathbb{R}^{\mathcal{N}}$ to the nonlinear function $\boldsymbol{f}$ in (112). The state vector $\boldsymbol{u} = [\boldsymbol{u}_{\mathrm{H}_2}, \boldsymbol{u}_{\mathrm{O}_2}, \boldsymbol{u}_{\mathrm{H}_2\mathrm{O}}, \boldsymbol{u}_T]^T \in \mathbb{R}^{\mathcal{N}}$ contains the mass fractions of the species and the temperature at the grid points. The nonlinear system (114) is solved with the Newton method.

Reduced-order model for reacting flow In [35] the POD-DEIM-Galerkin reduced-order model

$$\boldsymbol{V}^T\boldsymbol{A}\overline{\boldsymbol{u}} + \boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V}\tilde{\boldsymbol{u}} + \boldsymbol{V}^T\boldsymbol{Q}(\boldsymbol{P}^T\boldsymbol{Q})^{-1}\boldsymbol{F}(\boldsymbol{P}^T\overline{\boldsymbol{u}} + \boldsymbol{P}^T\boldsymbol{V}\tilde{\boldsymbol{u}}, \boldsymbol{\mu}) = 0$$



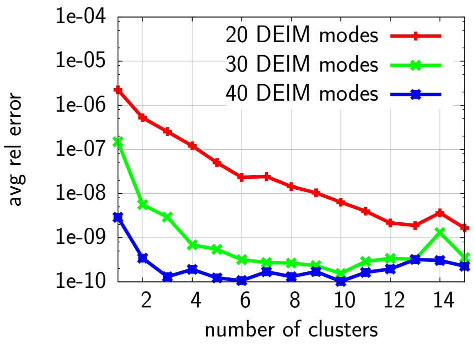
(a) comparison of DEIM and LDEIM (b) effect of feature extraction on accuracy

Figure 67: In (a) the accuracy comparison of DEIM and LDEIM is shown. In (b) the effect of the two feature extraction methods $\tilde{\boldsymbol{F}}_D$ and $\tilde{\boldsymbol{F}}_P$ with and without interpolation (40 POD modes, 20 DEIM modes) is illustrated.

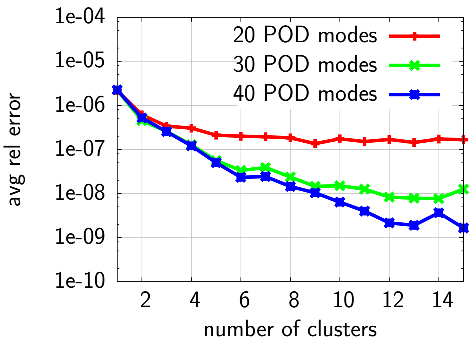
is introduced. The vector $\overline{\boldsymbol{u}} \in \mathbb{R}^{\mathcal{N}}$ is the arithmetic mean of the set of snapshots $\{\boldsymbol{u}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}(\boldsymbol{\mu}_M)\}$, the POD basis $\boldsymbol{V} \in \mathbb{R}^{\mathcal{N} \times n}$ is computed from $\{\boldsymbol{u}(\boldsymbol{\mu}_j) - \overline{\boldsymbol{u}}\}_{j=1}^{M}$. The DEIM interpolant $(\boldsymbol{Q}, \boldsymbol{P})$ is built from the set $\mathcal{S} = \{\boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}_1), \boldsymbol{\mu}_1), \dots, \boldsymbol{F}(\boldsymbol{u}(\boldsymbol{\mu}_M), \boldsymbol{\mu}_M)\}$. The snapshots are computed for the parameters on an equidistant 50×50 grid in $\mathcal{D}$.

Numerical results Let us now employ LDEIM instead of DEIM. We solve the POD-LDEIM-Galerkin system for parameters on the equidistant 24×24 grid in $\mathcal{D}$ and compute the average relative error to the full model solutions. The results are shown in Fig. 67. We have $n = 40$ POD modes, $k = 15$ clusters, and the dimension of the low-dimensional representation is set to $\tilde{m} = 5$. With LDEIM we achieve an about two orders of magnitude better result than with the classical DEIM, see Fig. 67a. In Fig. 67b we compare the two maps $\tilde{\boldsymbol{F}}_D$ and $\tilde{\boldsymbol{F}}_P$ with and without interpolating the required components of $\boldsymbol{F}$ with the matrices (109) and (110), respectively. No large difference between the two feature extraction methods can be seen, and there is no significant loss of accuracy if we interpolate the values of $\boldsymbol{F}$ at the points required by the feature extraction instead of directly evaluating $\boldsymbol{F}$.

In Fig. 68 we show the results of LDEIM with point-based feature extraction with interpolation when we fix the number of POD and DEIM modes, and only vary the number of clusters. Consider for example Fig. 68a where we plot the error curve for 40 POD and 20 DEIM modes. Increasing the number of clusters leads to a higher accuracy. However, if we have 40 instead of 20 DEIM modes, more than three clusters do not lead to an improvement. The reason is that even though the DEIM approximation gets more and more accurate as we increase the clusters, the POD basis is fixed and thus limits the overall accuracy. We want to emphasize that a large number of clusters does not deteriorate the accuracy, as is for example the case in other approaches [146]. Hence, the clustering of the low-dimensional representation leads to a stable cluster assignment here.



(a) 40 POD modes



(b) 20 DEIM modes

Figure 68: The number of POD/DEIM modes is fixed and the number of clusters is varied. Increasing the number of clusters does improve the accuracy up to where POD becomes the limiting factor for the overall result.

Let us finally have a look at the runtimes in Tab. 16. We show the average relative error and the runtimes of the state-based LDEIM with 40 POD and 10 DEIM modes for the two feature extraction methods $\tilde{\boldsymbol{F}}_D$ and $\tilde{\boldsymbol{F}}_P$. The computations were performed five times and the average runtime is reported. All runtimes are normalized with respect to the runtime for only one cluster for the respective feature extraction method without the interpolation procedure described in Sec. 12.3. Table 16 shows clearly that if we increase the number of clusters, the runtime increases only slightly. Notice the worst case reported in Tab. 16 where the runtime for 1 to 100 clusters increases by a factor of 1.5 only. Furthermore, the errors confirm once more that a large number of clusters does not lead to an unstable clustering.

