DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploitation of Component Grid Symmetries for Sparse Grid Density Estimation with the Combination Method

Kilian Glas

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploitation of Component Grid Symmetries for Sparse Grid Density Estimation with the Combination Method

Ausnutzung von Komponentengittersymmetrien für Dünngitter Dichteschätzung mit der Kombinationsmethode

Author:	Kilian Glas
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Kilian Röhner, M.Sc.
Submission Date:	15.10.2019

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.10.2019 Kilian Glas

Abstract

In this bachelor thesis, a method is introduced to improve the offline step of sparse grid density estimation with the combination technique. The developed approach exploits the geometrical properties of the subgrids in the combination scheme, to transform already decomposed corresponding system matrices. The transformation consists of a symmetric permutatation of the system matrix, aswell as the elementwise multiplication of a dimension blow-up factor. The former can be applied when two subgrids have level vectors, that are permutations of each other, while the latter yields an embedding into higher dimensions. The applicability is examined for the orthogonal decomposition into hessenberg form and the cholesky decomposition. For the orthogonal decomposition, the method has been implemented. Compared to the current implementation, it provides a speed up from cubic to quadratic time for the offline step of suitable component grids.

Contents

1. Introduction

Due to the rise of the internet during the last three decades and other more recent developments, such as the internet of things or more powerful sensors, the amount of generated data has grown exponentially. Simultaneously, hardware has become more powerful computationally, due to the effect of Moore's Law and the exploitation of parallel data processing, and storage has become cheaper and larger. Therefore, today it is possible, to process huge amounts of data, and to gain value from this plentiful resource. Consequently, efficient methods for data mining, which aims at discovering properties of a data set, and machine learning tasks such as classification, are much-needed. These methods are used for tasks such as image classification, user recommendations or evaluating the results of scientific experiments.

An important field in data mining is density estimation, which aims at approximating the underlying density function of a given data set. One approach for density estimation is to approximate the density function using interpolation on a full grid. However, such methods suffer from what is called the curse of dimensionality, which refers to the exponential growth of grid points with the number of dimensions of the approximation space. In the context of data mining in general, a dimension of the data set corresponds to a feature, and a higher amount of considered features often leads to a more accurate result.

Sparse grids tackle the curse of dimensionality, by omitting less important grid points. They can be derived by optimizing the ratio between the amount of grid points, and the introduced error. In [Wae17], sparse grid density estimation has been used for image classification, which is a high dimensional problem by nature, since each pixel of an image corresponds to a dimension in the approximation space.

A variant of sparse grids is the combination technique, which constructs the solution of a problem on a regular sparse grid, by combining multiple subproblems on smaller full grids. This yields several benefits compared to the conventional method. On the one hand, these advantages come from the fact, that the components can be treated independently, which enables parallelization and often times reduces the combined computational cost. On the other hand, the components are full grids and thus easier to handle when implementing algorithms.

In the context of the combination technique, a regular sparse grid has an associated combination scheme, which contains all corresponding component grids. Within this

scheme, some component grids have the same geometrical structure, asides from being reflections of each other. In this thesis, we introduce a method to exploit these symmetries, to lower the computational effort for sparse grid density estimation. Furthermore, the derived approach has been implemented in the C++ library SG++, which contains implementations of sparse grids for different applications [Pfl10b].
In Chapter 2, the mathematical background of this thesis is discussed. The concept of sparse grids and sparse grid density estimation is introduced. Additionally, less niche topics important for the derivation of the method are covered briefly. In Chapter 3 the approach to exploit the mentioned symmetries is introduced and analyzed in theory regarding correctness and applicability. Chapter 4 describes the implementation and examines the running time complexities of the implemented algorithms, as well as the integration of the method into SG++. Finally, Chapter 5 evaluates the results in terms of running time improvement and error introduced, compared to the conventional implementation in the library.

2. Theoretical Background

This chapter provides an overview of the mathematical background needed for this thesis. Section 2.2 covers the idea of sparse grids in general and Section 2.3 introduces the application of such to density estimation. In Section 2.4, methods for decomposing the system matrix introduced in Section 2.3 are described and Section 2.5 covers permutations, which are needed to derive the method introduced in Chapter 3.

2.1. Notation

At first, a few notations used in this thesis are introduced.
Vectors are denoted as lower case letters with an arrow $\vec{v}$, $\vec{v}_i$ represents the i-th element of the vector. Matrices are denoted by uppercase letters A, $A_{i,j}$ is the matrix element in the i-th row and j-th column.
The l_1-norm and the maximum norm of a d-dimensional vector are defined as:

$$|\vec{v}|_1 = \sum_{i=1}^{d} \vec{v}_i \text{ and } |\vec{v}|_\infty = \max_{1 \leq i \leq d} |\vec{v}_i|$$

Furthermore, the L_p-norm of a function $f : \Omega \subset \mathbb{R}^d \mapsto \mathbb{R}$ is defined as:

$$\| f \|_{L_p}^p = \int_\Omega |f(\vec{x})|^p d^d\vec{x}$$

Finally, the L_2-inner-product for continuous functions $f, g : \Omega \subset \mathbb{R} \mapsto \mathbb{R}$ is defined as:

$$\langle f, g \rangle = \int_\Omega f(x)g(x)dx$$

2.2. Sparse Grids

Sparse grids are a method to overcome the curse of dimensionality, i.e. the exponential growth of function evaluations, when using full equidistant grids for interpolation.

They are the result of optimizing the ratio between grid points, i.e. function evaluations and thus cost, and the asymptotic error of the approximation. Sparse grids can be applied in a variety of fields, such as solving partial differential equations, interpolation and others. The following sections provide a concise introduction to the topic, by taking sparse grid interpolation as an example. We first discuss hierarchical basis functions in the context of conventional full grid interpolation, in one and higher dimensional spaces. Subsequently, this is used as basis to derive sparse grids. At the end of this section, the combination technique for sparse grids is described, as well as adaptivity of sparse grids. For detailed information about sparse grids in general, [BG04] and [Pfl10a] are highly recommended.

2.2.1. Interpolation with Full Grids

We start of with conventional full grid interpolation. In this context, we want to approximate arbitrary unknown functions $f : [0,1]^d \mapsto \mathbb{R}$. Using dilation, most d-dimensional problems can be defined on the d-dimensional unit cube. To interpolate such a function, the underlying approximation space, i.e. the unit cube, has to be discretized. Full grid interpolation approaches define equidistant grid points with a mesh width 2^{-l} in all dimensions, where l denotes the level of the grid. Every grid point x_i serves as a support point of a corresponding basis function, thus f is evaluated in x_i. Using piecewise linear basis functions $\varphi_i(\vec{x})$, f can be approximated by the interpolant $u(\vec{x})$, which is defined as a weighted sum of basis functions:

$$u(\vec{x}) = \sum_i = \alpha_i \varphi_i(\vec{x}) \tag{2.1}$$

This method is visualized in Figure 2.1.
The weights α are obtained from the function evaluations at the corresponding support point.

Hierarchical Basis Functions

To derive sparse grids, a hierarchical decomposition of the approximation space is needed. Compared to the conventional nodal point basis described in the previous section, the grid points are divided into hierarchical subsets. This approach is discussed in the following.
This thesis only considers linear basis functions. Linear basis functions are accurate enough in many scenarios, but the principal of hierarchical grid interpolation can be implemented using various other basis functions, for further information see [Pfl10a] or [BG04].
The linear basis function used in this thesis, is the standard hat function:

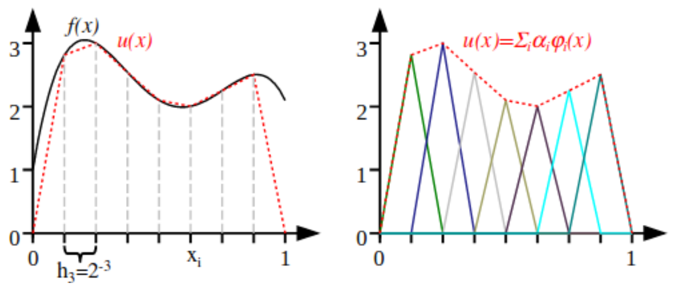


Figure 2.1.: The figure shows the unknown function f and the corresponding picewise linear interpolant u on the left. On the right, the basis functions are depicted. Figure taken from [Pfl10a].

$$\varphi(x)_{\text{lin}} = \max(1 - |x|, 0) \tag{2.2}$$

For the hierarchical decomposition of the approximation space, levels l are introduced. Each level contains a number of basis functions centered at points $x_{l,i}$, corresponding to a set of indices I_l. Those points are the only support of the corresponding basis functions. With $0 < i < 2^l$, the one dimensional level dependent basis functions are obtained from the standard hat function $\varphi(x)$ using dilation and translation:

$$\varphi(x)_{l,i} = \varphi(2^l x - i) \tag{2.3}$$

Thus, the points $x_{l,i}$ are located at:

$$x_{l,i} = 2^{-l} i \tag{2.4}$$

Defining the level dependent index set I_l,

$$I_l = \{i \in \mathbb{N} : 1 \leq i \leq 2^l - 1,\ i \text{ odd}\} \tag{2.5}$$

leads to the set of hierarchical subspaces W_l,

$$W_l = \text{span}\{\varphi_{l,i} : i \in I_l\} \tag{2.6}$$

shown in Figure 2.2.
The space of piecewise linear functions on a full grid V_n, for a given level n, can be formulated as the weighted sum of subspaces W_l:

$$V_n = \oplus_{l \leq n} W_l \tag{2.7}$$

Figure 2.3 shows the hierarchical interpolation of the example from Figure 2.1.

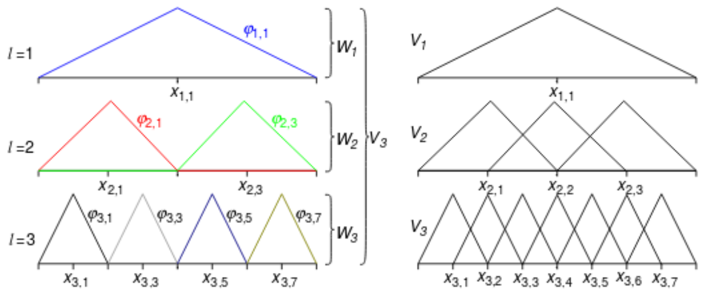


Figure 2.2.: The subspaces W_n with one-dimensional basis functions $\varphi_{l,i}$ and grid points $x_{l,i}$ (right) and the standard full grid approach (right). Figure taken from [Pfl10a].

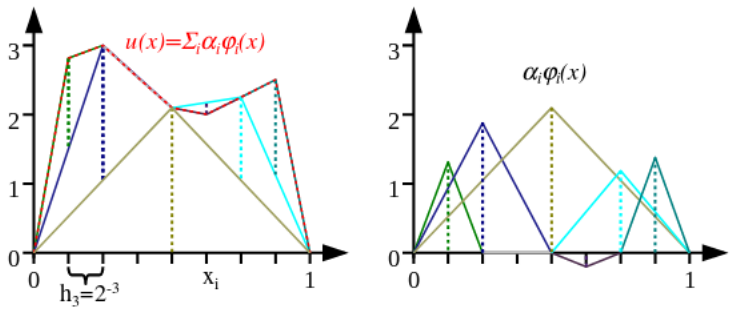


Figure 2.3.: The figure shows the linear interpolation on a one-dimensional full grid. On the left the picewise linear interpolant u is shown. The right hand side depicts the corresponding hierarchical basis functions.. Figure taken from [Pfl10a].

Higher Dimensions

This concept can be expanded to higher dimensions, by defining a multi dimensional basis function $\varphi_{\vec{l},\,\vec{i}}$, using a tensor product of one dimensional basis functions:

$$\varphi_{\vec{l},\,\vec{i}} = \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j) \tag{2.8}$$

Where $\vec{i}$ and $\vec{l}$ are d-dimensional vectors, with elements representing the level and index for each dimension.
Similarly, the remaining definitions from the previous section can be adjusted. The multidimensional index set $I_{\vec{l}}$,

$$I_l = \{i \in \mathbb{N}^d : 1 \leq i_j \leq 2^{l_j} - 1,\ i_j\text{odd},\ 1 \leq j \leq d\} \tag{2.9}$$

and the subspaces $W_{\vec{l}}$,

$$W_l = \text{span}\{\varphi_{l,i} : i \in I_l\} \tag{2.10}$$

where $\vec{l}$ again denotes the d-dimensional level.
This leads to the space of d-dimensional piecewise linear functions with mesh width h_n:

$$V_n = \oplus_{|\vec{l}|_\infty \leq n} W_{\vec{l}} \tag{2.11}$$

Figure 2.4 shows the corresponding basis functions of all subspaces of a 2-dimensional full grid of level 3.