	$\tilde{\boldsymbol{F}}_P$ w/out interp		$\tilde{\boldsymbol{F}}_D$ w/out interp		$\tilde{\boldsymbol{F}}_P$ with interp		$\tilde{\boldsymbol{F}}_D$ with interp	
k	error	time	error	time	error	time	error	time
1	9.26e-05	1.000	9.26e-05	1.000	9.26e-05	1.014	9.26e-05	1.037
10	2.90e-06	1.371	5.53e-06	1.289	2.20e-06	1.179	4.87e-06	1.231
20	9.19e-07	1.194	1.46e-06	1.071	9.96e-07	1.176	3.50e-06	1.102
30	4.92e-07	1.205	9.82e-07	1.343	5.21e-07	1.097	8.40e-07	1.257
40	2.87e-07	1.108	5.35e-07	1.125	3.32e-07	1.214	7.05e-07	1.189
50	2.33e-07	1.198	5.61e-07	1.290	2.06e-07	1.183	6.90e-07	1.161
60	2.04e-07	1.144	3.44e-07	1.143	2.01e-07	1.243	4.06e-07	1.164
70	2.15e-07	1.011	2.25e-07	1.148	1.98e-07	1.113	1.04e-06	1.325
80	4.02e-07	1.078	2.09e-07	1.303	1.58e-07	1.199	1.01e-06	1.384
90	2.03e-07	1.441	2.11e-07	1.238	2.41e-07	1.195	1.37e-06	1.419
100	1.39e-07	1.094	5.37e-07	1.144	1.18e-07	1.181	1.29e-06	1.524

Table 16: State-based LDEIM with 40 POD and 10 DEIM modes for our two feature extraction methods with up to 100 clusters: The runtimes only slightly increase when we increase the number of clusters. The reported errors confirm that the clustering does not get unstable as we increase the number of clusters.

13. Remarks

Common *a posteriori* model order reduction methods build reduced-order models from the set of snapshots $\mathcal{S} = \{\boldsymbol{u}(\boldsymbol{\mu}_1), \dots, \boldsymbol{u}(\boldsymbol{\mu}_M)\}$. In contrast, *post analysis* model order reduction discussed in this part, first analyzes the data in $\mathcal{S}$ with learning methods and then does not construct only one global but multiple local reduced-order models from the pre-processed data. Unsupervised learning methods such as clustering play a crucial role to derive the local reduced-order models.

Two sparse-grid-based clustering methods were introduced. We presented an out-of-sample extension with sparse grids for spectral clustering and a density-based clustering method where the estimated density function is approximated on a sparse grid. All of our clustering methods are able to find clusters with a non-convex shape.

Spectral clustering methods perform well in many situations but the time to solve the underlying eigenproblem usually grows with $\mathcal{O}(M^3)$ in the number of data points M. This makes so-called out-of-sample extensions necessary. These are maps from the high-dimensional ambient space where the data points come from, into a low-dimensional latent space which corresponds to the cluster assignment. We constructed such a map by approximating the eigenfunctions of the Laplace-Beltrami operator with sparse grid functions. These sparse grid functions can be evaluated at any data point to rapidly obtain the cluster assignment of large data sets. Furthermore, if the out-of-sample extension is constructed, the eigenproblem does not depend on the number of data points anymore but only on the number of sparse grid points. We applied spectral clustering with our sparse-grid-based out-of-sample extension to various benchmark examples and demonstrated that it achieves higher accuracies than the widely-used Nyström-based out-of-sample extension. Furthermore, we combined spectral clustering and our out-of-sample extension to a nonlinear dimensionality reduction method and analyzed snapshots of a flow simulation data where two parameters were varied. The embedding of the data reflected the two parameters. Finally, the out-of-sample extension was applied to image segmentation where we have shown that only a few sparse grid points are necessary to obtain a reasonable segmentation of an image.

We continued with our density-based clustering method. It splits the similarity graph constructed from the given data points into components by removing vertices of the graph in low-density regions. These low-density regions are found with the help of an estimated density function discretized on a sparse grid. The method is well-suited for large data sets because the similarity graph can be constructed in parallel and the runtime of the sparse-grid-based density estimation method does only slightly depend on the number of data points. Furthermore, our density-based clustering method allows us to distinguish between weak and strong clusters and it includes an indicator for the number of clusters. We demonstrated the method and its properties on benchmark and real-world examples.

Having introduced these two clustering methods, we considered two *post analysis* model order reductions methods. In the SIMDATA-NL workflow the nodes of large finite element models are clustered and for each cluster a local reduced-order model is computed. In contrast, the localized discrete empirical interpolation method clusters the snapshots to construct local interpolants for the approximation of the nonlinear term in reduced-order models of nonlinear PDEs.

The SIMDATA-NL workflow has been developed to analyze car crash simulation data. Its two core steps are clustering and nonlinear dimensionality reduction. We focused on the clustering step. The purpose of the clustering is twofold. It should give insight into the crash behavior of the car and the cluster assignment is the input for the successive dimensionality reduction step in the workflow. We analyzed a frontal crash of a Chevrolet pick-up truck and of a Ford Taurus. The nodes of the finite element models of the car were clustered with k-means and our two sparse-grid-based clustering methods. We plotted the cluster assignments and observed that our density-based method and spectral clustering with our out-of-sample extension captured all bending patterns during the crash whereas k-means was not able to find these details in all cases. To quantitatively assess the quality of the clustering we built a local POD reduced-order model for each cluster separately and reconstructed the simulation data. First, the results confirmed that our sparse-grid-based methods perform better than k-means in many situations, and, second, they clearly showed that first dividing the model into clusters and then locally reconstructing each cluster on its own gives distinctly better results than to reconstruct the whole car model at once. This supports the SIMDATA-NL workflow where the nonlinear dimensionality reduction methods are applied to each cluster separately.

The localized discrete empirical interpolation method (LDEIM) was developed to approximate nonlinear terms in reduced-order models of nonlinear PDEs. Instead of the nodes of the finite element model as in the SIMDATA-NL workflow, LDEIM clusters the snapshots and constructs a separate interpolant for each cluster in the Offline phase. Each of these local interpolants is adapted to only a certain region in state space and thus to a specific system behavior. Depending on the current state of the system, one of the interpolants is selected for the actual approximation of the nonlinear term in the Online phase. The state of the system is described by an indicator which is obtained from the nonlinear term with feature extraction. The LDEIM method employs clustering to partition the set of snapshots, and classification to choose an appropriate local interpolant. We applied our localized discrete empirical interpolation method to a reacting flow simulation of an H_2-Air flame where LDEIM achieved accuracies about two orders of magnitude better than DEIM.