When expanding to higher dimensions, the curse of dimensionality arises. A d-dimensional full grid contains $(2^n - 1)^d$ grid points. This results in

$$\mathcal{O}(2^{nd}) \tag{2.12}$$

function evaluations. Hence, the cost of full grid interpolation increases exponentially with respect to the number of dimensions, which poses a huge drawback when dealing with high dimensional problems. This observation leads to the sparse grid approach.

2.2.2. Modlinear basis function

The linear basis functions used in the previous sections, are zero on the domain's boundary $\delta\Omega$. Hence, they do not contain boundary information. If such information is needed, it is possible to add basis functions with corresponding support points on the boundary. However, this results in exponential growth of the number of grid points

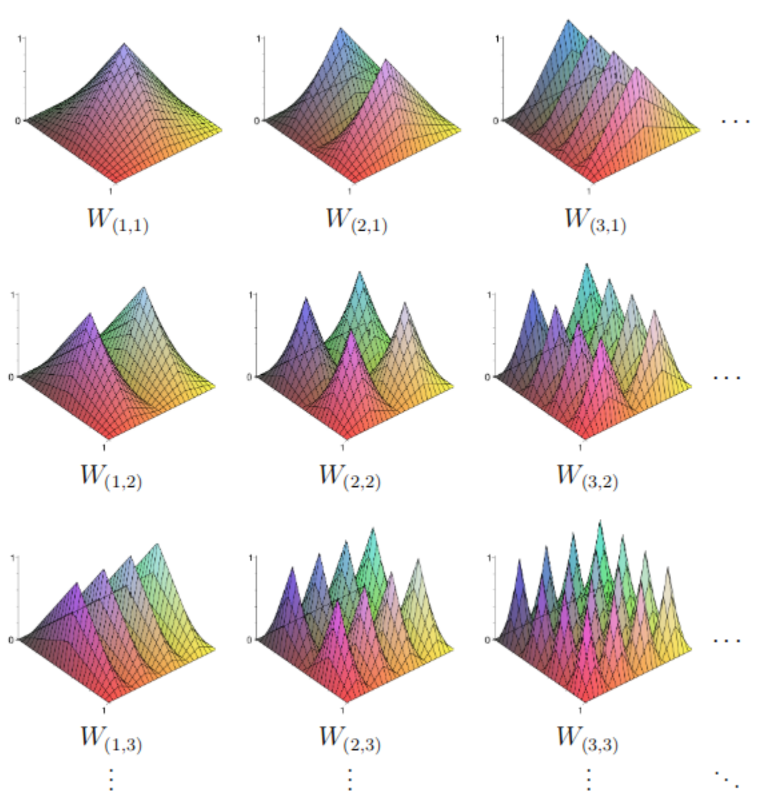


Figure 2.4.: The basis functions of the 2-dimensional subspaces $W_{\vec{l}}$ for $|\vec{l}|_\infty \leq 3$. Figure taken from [Pfl10a].

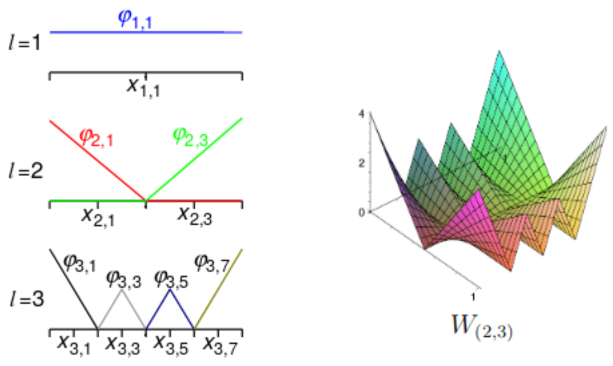


Figure 2.5.: The one-dimensional modified linear basis functions up to level 3 on the left. On the right, the basis functions of the 2-dimensional subspace $W_{2,3}$. Figure taken from [Pfl10a].

on the boundary (see [Pfl10a]).
Instead of adding basis functions, another way is to extrapolate towards the boundary. This leads to the modified linear basis function, which is denoted as modlinear basis function throughout the rest of this thesis [Pfl10a]:

$$\varphi_{l,i} = \begin{cases} 1 & l = 1 \wedge i = 1 \\ \left\{ \begin{array}{ll} 2 - 2^l x & x \in [0, 2^{-l+1}] \\ 0 & \text{else} \end{array} \right\} & l > 1 \wedge i = 1 \\ \left\{ \begin{array}{ll} 2^l x + 1 - i & x \in [1 - 2^{-l+1}, 1] \\ 0 & \text{else} \end{array} \right\} & l > 1 \wedge i = 2^l - 1 \\ \varphi(2^l x - i) & \text{else} \end{cases} \tag{2.13}$$

Figure 2.5 shows one- and two-dimensional example basis functions.

2.2.3. From Full Grids to Sparse Grids

The general idea of sparse grids is to omit grid points of a full grid, and only keep the points, that contribute the most to the interpolant.
This is achieved by selecting the subspaces from the hierarchical representation of the full grid, whose basis functions have the highest support. This optimization problem can be formulated as a continuous knapsack problem, for a more detailed description we refer to [BG04].

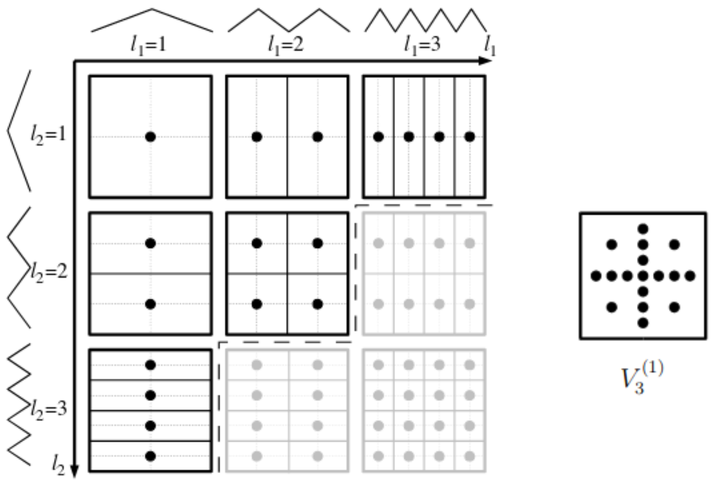


Figure 2.6.: Sparse grid of level $n = 3$ for the sparse grid space $V_3^{(1)}$ (right) and the corresponding subspaces $W_{\vec{l}}$ (left). Figure taken from [Pfl10a].