14. Conclusions and Future Work

We presented a reduced-order model hierarchy that reflects the close relationship of model order reduction and machine learning. Following this hierarchy, we introduced sparse grid learning methods, employed them for model order reduction, and applied the derived methods to various applications. The objective was to develop novel sparse grid learning techniques and to show that combining classical machine learning with model order reduction leads to improved methods with respect to accuracy, runtime, or scope.

Our model reduction hierarchy consists of three levels corresponding to Parts I–III. In Part I the focus was on *a priori* model order reduction where the problem is solved in a space that is tailored to a whole problem class. We considered sparse grid spaces and developed a multigrid method to solve PDEs discretized on sparse grids. Methods for *a posteriori* model order reduction were the topic of Part II. From given data (snapshots), we learned the input-output relationship corresponding to the parameters and the outputs of interest as defined by our problem with regression and classification methods based on sparse grids. The result was a non-intrusive model reduction method which was competitive to the reduced basis method for our examples but which does not require to adapt the underlying solvers. Finally, in Part III we introduced two *post analysis* model reduction methods which first analyze the given data with unsupervised learning methods, e.g., clustering methods, and then build multiple local reduced-order models, each of them tailored to a specific system behavior.

Across Parts I–III we developed supervised and unsupervised sparse grid learning methods for classification, regression, clustering, (nonlinear) dimensionality reduction, and probability density estimation. Besides applications from computational science and engineering, our learning methods were also applied to classical learning problems such as the analysis of simulation data, image segmentation, biochemical screenings, and sampling from estimated density functions.

The main contributions of this thesis can be summarized as follows:

- With the multigrid method developed in Sec. 4 it is computationally feasible to solve the multi-dimensional convection-diffusion equation on sparse grids without additional approximates of, e.g., the bilinear forms of the operators. Furthermore, due to the underlying ANOVA-like decomposition of the solution function, the multigrid algorithm allows to coarsen and refine in each direction independently and still provides a valid system of equations for each grid of the sparse grid hierarchy.

- The probability density estimation method based on sparse grids as introduced in Sec. 7.1 makes processing of large data sets possible. To extend the scope of the density estimation method we presented algorithms to marginalize, conditionalize, and sample sparse grid density functions, and which scale only linearly in the number of data points.

- Based on the sparse grid density estimation method, we developed a probabilistic, generative classification method in Sec. 7.2 with an Offline/Online splitting to significantly reduce the runtime of the prediction for new, previously unseen data.

- Multiple sparse grid classifiers were put together into a team or ensemble in Sec. 7.3. The ensemble of classifiers achieved similar accuracies as an adaptive sparse grid classifier but with a reduced runtime because regular sparse grids were sufficient.

- Two clustering methods based on sparse were introduced in Sections 10.1 and 10.2. Both are non-convex clustering methods allowing arbitrary cluster shapes. In the presented examples, we achieved excellent accuracies with our methods compared to standard clustering approaches. To the author's knowledge, these are the first clustering methods based on sparse grids. This means that now all major tasks in machine learning — classification, regression, clustering, density estimation, and dimensionality reduction — can be performed with sparse grid methods.

- Finally, in Sections 11 and 12, *post analysis* model reduction approaches for the analysis of car crash data and for the approximation of nonlinear functions were introduced. Both are based on clustering methods to adapt the local reduced models to specific system behaviors.

The possibilities for future research in the topics discussed in this thesis are manifold. Maybe one of the first questions that might be posed is how to continue the model reduction hierarchy after *post analysis* model reduction? It is evident that from *a priori* to *post analysis* methods the costs of the Offline phase have dramatically increased. It is not clear if a further increase in Offline costs can be still compensated by even lower Online costs. At least limitations in the scope of such methods are to be expected. Model order reduction approaches that have not been covered here are multi-fidelity approaches where low-, medium-, and high-fidelity models are combined. We also did not cover models that are updated in the Online phase. We refer to, e.g., [137, 5, 30, 63, 183] for more details on these approaches.

We summarize a couple of more concrete topics of future research. These include a sparse grid multigrid method for elliptic PDEs with variable coefficients, i.e., with coefficients that are functions and that do not have tensor product structure. This has been an open problem in the context of sparse grids for a long time now, but no satisfying answer has been found yet [150, 1]. Our multigrid approach based on the ANOVA-like decomposition might be a starting point in that direction. Besides that, it would be interesting to investigate to what extent the classification method based on sparse grid density estimation with the Offline/Online splitting is capable to perform Online learning, i.e., learning from data streams. We have only considered the density-based method for classification. An extension to regression has already been presented in [193]. However, thorough study has not been done yet. Finally, it has only been demonstrated on toy examples so far (Burgers' equation) that the localized discrete empirical interpolation method (LDEIM) is well-suited for time-dependent problems where the trajectories show a cyclic behavior in the state space. A real-world example is required to confirm the advantages of (state-based) LDEIM compared to parameter-based localization approaches.

A. Data Sets

3S [148], three dimensions, three classes
Three-dimensional extension of the two moons data set. Three partial spheres are interwoven.

3Snoise [148], three dimensions, three classes
Similar to the 3S data set but it contains more noise.

Berkeley Segmentation Data set [133]
Contains 300 example pictures for image segmentation.

bupa [19], six dimensions, two classes
The data set describes liver disorders. A widely-used benchmark data set in data mining.

checkerboard [153], two or more dimensions, two classes
Data points generated with 3×3 checker board pattern in two dimensions. Straightforward generalizes to d dimensions possible.

codRNA [176], eight dimensions, two classes
Non-coding RNAs (ncRNA) are transcripts that have function without being translated to protein. This data set is used to a train classifier for computational pre-screening of RNAs.

fluid [89], five dimensions
Each data points corresponds to a solution of the Navier-Stokes equations for the driven cavity example.

iris [69], four dimensions, three classes
A famous benchmark data set for classification and clustering. It describes three species of Iris flowers.

oil flow [27], 12 dimensions, two or three classes
A pipeline usually contains a mixture of oil, water, and gas. The data points reflect measurements taken from a pipeline where the fractions of the three substances correspond to three classes.

old faithful [28], two dimensions, two classes
Contains measurements of the eruption of the Old Faithful geyser.

olive [197], eight dimensions, three classes
Data points correspond to fatty acids of Italian olive oils coming from three different regions of Italy.

sergio [9], five dimensions
The data set is the result of measurements for uncertainty analysis of multidisciplinary systems. Provided by Sergio Amaral.

shuttle [79], nine dimensions, two classes
The data points are the configurations of shuttles during start procedure. Note that this is not the original shuttle data set as can be found in, e.g., the UCI Machine Learning Repository. It is the modified version as described in [79].

svmguide1 [110], four dimensions, two classes
Benchmark data set used for the libsvm library.