When optimizing the error regarding the L_2 or maximum norm, the set of sparse grid interpolants of level n,

$$V_n^{(1)} = \oplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}} \tag{2.14}$$

Sis obtained. Figure 2.6 displays the 2-dimensional sparse grid of level $n = 3$ and its corresponding subspaces $W_{\vec{l}}$.
Using sparse grids reduces the amount of function evaluations to:

$$\mathcal{O}(2^n \ n^{d-1}) \tag{2.15}$$

This poses a significant improvement compared to (2.12). On the other hand, the asymptotic error only increases slightly from $\mathcal{O}(h_n^2)$ to:

$$\mathcal{O}(h_n^2 (\log h_n^{-1})^{d-1}) \tag{2.16}$$

These values are taken from [BG04].

2.2.4. Sparse Grid Combination Technique

A variant of sparse grids is the combination technique [GSZ92]. A sparse grid interpolant can be expressed as a linear combination of coarser, so called anisotropic or

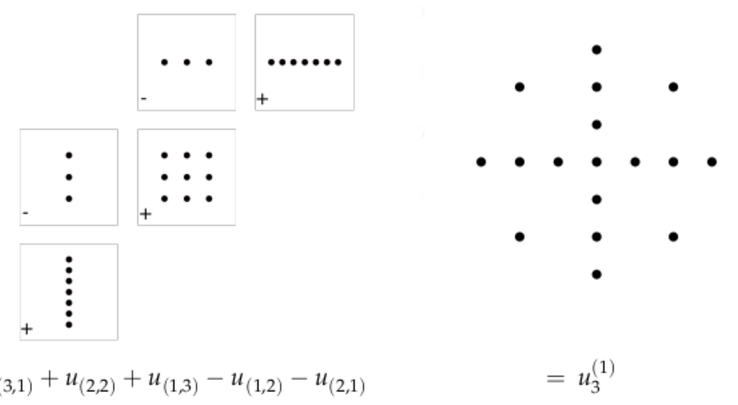


Figure 2.7.: Anisotropic component grids (left) for a corresponding $u_3^{(1)} \in V_3^{(1)}$ (right). The coefficients of the components are 1 and -1, respectively. Figure taken from [Rös19].

anisotropic full grids. An anisotropic grid is a full grid with level vector $\vec{l}$ and mesh widths h_d in the corresponding dimension d. A sparse grid interpolant $u(\vec{x}) \in V_n^{(1)}$ can be represented as:

$$u(\vec{x}) = \sum_{k=0}^{d-1} (-1)^k \binom{d-1}{k} \sum_{|\vec{l}|_1 = n+(d-1)-k} u(\vec{x})_{\vec{l}} \tag{2.17}$$

The function $u(\vec{x})_{\vec{l}}$ denotes the sub interpolant on the anisotropic component grid with level vector $\vec{l}$. The resulting component grid structure is displayed in Figure 2.7. Note that in higher dimensional examples, the coefficients of the components can be different to 1 or -1.

Since the total solution is obtained by combining component solutions, the combination technique provides several benefits compared to using standard sparse grids. Due to their independence, component interpolants can be computed in parallel. Furthermore, the computational cost of algorithms using grid interpolation in general - including sparse grid interpolation - depends on the number of grid points of the underlying grid. Hence, if the memory or running time complexity of an algorithm in non-linear, the combined cost for solving multiple, much smaller subproblems is significantly lower. In addition, full grids are easier to handle during implementation.

2.2.5. Spatial Adaptivity

In real world applications, examined function f can have different degrees of smoothness over the approximation space. Therefore, inserting points where the function is fluctuating a lot, can be highly beneficial with respect to overall accuracy of the interpolation. On the other hand, areas where the function is comparatively smooth may be suitable for coarsening the grid, i.e. removing points, which improves performance. Refinement and coarsening can be performed on regular sparse grids and anisotropic component grids, using a hierarchical approach analogous to Section 2.2.3. Adaptivity is of minor importance for this thesis, hence, a detailed explanation is renounced. The topic is described in depth in [Pfl10a].

2.3. Sparse Grid Density Estimation

Sparse grid interpolation can be used for density estimation, i.e. to approximate the underlying density function of a given data set. Especially for high dimensional data sets, this is a promising approach [Peh13]. In real world applications, a higher amount of dimensions of a data set, i.e the features of the data points, often leads to more accurate results. Other tasks such as image classification, require a high amount of dimensions by nature [Wae17].

2.3.1. Density Estimation

Density estimation is a statistical method, to estimate the underlying density function of a given data set $S = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d$, where x_i are samples of a random variable X with unknown distribution $p(X)$. In practice, those random variables are often highly dimensional. In general, it is distinguished between parametric and non-parametric density estimation. The parametric approach is applicable, when general information about the underlying density function is available. For this case, a fixed probability distribution can be chosen and only its parameters have to be adjusted. A widely used example of this approach is the Gaussian mixture model. However, in most cases, information about the examined random variable is not available, especially in the case of more complex correlations. For such scenarios a non-parametric approach can be used. Kernel density estimation is the most widely used method, but is costly to achieve good results with, and is unsuitable for higher dimensions due to an exponential growth of the computational cost (curse of dimensionality) [Peh13].

2.3.2. Density Estimation with Sparse Grids

Due to those performance issues, using sparse grids for density estimation is a promising method, especially when dealing with large or highly dimensional data sets. The approach is non-parametric and able overcome some drawbacks of methods such as kernel density estimation.
Essentially, the idea is to generalize a highly over fitted initial guess by using spline smoothing. In [Peh13], for the initial guess p_ϵ, Dirac delta functions centered at the data points x_i are used. Thus, the initial density function can be thought of having a value of 0 at every point $x' \in \Omega \setminus S$, and a value of infinity at every point $x_i \in S$. With V denoting an arbitrary function space, the smoothed approximation p is obtained by the following least squares approach:

$$p = \underset{f \in V}{\arg\min} \int_\Omega (f(x) - p_\epsilon(x))^2 \, dx + \lambda \|\Lambda f\|_{L_2}^2 \tag{2.18}$$