Swiss roll [24], three dimensions
It contains the data points of a Swiss roll. It is used in dimensionality reduction to embed the data points in a two-dimensional space.

two moons [74], two dimensions, two classes
A common benchmark data set containing two interwoven moons.

B. Software

ANN A library that contains data structures to compute approximated nearest neighbors.
http://www.cs.umd.edu/˜mount/ANN/

BOOST This C++ template library contains a rich collection of routines covering a broad range of applications. We use the Boost Graph Library.
http://www.boost.org/

GSL The GNU Scientific Library provides routines for many numerical methods.
http://www.gnu.org/software/gsl/

libsvm A fast implementation of support vector machines.
http://www.csie.ntu.edu.tw/˜cjlin/libsvm/

LS-DYNA Fraunhofer SCAI used this software to simulate the car crash tests.
http://www.lstc.com/products/ls-dyna

LS-PrePost Software to visualize and explore simulation data.
http://www.lstc.com/lspp/

MATLAB We use MATLAB and its toolboxes for a variety of tasks.
http://www.mathworks.com/

R The R programming language and the corresponding CRAN software repository is popular in statistics.
http://r-project.org/

SG^{++} The sparse grid library has been developed by Dirk Pflüger [153]. It contains sparse grid routines for data mining tasks (classification based on sparse grid regression) and several numeric methods for solving PDEs.
http://www5.in.tum.de/SGpp

scikit-learn A Python library that implements many machine learning methods from classification to clustering.
http://scikit-learn.org/

References

[1] S. Achatz. *Adaptive finite Dünngitter-Elemente höherer Ordnung für elliptische partielle Differentialgleichungen mit variablen Koeffizienten.* Thesis (Ph.D.), Technische Universität München, 2003.

[2] S. Ackermann, L. Gaul, M. Hanss, and T. Hambrecht. Principal component analysis for detection of globally important input parameters in nonlinear finite element analysis. In *5th Weimar Optimization and Stochastic Days*, 2008.

[3] J. Adorf. Nonlinear clustering on sparse grids. Student project, Technische Universität München, 2012.

[4] N. Agarwal and N. Aluru. A data-driven stochastic collocation approach for uncertainty quantification in MEMS. *International Journal for Numerical Methods in Engineering*, 83(5):575–597, 2010.

[5] N. Alexandrov, R. Lewis, C. Gumbert, L. Green, and P. Newman. Approximation and model management in aerodynamic optimization with variable-fidelity models. *Journal of Aircraft*, 38(6):1093–1101, 2001.

[6] D. Allaire, Q. He, J. Deyst, and K. Willcox. An information-theoretic metric of system complexity with application to engineering system design. *Journal of Mechanical Design*, 134(10):100906–10, 2012.

[7] C. Alzate and J. Suykens. A weighted kernel PCA formulation with out-of-sample extensions for spectral clustering methods. In *International Joint Conference on Neural Networks*, 2006.

[8] C. Alzate and J. Suykens. Multiway spectral clustering with out-of-sample extensions through weighted kernel PCA. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(2):335–347, 2010.

[9] S. Amaral, D. Allaire, and K. Willcox. A decomposition approach to uncertainty analysis of multidisciplinary systems. In *12th AIAA Aviation Technology, Integration, and Operations Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference.* AIAA Paper 2012-5563, 2012.

[10] D. Amsallem, J. Cortial, K. Carlberg, and C. Farhat. A method for interpolating on manifolds structural dynamics reduced-order models. *International Journal for Numerical Methods in Engineering*, 80(9):1241–1258, 2009.

[11] D. Amsallem and C. Farhat. Interpolation method for adapting reduced-order models and application to aeroelasticity. *AIAA Journal*, 46(7):1803–1813, 2008.

[12] D. Amsallem and C. Farhat. An online method for interpolating linear parametric reduced-order models. *SIAM Journal on Scientific Computing*, 33(5):2169–2198, 2011.

[13] D. Amsallem, M. Zahr, and C. Farhat. Nonlinear model order reduction based on local reduced-order bases. *International Journal for Numerical Methods in Engineering*, 92(10):891–916, 2012.

[14] E. Anderson, Z. Bai, C. Bischof, L. Blackford, J. Demmel, J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen. *LAPACK Users' guide*. SIAM, 1999.

[15] P. Astrid, S. Weiland, K. Willcox, and T. Backx. Missing point estimation in models described by proper orthogonal decomposition. *Automatic Control, IEEE Transactions on*, 53(10):2237–2251, 2008.

[16] C. Audouze, F. De Vuyst, and P. Nair. Reduced-order modeling of parameterized PDEs using time–space-parameter principal component analysis. *International Journal for Numerical Methods in Engineering*, 80(8):1025–1057, 2009.

[17] C. Audouze, F. De Vuyst, and P. Nair. Nonintrusive reduced-order modeling of parameterized time-dependent partial differential equations. *submitted*, 2010.

[18] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of the 21st ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, 2002.

[19] K. Bache and M. Lichman. UCI machine learning repository, 2013.

[20] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors. *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. SIAM, 2000.

[21] C. T. H. Baker. *The Numerical Treatment of Integral Equations*. Clarendon Press, 1977.

[22] R. Balder. *Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern*. Thesis (Ph.D.), Technische Universitä München, 1994.

[23] M. Barrault, Y. Maday, N.-C. Nguyen, and A. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[24] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[25] Y. Bengio, J.-F. Paiement, and P. Vincent. Out-of-sample extensions for LLE, isomap, MDS, eigenmaps, and spectral clustering. In *Advances in Neural Information Processing Systems*, pages 177–184. MIT Press, 2003.

[26] G. Berkooz, P. Holmes, and J. L. Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Review of Fluid Mechanics*, 25(1):539–575, 1993.

[27] C. Bishop and G. James. Analysis of multiphase flows using dual-energy gamma densitometry and neural networks. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 327(2–3):580–593, 1993.

[28] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2007.

[29] B. Bohn, J. Garcke, R. Iza-Teran, A. Paprotny, B. Peherstorfer, U. Schepsmeier, and C.-A. Thole. Analysis of car crash simulation data with nonlinear machine learning methods. In *Proc. of the International Conference on Computational Science*, 2013.

[30] A. Booker, J. Dennis, P. Frank, D. Serafini, V. Torczon, and M. Trosset. A rigorous framework for optimization of expensive functions by surrogates. *Structural optimization*, 17(1):1–13, 1999.

[31] A. Brandt. *Multigrid techniques: 1984 guide with applications to fluid dynamics*, volume 85 of *GMD-Studien.* GMD, 1984.

[32] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[33] S. Brenner and R. Scott. *The Mathematical Theory of Finite Element Methods.* Springer, 2008.

[34] G. Brock, V. Pihur, S. Datta, and S. Datta. clValid: an R package for cluster validation. *Journal of Statistical Software*, 25(4):1–22, 2008.

[35] M. Buffoni and K. Willcox. Projection-based model reduction for reacting flows. In *40th Fluid Dynamics Conference and Exhibit*, Fluid Dynamics and Co-located Conferences. AIAA Paper 2010-5008, June 2010.

[36] T. Bui-Thanh, K. Willcox, and O. Ghattas. Model reduction for large-scale systems with high-dimensional parametric input space. *SIAM Journal on Scientific Computing*, 30(6):3270–3288, 2008.

[37] H.-J. Bungartz. A multigrid algorithm for higher order finite elements on sparse grids. *Electronic Transactions on Numerical Analysis*, 6:63–77, 1997.

[38] H.-J. Bungartz. *Finite Elements of Higher Order on Sparse Grids.* Habilitationsschrift, Technische Universität München, 1998.

[39] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:1–123, 2004.

[40] H.-J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. Pointwise convergence of the combination technique for Laplace's equation. *Journal of Numerical Mathematics*, 2:21–45, 1994.

[41] H.-J. Bungartz, M. Mehl, T. Neckel, and T. Weinzierl. The PDE framework Peano applied to fluid dynamics: an efficient implementation of a parallel multiscale fluid dynamics solver on octree-like adaptive cartesian grids. *Computational Mechanics*, 46(1):103–114, 2010.

[42] H.-J. Bungartz and S. Zimmer. *Numerische Simulation als interdisziplinäre Herausforderung*. Springer, 2002.

[43] J. Burkardt, M. Gunzburger, and H.-C. Lee. POD and CVT-based reduced-order modeling of Navier–Stokes flows. *Computer Methods in Applied Mechanics and Engineering*, 196(1–3):337–355, 2006.

[44] D. Butnaru. *Computational Steering with Reduced Complexity*. Thesis (Ph.D.), Technische Universität München, 2013.

[45] D. Butnaru, G. Buse, and D. Pflüger. A parallel and distributed surrogate model implementation for computational steering. In *Proc. of the 11th International Symposium on Parallel and Distributed Computing*, 2012.

[46] D. Butnaru, B. Peherstorfer, D. Pflüger, and H.-J. Bungartz. Fast insight into high-dimensional parametrized simulation data. In *11th International Conference on Machine Learning and Applications*, 2012.

[47] D. Butnaru, D. Pflüger, and H.-J. Bungartz. Towards high-dimensional computational steering of precomputed simulation data using sparse grids. In *Proc. of the International Conference on Computational Science*, 2011.

[48] K. Carlberg, C. Bou-Mosleh, and C. Farhat. Efficient non-linear model reduction via a least-squares Petrov-Galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering*, 86(2):155–181, 2011.

[49] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:1–27, 2011.

[50] X. Chang and D. Stehlé. Rigorous perturbation bounds of some matrix factorizations. *SIAM Journal on Matrix Analysis and Applications*, 31(5):2841–2859, 2010.

[51] S. Chaturantabut and D. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[52] S. Chaturantabut and D. Sorensen. A state space error estimate for POD-DEIM nonlinear model reduction. *SIAM Journal on Numerical Analysis*, 50(1):46–63, 2012.

[53] F. R. K. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[54] J. Degroote, J. Vierendeels, and K. Willcox. Interpolation among reduced-order matrices to obtain parameterized models for design, optimization and probabilistic analysis. *International Journal for Numerical Methods in Fluids*, 63(2):207–230, 2010.

[55] T. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*. Springer, 2000.

[56] M. Dihlmann, M. Drohmann, and B. Haasdonk. Model reduction of parametrized evolution problems using the reduced basis method with adaptive time-partitioning. In *Proc. of the International Conference on Adaptive Modeling and Simulation*, 2011.

[57] Q. Du and M. Gunzburger. Model reduction by proper orthogonal decomposition coupled with centroidal Voronoi tessellations. In *Proc. ASME 2002 Joint U.S.-European Fluids Engineering Division Conference*, 2002.

[58] T. Duong. ks: Kernel density estimation and kernel discriminant analysis for multivariate data in R. *Journal of Statistical Software*, 21(7):1–16, 2007.

[59] B. Efron and C. Stein. The jackknife estimate of variance. *Annals of Statistics*, 9(3):586–596, 1981.

[60] J. Eftang, D. Knezevic, and A. Patera. An hp certified reduced basis method for parametrized parabolic partial differential equations. *Mathematical and Computer Modelling of Dynamical Systems*, 17(4):395–422, 2011.

[61] J. Eftang, A. Patera, and E. Rønquist. An hp certified reduced basis method for parametrized elliptic partial differential equations. *SIAM Journal on Scientific Computing*, 32(6):3170–3200, 2010.

[62] J. Eftang and B. Stamm. Parameter multi-domain hp empirical interpolation. *International Journal for Numerical Methods in Engineering*, 90(4):412–428, 2012.

[63] M. Eldred, A. Giunta, and S. Collis. Second-order corrections for surrogate-based optimization with model hierarchies. In *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. AIAA Paper 2004-4457, 2004.

[64] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[65] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in knowledge discovery and data mining*. MIT Press, 1996.

[66] P. Feldmann and R. Freund. Efficient linear circuit analysis by Pade approximation via the Lanczos process. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(5):639–649, 1995.

[67] C. Feuersänger. Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen. Thesis (Master), Universität Bonn, 2005.

[68] C. Feuersänger. *Sparse Grid Methods for Higher Dimensional Approximation.* Thesis (Master), Universität Bonn, 2010.

[69] R. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[70] N. C. O. for Information Technology Research and Development. Computational science: Ensuring America's competitiveness. Technical report, PITAC, 2005.

[71] D. Forsyth and J. Ponce. *Computer Vision.* Pearson, 2012.