In Equation 2.18, the first term causes the approximation to be as close to the initial guess as possible, while $\lambda \parallel \Lambda f \parallel_{L_2}^2$ is a regularization term to guarantee a certain degree of smoothness. For Λ one might choose ∇ or some other kind of derivative, while λ is a regularization parameter that can be adjusted.
As already mentioned the initial guess p_ϵ is a linear combination of Dirac delta functions δ_{x_i} centered at the data points x_i, hence:

$$p_\epsilon = \frac{1}{M} \sum_{i=1}^{M} \delta_{x_i} \tag{2.19}$$

Because $\int_\Omega \delta_{x_i} dx = 1$ per definition of Dirac delta functions, the factor $\frac{1}{M}$ ensures that $\int_\Omega p_\epsilon(x) dx = 1$.
For sparse grid density estimation for the function space V, the sparse grid interpolation space $V_l^{(1)}$ is chosen. Let Φ_l be the set of hierarchical basis functions of $V_l^{(1)}$. Inserting Equation 2.19 into Equation 2.18 and applying some transformations (see [HHR00]), leads to the variational equation,

$$\int_\Omega p(x)\varphi(x) dx + \lambda \int_\Omega \Lambda p(x) \ \Lambda\varphi(x) dx = \frac{1}{M} \sum_{i=1}^{M} \varphi(x_i) \tag{2.20}$$

that holds for every $\varphi \in \Phi_l$. Since $p \in V_l^{(1)}$, p is a linear combination of basis functions, hence:

$$p = \sum_{(l,i) \in I_l} \alpha_{l,i} \varphi_{l,i} \tag{2.21}$$

Inserting Equation 2.21 into Equation 2.20, the system of linear equations resulting from 2.20 can be formulated:

$$(R + \lambda C)\alpha = b \tag{2.22}$$

With $R_{i,j} = \langle \varphi_i, \varphi_j \rangle_{L_2}$, $C_{i,j} = \langle \Lambda\varphi_i, \Lambda\varphi_j \rangle_{L_2}$ and $b_i = \frac{1}{M}\sum_{j=1}^{M} \varphi_i(x_j)$.
The system matrix R is symmetric and positive definite, C is the regularization matrix, often chosen to be the identity matrix.

2.3.3. Online/Offline Splitting

In Equation 2.22, only the right hand side is affected by the data set. Hence, computing the approximation p can be split into an offline and an online phase. During the offline phase, the heavy lifting necessary to solve the system is done, i.e. decomposing the system matrix R, which has a complexity of $\mathcal{O}(n^3)$. Solving the system for concrete data points and adding new ones, is done completely during the online phase. When R has already been decomposed, the system can be solved by multiplying the inverse matrix (orthogonal decomposition) or forward and backward substitution (Cholesky decomposition). Thus, adding a new data point and updating the model can be done with costs in $\mathcal{O}(n^2)$ [SK09].

2.4. Matrix decompositions

During the offline step, the matrix is decomposed to enable efficient processing of the sample set. Matrix decomposition is a highly researched field in numerical mathematics and there exist many different methods. This thesis considers two matrix decompositions. The orthogonal decomposition into Hessenberg form, which is applicable for symmetric matrices, and the Cholesky decomposition, which additionally requires the matrix to be positive definite. These properties are possessed by the left hand side matrix in sparse grid density estimation (see section 2.3.2). The two methods are implemented in SG++, and are adjusted in the context of this work.

2.4.1. Orthogonal Decomposition into Hessenberg Matrix

The orthogonal decomposition into a Hessenberg matrix (in the following denoted as orthogonal decomposition), is a matrix decomposition similar to the widely used QR-decomposition. The QR-decomposition decomposes a arbitrary matrix $A \in \mathbb{C}^{n \times n}$ into a orthogonal matrix Q and a upper triangular matrix R. The decomposition can be computed using Givens-Rotations, or Householder reflections [DR08].

The orthogonal decomposition uses a similar approach, but is also exploiting the properties of symmetric matrices. For a symmetric matrix $A \in \mathbb{R}^{n \times n}$, an orthogonal matrix Q and a tridiagonal matrix T can computed, s.t.:

$$A = QTQ^T \tag{2.23}$$

The matrix Q is obtained by composing $(n-2)$ Householder reflections H_i. H is also called Householder matrix in the literature. Multiplying H_i to A from the right hand side sets all entries of A under the lower sub-diagonal in the corresponding row i to zero. Hence, multiplying H results in a upper triangular matrix with additional entries on the lower sub-diagonal, which is called Hessenberg matrix. Since A is symmetric, applying the reflections from the right leads to the desired tridiagonal matrix T:

$$H_{n-2} \dots H_1 A H_1 \dots H_{n-2} = T \tag{2.24}$$

Householder matrices are orthogonal and symmetric [DR08]. In addition, compositions of orthogonal matrices are orthogonal again [SK09]. Thus, Equation 2.24 can be transformed into:

$$A = (H_{n-2} \dots H_1)^{-1}\ T\ (H_1 \dots H_{n-2})^{-1} = (H_{n-2} \dots H_1)^T\ T\ (H_1 \dots H_{n-2})^T \tag{2.25}$$

With $Q = (H_1 \dots H_{n-2})$, Equation 2.24 can be transformed into Equation 2.23, by exploiting the symmetry property of H_i:

$$(H_{n-2} \dots H_1)^T TQ^T = (H_1^T \dots H_{n-2}^T) TQ^T = (H_1 \dots H_{n-2}) TQ^T = QTQ^T \tag{2.26}$$

To invert the decomposition, only T has to be inverted:

$$(QTQ^T)^{-1} = (Q^T)^{-1} T^{-1} Q^{-1} = (Q^T)^T T^{-1} Q^T = QT^{-1}Q^T \tag{2.27}$$

For a more detailed description of the orthogonal decomposition, and its application to sparse grid density estimation, [Bos17] is recommended.

2.4.2. Cholesky Decomposition

Since the Cholesky decomposition is widely known and is appearing in most of the introductory literature about numerical mathematics, it is only introduced concisely in this section. For further reading about the method in general, see [SK09] or [DR08]. A symmetric and positive definite matrix A can be decomposed into:

$$A = LL^T \tag{2.28}$$

The matrix L is a lower triangular and is called the Cholesky factor. The decomposition can be computed in $\mathcal{O}(n^3)$ steps. [SK09]
A system of linear equations $Ax = b$ can be solved by decomposing A accordingly and applying forward,

$$Ly = b \tag{2.29}$$

and backward substitution,

$$L^T x = y \tag{2.30}$$

Both forward and backward substitution require quadratic time [SK09].
In [Sie16], the Cholesky decomposition is presented in the context of sparse grid density estimation.