[72] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):214–225, 2004.

[73] M. Frangos, Y. Marzouk, K. Willcox, and B. van Bloemen Waanders. Surrogate and reduced-order modeling: A comparison of approaches for large-scale statistical inverse problems. In *Large-Scale Inverse Problems and Quantification of Uncertainty*, pages 123–149. Wiley, 2010.

[74] F. Franzelin. Classification with estimated densities on sparse grids. Thesis (Master), Technische Universität München, 2011.

[75] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[76] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 28(2):337–407, 2000.

[77] M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, 2005.

[78] J. Gama. *Knowledge Discovery from Data Streams.* Chapman & Hall, 2010.

[79] J. Garcke, M. Griebel, and M. Thess. Data mining with sparse grids. *Computing*, 67(3):225–253, 2001.

[80] W. Gilks, N. Best, and K. Tan. Adaptive rejection metropolis sampling within Gibbs sampling. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 44(4):455–472, 1995.

[81] W. Gilks and P. Wild. Adaptive rejection sampling for Gibbs sampling. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 41(2):337–348, 1992.

[82] M. Grepl. *Reduced-Basis Approximation and A Posteriori Error Estimation for Parabolic Partial Differential Equations.* Thesis (Ph.D.), Massachusetts Institute of Technology, 2005.

[83] M. Grepl, Y. Maday, N.-C. Nguyen, and A. Patera. Efficient reduced-basis treatment of nonaffine and nonlinear partial differential equations. *ESAIM: Mathematical Modelling and Numerical Analysis*, 41(03):575–605, 2007.

[84] M. Grepl and A. Patera. A posteriori error bounds for reduced-basis approximations of parametrized parabolic partial differential equations. *ESAIM: Mathematical Modelling and Numerical Analysis*, null:157–181, 2005.

[85] M. Griebel. *Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen-Transformations-Mehrgitter-Methode.* Thesis (Ph.D.), Technische Universität München, 1990.

[86] M. Griebel. A domain decomposition method using sparse grids. In *Contemporary Mathematics, Vol. 157, DDM6*, pages 255–261, 1994.

[87] M. Griebel. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. *Computing*, 61(2):151–179, 1998.

[88] M. Griebel. Sparse grids and related approximation schemes for higher dimensional problems. In *Foundations of Computational Mathematics*, pages 106–161, 2006.

[89] M. Griebel, T. Dornseifer, and T. Neunhoeffer. *Numerische Simulation in der Strömungsmechanik.* Vieweg, 1995.

[90] M. Griebel and M. Holtz. Dimension-wise integration of high-dimensional functions with applications to finance. *Journal of Complexity*, 26(5):455–489, 2010.

[91] M. Griebel, F. Kuo, and I. Sloan. The smoothing effect of the ANOVA decomposition. *Journal of Complexity*, 26:523–551, 2010.

[92] M. Griebel and P. Oswald. On additive Schwarz preconditioners for sparse grid discretizations. *Numerische Mathematik*, 66(1):449–463, 1993.

[93] M. Griebel and P. Oswald. Tensor product type subspace splittings and multilevel iterative methods for anisotropic problems. *Advances in Computational Mathematics*, 4(1):171–206, 1995.

[94] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In *Iterative Methods in Linear Algebra*, 1992.

[95] M. Griebel, C. Zenger, and S. Zimmer. Multilevel Gauss-Seidel-algorithms for full and sparse grid problems. *Computing*, 50(2):127–148, 1993.

[96] B. Haasdonk, M. Dihlmann, and M. Ohlberger. A training set and multiple basis generation approach for parametrized model reduction based on adaptive grids in parameter space. In *Proc. Mathematical and Computer Modelling of Dynamical Systems*, 2011.

[97] B. Haasdonk and M. Ohlberger. Adaptive basis enrichment for the reduced basis method applied to finite volume schemes. In *Proc. 5th International Symposium on Finite Volumes for Complex Applications*, 2008.

[98] B. Haasdonk and M. Ohlberger. Space-adaptive reduced basis simulation for time-dependent problems. In *Proc. Vienna International Conference on Mathematical Modelling*, 2009.

[99] W. Hackbusch. *Multi-Grid Methods and Applications.* Springer, 1985.

[100] L. Hansen and P. Salamon. Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(10):993–1001, 1990.

[101] H. Harbrecht, R. Schneider, and C. Schwab. Multilevel frames for sparse tensor product spaces. *Numerische Mathematik*, 110(2):199–220, 2008.

[102] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2009.

[103] X. He, S. Yan, Y. Hu, and H.-J. Zhang. Learning a locality preserving subspace for visual recognition. In *9th IEEE International Conference on Computer Vision*, 2003.

[104] M. Hegland, J. Garcke, and V. Challis. The combination technique and some generalisations. *Linear Algebra and its Applications*, 420(2–3):249–275, 2007.

[105] M. Hegland, G. Hooker, and S. Roberts. Finite element thin plate splines in density estimation. *ANZIAM Journal*, 42:C712–C734, 2000.

[106] A. Heinecke, B. Peherstorfer, D. Pflüger, and Z. Song. Sparse grid classifiers as base learners for AdaBoost. In *International Conference on High Performance Computing and Simulation*, 2012.

[107] A. Heinecke and D. Pflüger. Multi- and many-core data mining with adaptive sparse grids. In *Proc. of the 8th ACM International Conference on Computing Frontiers*, 2011.

[108] A. Heinecke and D. Pflüger. Emerging architectures enable to boost massively parallel data mining using adaptive sparse grids. *International Journal of Parallel Programming*, 2012.

[109] A. Heinecke, S. Schraufstetter, and H.-J. Bungartz. A highly-parallel Black-Scholes solver based on adaptive sparse grids. *International Journal of Computer Mathematics*, 2012.

[110] C.-W. Hsu, C.-C. Chang, and C.-J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, 2003.

[111] D.-S. Huang, X.-P. Zhang, G.-B. Huang, Y. Pang, L. Zhang, Z. Liu, N. Yu, and H. Li. Neighborhood preserving projections (NPP): a novel linear dimension reduction method. In *Advances in Intelligent Computing*. Springer, 2005.

[112] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[113] R. Iza-Teran. Enabling the analysis of finite element simulation-bundles. *SIAM Journal of Uncertainty Quantification*, 2012. accepted.

[114] A. Izenman. Recent developments in nonparametric density estimation. *Journal of the American Statistical Association*, 86(413):205–224, 1991.

[115] T. Jebara. *Machine Learning. Discriminative and Generative.* Kluwer Academic Publishers, 2004.

[116] M. Jones, J. Marron, and S. Sheather. A brief survey of bandwidth selection for density estimation. *Journal of the American Statistical Association*, 91(433):401–407, 1996.

[117] M. Kahlbacher and S. Volkwein. Galerkin proper orthogonal decomposition methods for parameter dependent elliptic systems. *Discussiones Mathematicae: Differential Inclusions, Control and Optimization*, 27:95–117, 2007.

[118] M. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(3):425–464, 2001.

[119] N. Khalil and F. Zacherl. Using pre-computed matrices in a density-based classification method with sparse grids. Student project, Technische Universität München, 2012.

[120] C. Kirst. Optimierung und Erweiterung eines Softwarepakets zur Berechnung von Reduced Basis Lösungen. Student project, Technische Universität München, 2012.

[121] A. Klenke. *Probability Theory: A Comprehensive Course.* Springer, 2008.

[122] D. Knezevic, N.-C. Nguyen, and A. Patera. Reduced basis approximation and a posteriori error estimation for the parametrized unsteady Boussinesq equations. *Mathematical Models and Methods in Applied Sciences*, 21(07):1415–1442, 2011.

[123] K. Kunisch and S. Volkwein. Galerkin proper orthogonal decomposition methods for parabolic problems. *Numerische Mathematik*, 90(1):117–148, 2001.

[124] K. Kunisch and S. Volkwein. Galerkin proper orthogonal decomposition methods for a general equation in fluid dynamics. *SIAM Journal on Numerical Analysis*, 40(2):492–515, 2002.

[125] J. Lee and M. Verleysen. *Nonlinear Dimensionality Reduction*. Springer, 2007.

[126] C. Lieberman, K. Willcox, and O. Ghattas. Parameter and state model reduction for large-scale statistical inverse problems. *SIAM Journal on Scientific Computing*, 32(5):2523–2542, 2010.

[127] Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu. Understanding of internal clustering validation measures. *10th International Conference on Data Mining*, 2010.

[128] B. Lohmann and R. Eid. Efficient order reduction of parametric and nonlinear models by superposition of locally reduced models. In *Methoden und Anwendungen der Regelungstechnik*, 2009.

[129] H. Ly and H. Tran. Modeling and control of physical processes using proper orthogonal decomposition. *Proc. of the 6th Bozeman conference on computation and control*, 33(1-3):223–236, 2001.

[130] X. Ma and N. Zabaras. An adaptive high-dimensional stochastic model representation technique for the solution of stochastic partial differential equations. *Journal of Computational Physics*, 229(10):3884–3915, 2010.

[131] Y. Maday and B. Stamm. Locally adaptive greedy approximations for anisotropic parameter reduced basis spaces. *arXiv:1204.3846*, 2012.

[132] J. Marron and M. Wand. Exact mean integrated squared error. *Annals of Statistics*, 20(2):712–736, 1992.

[133] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th International Conference on Computer Vision*, 2001.

[134] L. Mei and C.-A. Thole. Data analysis for parallel car-crash simulation results and model optimization. *Simulation Modelling Practice and Theory*, 16(3):329–337, 2008.

[135] B. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *Automatic Control, IEEE Transactions on*, 26(1):17–32, 1981.

[136] K. Murphy. *The Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[137] L. Ng, D. Huynh, and K. Willcox. Multifidelity uncertainty propagation for optimization under uncertainty. In *12th AIAA Aviation Technology, Integration, and*

Operations Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference. AIAA Paper 2012-5602, 2012.

[138] N.-C. Nguyen, G. Rozza, and A. Patera. Reduced basis approximation and a posteriori error estimation for the time-dependent viscous Burgers' equation. *Calcolo*, 46(3):157–185, 2009.

[139] A. Noor and J. Peters. Reduced basis technique for nonlinear analysis of structures. *AIAA Journal*, 18(4):455–462, 1980.

[140] E. Nyström. Über die Praktische Auflösung von Integralgleichungen mit Anwendungen auf Randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

[141] H. Panzer, J. Mohring, R. Eid, and B. Lohmann. Parametric model order reduction by matrix interpolation. *at – Automatisierungstechnik*, 58(8):475–484, 2010.

[142] A. Patera and G. Rozza. *Reduced Basis Approximation and A Posteriori Error Estimation for Parametrized Partial Differential Equations.* MIT Pappalardo Graduate Monographs in Mechanical Engineering, 2007.

[143] B. Peherstorfer. HT-Multigrid-compatible time/space discretizations of the Stokes equations in a pressure gradient formulation. Thesis (Master), Technische Universität München, 2010.

[144] B. Peherstorfer. Reduced basis methods with sparse grids. Student project, Technische Universität München, 2010.

[145] B. Peherstorfer and H.-J. Bungartz. Semi-coarsening in space and time for the hierarchical transformation multigrid method. *Procedia Computer Science*, 9(0):2000–2003, 2012.

[146] B. Peherstorfer, D. Butnaru, K. Willcox, and H. Bungartz. Localized discrete empirical interpolation method. *SIAM Journal on Scientific Computing*, 2013. submitted.

[147] B. Peherstorfer, D. Pflüger, and H.-J. Bungartz. A sparse-grid-based out-of-sample extension for dimensionality reduction and clustering with Laplacian eigenmaps. In *Proc. of AI 2011: Advances in Artificial Intelligence*. Springer, 2011.

[148] B. Peherstorfer, D. Pflüger, and H.-J. Bungartz. Clustering based on density estimation with sparse grids. In *KI 2012: Advances in Artificial Intelligence.* Springer, 2012.

[149] B. Peherstorfer, S. Zimmer, and H.-J. Bungartz. Model reduction with the reduced basis method and sparse grids. In *Sparse Grids and Applications.* Springer, 2013.

[150] C. Pflaum. *Diskretisierung elliptischer Differentialgleichungen mit dünnen Gittern.* Thesis (Ph.D.), Technische Universität München, 1996.

[151] C. Pflaum. A multilevel algorithm for the solution of second order elliptic differential equations on sparse grids. *Numerische Mathematik*, 79(1):141–155, 1998.

[152] D. Pflüger. Data Mining mit Dünnen Gittern. Thesis (Master), Universität Stuttgart, 2005.

[153] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems.* Verlag Dr. Hut, 2010.