2.5. Permutations

Finally, this chapter covers permutations, which are needed for the approach derived in Chapter 3.
In this context, a permutation is the rearrangement of the sequence of a set of n objects, identified by their indices $i \in [n]$. More formally, a permutation is a bijective mapping $\pi : [n] \mapsto [n]$. Hence, $\pi(i)$ is the new index for the object identified by i in the rearrangement.
Permutations can be applied to matrices, to reorder their rows and columns, respectively. A row or column permutation is done by multiplying a so called permutation matrix $P \in \mathbb{R}^{n \times n}$, where n denotes the number of rows and columns, respectively. For a row permutation π, the elements $P_{i,j}$ of P are given by:

$$P_{i,j} = \begin{cases} 1 & \pi(i) = j \\ 0 & else \end{cases} \tag{2.31}$$

Since the columns and rows of P are the unit vectors $u_1, \dots, u_n$, all columns and rows are pairwise linearly independent. Thus, permutation matrices are orthogonal.
A row permutation $A'_R \in \mathbb{R}^{m \times n}$ of a matrix $A \in \mathbb{R}^{m \times n}$, is obtained by multiplying the corresponding permutation matrix $P \in \mathbb{R}^{m \times m}$ from the left. Hence:

$$A'_R = PA \tag{2.32}$$

For column permutation, the approach can be derived analogously. The permutation matrix P is given by:

$$P_{i,j} = \begin{cases} 1 & \pi(j) = i \\ 0 & else \end{cases} \tag{2.33}$$

Multiplying the permutation matrix $P \in \mathbb{R}^{n \times n}$ from the right, yields a column permutation A'_C:

$$A'_C = AP \tag{2.34}$$

If the matrix A is symmetric, a symmetric permutation A' is attained, by applying the same permutations for rows and columns:

$$A' = PAP^T \tag{2.35}$$

Applying symmetric permutation, conserves the symmetry property. This is exploited in Chapter 3.

3. Theory

When examining the combination scheme introduced in Section 2.7, it can be observed, that the scheme contains multiple anisotropic component grids, that are equal asides from being rotated in space. In Figure 2.7 this is the case for the subspaces corresponding to $u_{3,1}$, $u_{1,3}$ and $u_{2,1}$, $u_{1,2}$, respectively. For sparse grids in higher dimensional spaces and with higher levels, the number of such subspaces increases even more. Figure 3.1 shows the scheme of $V_4^{(1)}$. In this sense similar component grids are $G_{(3,1)}$ and $G_{(1,3)}$, $G_{(4,1)}$ and $G_{(1,4)}$ and finally $G_{(3,2)}$ and $G_{(2,3)}$. Those related pairs are depicted by the same color.
Additionally, the combination scheme yields a second deduction. In dimensions higher than one, some component grids are lower dimensional anisotropic grids embedded into the higher dimensional approximation space. For instance, $G_{(3,1,1)}$ can be obtained from $G_{(3)}$ by adding a basis function of level 1 to each grid point in dimension 2 and 3.
These observations can be used to derive a method to improve the offline step of sparse grid density estimation significantly. The conventional implementation of the offline step builds the corresponding sparse grid density estimation system matrices A and decomposes them for every component grid separately. The result of this chapter will be, that, depending on the decomposition method, the cubic cost for decomposing the system matrix, only has to be spent once for every group of component grids, that can be transformed into each other by applying some kind of reflection. For all other grids in such a group, the cost for constructing the system matrix can be reduced to some degree, by using the permutation method we will derive.
Furthermore, we will show, that the system matrix of a lower dimensional anisotropic grid embedded into a higher dimension, can be constructed by simply multiplying a factor depending on the dimension delta.
In the following sections, we consider a d-dimensional sparse grid density estimation problem. The sparse grid is of level n and the corresponding linear system of equations is:

$$(R + \lambda C)\alpha = b \tag{3.1}$$

Within this chapter, no regularization will be applied, hence, $\lambda = 0$. Thus, the system matrix A is equal to R.

Regularization and adaptivity tasks can be applied afterwards, in a step at the beginning of the online step (see [Sie16], [Bos17] and [Fuc18]).

3.1. Matrix Permutation

3.1.1. Equivalent grids

Let Π_d denote the space of all permutations $\pi : [d] \mapsto [d]$. We define the following equivalence relation for d-dimensional vectors $\vec{u}$ and $\vec{v}$:

$$\vec{u} \sim_\pi \vec{v} \quad \Leftrightarrow \quad \exists \pi \in \Pi_d : \pi(\vec{u}) = \vec{v} \tag{3.2}$$

The implied equivalence classes are given by:

$$[\vec{v}]_{\sim_\pi} = \{\vec{u} \mid \vec{v} \sim_\pi \vec{u}\} \tag{3.3}$$

Equation 3.2 and Equation 3.3 can be extended for anisotropic full grids $G_{\vec{l}}$ and $G_{\vec{l}'}$:

$$G_{\vec{l}} \sim_\pi G_{\vec{l}'} \quad \Leftrightarrow \quad \vec{l} \sim_\pi \vec{l}' \tag{3.4}$$

$$[G_{\vec{l}}]_{\sim_\pi} = \{G_{\vec{l}'} \mid G_{\vec{l}} \sim_\pi G_{\vec{l}'}\} \tag{3.5}$$

Those definitions allow us to formulate the geometrical observations from the beginning of this chapter in a formal manner.
The component grids in the combination scheme can be divided into equivalence classes regarding $\sim_\pi$. It is striking, that the geometrical similar grids from figure 2.7 and figure 3.1 belong to the same equivalence classes.
Whether a component grid belongs to the combination scheme of a sparse grid, is determined by the l_1-norm of its level vector $\vec{l}$. Since permuting a vector does not affect its l_1-norm, all elements of the equivalence class $[G_{\vec{l}}]_{\sim_\pi}$, where $G_{\vec{l}}$ denotes a component grid belonging to the scheme, also belong to the latter.
The result of this section will be, that within the same equivalence class regarding $\sim_\pi$, the system matrix A corresponding to the subproblem on a component grid $G_{\vec{l}}$ can be transformed into the system matrix A' of any subproblem on a component grid $G_{\vec{l}'}$, by applying a symmetric permutation, hence,

$$A' = PAP^T \tag{3.6}$$

where P denotes a permutation matrix.
Within two component grids $G_{\vec{l}}$ and $G_{\vec{l}'}$, with $G_{\vec{l}} \sim_\pi G_{\vec{l}'}$, each point $x_{\vec{l},\vec{i}}$ in $G_{\vec{l}}$ can be

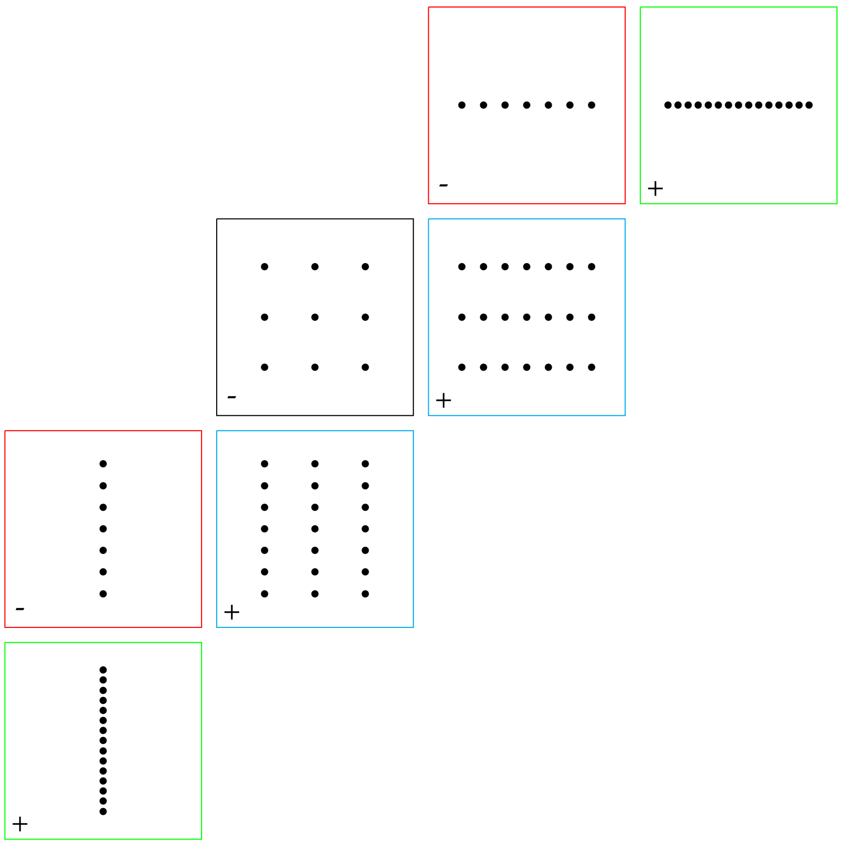


Figure 3.1.: The figure shows the combination scheme of a sparse grid corresponding to $V^(1)_4$. Similar grids are displayed by the same color. The grids $G_{(3,1)}, G_{(1,3)}$ (red), $G_{(4,1)}, G_{(1,4)}$ (green) and $G_{(3,2)}, G_{(2,3)}$ (blue) are rotations, respectivly. Additionaly, $G_{(3,1)}, G_{(1,3)}, G_{(4,1)}, G_{(1,4)}$ are one-dimensional anisotropic full gris embedded into the two-dimensional approximation space.

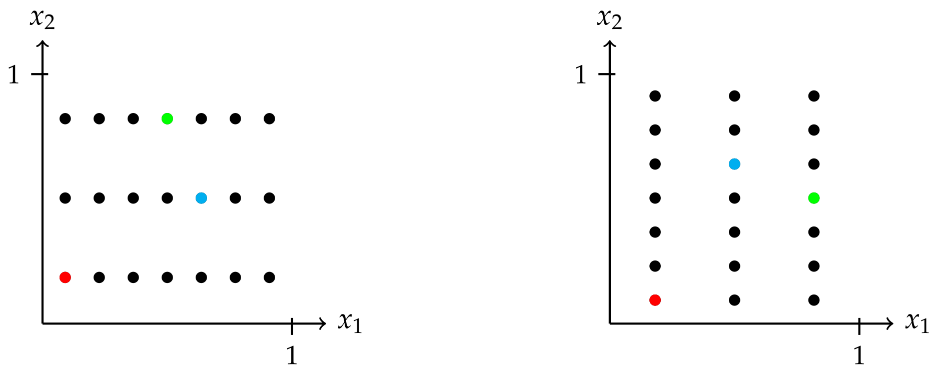


Figure 3.2.: The figure shows corresponding point in the anisotropic full grids $G_{(3,2)}$ (left) and $G_{(2,3)}$ (right). The red points are $x_{(3,1),(1,1)}$ (left) and $x_{(1,3),(1,1)}$ (right), the green ones $x_{(1,2),(1,2)}$ (left) and $x_{(2,1),(2,1)}$ (right) and finally, the blue points represent $x_{(3,1),(3,1)}$ (left) and $x_{(1,3),(1,3)}$ (right).

mapped to its corresponding point $x_{\vec{l}',\vec{i}'}$ in $G_{\vec{l}'}$ by applying π to its level and index vector, respectively. Thus,

$$x_{\pi(\vec{l}),\pi(\vec{i})} = x_{\vec{l}',\vec{i}'} \tag{3.7}$$

For the sake of simplicity, the notation $\pi(\vec{l},\vec{i}) = (\pi(\vec{l}),\pi(\vec{i}))$ will be used in the following. Figure 3.2 shows some corresponding points in $G_{(3,2)}$ and $G_{(2,3)}$. This figure gives an idea about how permuting the level vectors and reordering the grid points accordingly, is indeed some kind of reflection of the grid in a geometrical sense.

3.1.2. Constructing the Permutation Matrix

As explained in Section 2.3.2, the row indices i and column indices j, with $i, j \in [N]$, where N is the total number of grid points, of the matrix R (remember, $A = R$), correspond to basis functions φ_i and φ_j, respectively. Since linear basis functions only have one support point, this implies that they can also be assigned to grid points x_i and x_j, respectively.

For deriving the symmetric permutation displayed in Equation 3.6, we consider row permutation first.

At first, the connection between a row index i and its corresponding grid point x_i needs to be clarified. The i-th row of R contains all L_2-inner-products of φ_i, i.e. the basis function supported by x_i, and all other basis functions (including φ_i).