[154] D. Pflüger. Spatially adaptive refinement. In *Sparse Grids and Applications.* Springer, 2012.

[155] D. Pflüger, I. Muntean, and H.-J. Bungartz. Adaptive sparse grid classification using grid environments. In *Proc. International Conference on Computational Science.* Springer, 2007.

[156] D. Pflüger, B. Peherstorfer, and H.-J. Bungartz. Spatially adaptive sparse grids for high-dimensional data-driven problems. *Journal of Complexity*, 26(5):508–522, 2010.

[157] H. Qiao, P. Zhang, D. Wang, and B. Zhang. An explicit nonlinear mapping for manifold learning. *Cybernetics, IEEE Transactions on*, 43(1):51–63, 2013.

[158] A. Quarteroni and G. Rozza. Numerical solution of parametrized Navier–Stokes equations by reduced basis methods. *Numerical Methods for Partial Differential Equations*, 23(4):923–948, 2007.

[159] H. Rabitz and Ö. Alis. General foundations of high-dimensional model representations. *Journal of Mathematical Chemistry*, 25(2):197–233, 1999.

[160] M. Rathinam and L. Petzold. A new look at proper orthogonal decomposition. *SIAM Journal on Numerical Analysis*, 41(5):1893–1925, 2003.

[161] M. Rewieński and J. White. Model order reduction for nonlinear dynamical systems based on trajectory piecewise-linear approximations. *Linear Algebra and its Applications*, 415(2–3):426–454, 2006.

[162] C. Robert and G. Casella. *Monte Carlo Statistical Methods.* Springer, 2004.

[163] S. Roberts and S. Bolt. A note on the convergence analysis of a sparse grid multivariate probability density estimator. In *Proc. of the 14th Biennial Computational Techniques and Applications Conference*, volume 50 of *ANZIAM Journal*, pages C858–C870, 2009.

[164] P. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(1):53–65, 1987.

[165] B. Schölkopf and A. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2002.

[166] B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 2011.

[167] C. Schwab and R. Todor. Sparse finite elements for stochastic elliptic problems – higher order moments. *Computing*, 71(1):43–63, 2003.

[168] D. Scott and B. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* Wiley, 1992.

[169] B. Silverman. *Density Estimation for Statistics and Data Analysis.* CRC Press, 1986.

[170] L. Sirovich. Turbulence and the dynamics of coherent structures. *Quarterly of Applied Mathematics*, pages 561–571, 1987.

[171] S. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. In *Dokl. Akad. Nauk SSSR*, volume 148, pages 1042–1043, 1963. Engl. Transl.: Soviet Math. Dokl. 4:240-243, 1963.

[172] P. Smyth and D. Wolpert. Linearly combining density estimators via stacking. *Machine Learning*, 36(1-2):59–83, 1999.

[173] B. Stein, S. Meyer zu Eissen, and F. Wissbrock. On cluster validity and the information need of users. In *International Conference on Artificial Intelligence and Applications*, 2003.

[174] C.-A. Thole, L. Nikitina, I. Nikitin, and T. Clees. Advanced mode analysis for crash simulation results. In *Proc. of the 9th LS-DYNA Forum*, 2010.

[175] U. Trottenberg, C. Oosterlee, and A. Schüller. *Multigrid.* Academic Press, 2001.

[176] A. Uzilov, J. Keegan, and D. Mathews. Detection of non-coding RNAs on the basis of predicted secondary structure formation free energy change. *BMC Bioinformatics*, 7(1):173, 2006.

[177] V. Vapnik. *Statistical Learning Theory.* Wiley, 1998.

[178] K. Veroy and A. Patera. Certified real-time solution of the parametrized steady incompressible Navier–Stokes equations: rigorous reduced-basis a posteriori error bounds. *International Journal for Numerical Methods in Fluids*, 47(8–9):773–788, 2005.

[179] U. von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17:395–416, 2007.

[180] U. von Luxburg, R. Williamson, and I. Guyon. Clustering: Science or art? *Journal of Machine Learning Research – Proceedings Track*, 27:65–80, 2012.

[181] D. Wagner and F. Wagner. Between min cut and graph bisection. In *Mathematical Foundations of Computer Science*. Springer, 1993.

[182] G. Wahba. *Spline Models for Observational Data*. SIAM, 1990.

[183] K. Washabaugh, D. Amsallem, M. Zahr, and C. Farhat. Nonlinear model reduction for CFD problems using local reduced-order bases. In *42nd AIAA Fluid Dynamics Conference and Exhibit*. AIAA Paper 2012-2686, 2012.

[184] M. Way, J. Scargle, K. Ali, and A. Srivastava, editors. *Advances in Machine Learning and Data Mining for Astronomy*. Chapman & Hall, 2012.

[185] D. Wilkinson. Parallel Bayesian computation. In *Handbook of Parallel Computing and Statistics*. Chapman & Hall, 2006.

[186] C. Williams and M. Seeger. The effect of the input density distribution on kernel-based classifiers. In *Proc. of the 17th International Conference on Machine Learning*, 2000.

[187] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, 2001.

[188] D. Wirtz, D. Sorensen, and B. Haasdonk. A-posteriori error estimation for DEIM reduced nonlinear dynamical systems. *Submitted to SIAM Journal on Scientific Computing*, 2012.

[189] D. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[190] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2008.

[191] H. Yserentant. On the multi-level splitting of finite element spaces. *Numerische Mathematik*, 49(4):379–412, 1986.

[192] N. Zabaras and B. Ganapathysubramanian. A scalable framework for the solution of stochastic inverse problems using a sparse grid collocation approach. *Journal of Computational Physics*, 227(9):4697–4735, 2008.

[193] F. Zacherl. New loss functions for sparse grid classifiers. Thesis (Master), Technische Universität München, 2013.

[194] A. Zeiser. Fast Matrix-Vector multiplication in the sparse-grid Galerkin method. *SIAM Journal of Scientific Computing*, pages 1–19, 2010.

[195] C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations*, volume 31 of *Notes on Numerical Fluid Mechanics*, pages 241–251. Vieweg, 1991.

[196] Y. Zhou. *Model reduction for nonlinear dynamical systems with parametric uncertainties*. Thesis (Master), Massachusetts Institute of Technology, Dept. of Aeronautics and Astronautics, 2012.

[197] J. Zupan, M. Novič, X. Li, and J. Gasteiger. Classification of multicomponent analytical data of olive oils using different neural networks. *Analytica Chimica Acta*, 292(3):219–234, 1994.