The idea leading to the row permutation, is to first obtain the level vector $\vec{l}'$ and index

vector $\vec{i'}$ of the point $x_{i'}$ assigned to the row of A' with index i'. To do so, a fixed mapping $h_{\vec{l}} : [N] \mapsto \mathbb{N}^d \times \mathbb{N}^d$ of indices to pairs of level and index vectors is required. The function depends on the level vector $\vec{l}$ of the grid it is defined on. It should be noted, that h can be chosen arbitrarily and depends on the implementation, but it is necessary that such a h exists in order to guarantee the correctness of this method.
The point $x_{\vec{l'},\vec{i'}}$ can now be mapped to the point $x_{\pi(\vec{l'}),\pi(\vec{i'})} = x_{\vec{l},\vec{i}}$, according to the previous chapter. The obtained point $x_{\vec{l},\vec{i}}$ can be assigned to a row index i of the system matrix A on the underlying Grid $G_{\vec{l}}$ by applying $h_{\vec{l}}^{-1}$.
This can be used to formulate P:

$$P_{i,j} = \begin{cases} 1 & j = h_{\vec{l}}^{-1}(\pi(h_{\vec{l'}}(i))) \\ 0 & \text{else} \end{cases} \tag{3.8}$$

The column permutation can be derived analogously, and is given by P^T.
The elements of PAP^T are given by:

$$(PAP^T)_{i,j} = A_{h_{\vec{l}}^{-1}(\pi(h_{\vec{l'}}(i))),h_{\vec{l}}^{-1}(\pi(h_{\vec{l'}}(j)))} = \langle \varphi_{\pi(h_{\vec{l'}}(i))}, \varphi_{\pi(h_{\vec{l'}}(j))} \rangle_{L_2} \tag{3.9}$$

Note that the basis functions on the right hand side are depicted with their corresponding level and index vectors rather than their matrix indices. Consequently, in order to show the correctness of the method, i.e. Equation 3.6, the following equation remains to be proven:

$$\langle \varphi_{\vec{l},\vec{i}}, \varphi_{\vec{\tilde{l}},\vec{\tilde{i}}} \rangle_{L_2} = \langle \varphi_{\pi(\vec{l}),\pi(\vec{i})}, \varphi_{\pi(\vec{\tilde{l}}),\pi(\vec{\tilde{i}})} \rangle_{L_2} \tag{3.10}$$

For d-dimensional linear basis functions $\varphi_{\vec{l},\vec{i}}$ and $\varphi_{\vec{\tilde{l}},\vec{\tilde{i}}}$ and a vector permutation π according to the previous section.
Using the definitions of the L_2-inner-product as well as the basis functions, the left hand side of Equation 3.10 can be transformed into:

$$\int_0^1 \prod_{j=1}^{d} \varphi_{\vec{l}_j,\vec{i}_j}(\vec{x}_j) \prod_{j=1}^{d} \varphi_{\vec{\tilde{l}}_j,\vec{\tilde{i}}_j}(\vec{x}_j) d^d\vec{x} = \int_0^1 \prod_{j=1}^{d} \varphi_{\vec{l}_j,\vec{i}_j}(\vec{x}_j) \varphi_{\vec{\tilde{l}}_j,\vec{\tilde{i}}_j}(\vec{x}_j) d^d\vec{x} \tag{3.11}$$

The permutation π only alters the sequence of the elements of a vector, but does not remove, replace or add such. Therefore, every one dimensional basis function $\varphi_{l,i}$, that appears in the tensor product of a d-dimensional basis function $\varphi_{\vec{l},\vec{i}}$, also appears in the tensor product of all basis functions $\varphi_{\pi(\vec{l},\vec{i})}$.
Considering,

$$\forall j,k \in [d] : \int_a^b \varphi_{l,i}(\vec{x}_j)\varphi_{\tilde{l},\tilde{i}}(\vec{x}_j) d\vec{x}_j = \int_a^b \varphi_{l,i}(\vec{x}_k)\varphi_{\tilde{l},\tilde{i}}(\vec{x}_k) d\vec{x}_k \tag{3.12}$$

Equation 3.11 is equal to:

$$\int_0^1 \prod_{j=1}^{d} \varphi_{\pi(\vec{l})_j,\pi(\vec{i})_j}(\vec{x}_j)\varphi_{\pi(\vec{\tilde{l}})_j,\pi(\vec{\tilde{i}})_j}(\vec{x}_j)d^d\vec{x} = \langle \varphi_{\pi(\vec{l}),\pi(\vec{i})}, \varphi_{\pi(\vec{\tilde{l}}),\pi(\vec{\tilde{i}})} \rangle_{L_2} \tag{3.13}$$

This finishes the proof of Equation 3.10, and thus assures the correctness of the permutation approach.

3.2. Dimension Blow-Up

Within the combination scheme of a sparse grid, some of the component grids are lower dimensional full grids embedded into a higher dimension. A lower dimensional grid's level vector contains elements equal to 1, and all of its grid points are located in an affine subspace parallel to the linear subspace spanned by the standard unit vectors of the dimensions with level vector elements unequal to 1. In the following we will show, that the corresponding sparse grid density estimation system matrix A' of the embedded base grid, can be obtained from the base system matrix A, by the elementwise multiplication of a factor ρ, hence:

$$A' = \rho A \tag{3.14}$$

We denote this process as dimension blow-up, and ρ is called the blow-up factor.
When embedding a component grid into a higher dimension, a base function of level 1 is added in the latter, by adding it to the tensor product of each grid point. From this observation, we can derive the blow-up factor ρ.
Let $G_{\vec{l}}$ denote the d-dimensional base grid, and $G'_{\vec{l}'}$ a $d+1$-dimensional embedding, where $\vec{l}' = (\vec{l}_1, \dots, 1, \dots, \vec{l}_d)$. A and A' denote the corresponding system matrices, respectively. Since $A = R$, an element $A'_{i,j}$ is given by:

$$A'_{i,j} = \langle \varphi'_i, \varphi'_j \rangle_{L_2} = \int_0^1 \varphi'_i(\vec{x})\ \varphi'_j(\vec{x})\ d^{d+1}\vec{x} \tag{3.15}$$

The n-dimensional basis function $\varphi_i(\vec{x})$ is defined as a tensor product of n one-dimensional basis functions:

$$\varphi_i(\vec{x}) = \prod_{k=1}^{n} \varphi_{i_k}(\vec{x}_k) \tag{3.16}$$

Inserting Equation 2.2 into Equation 2.1 yields:

$$\langle \varphi'_i, \varphi'_j \rangle_{L_2} = \int_0^1 \prod_{k=1}^{d+1} \varphi_{i_k}(\vec{x}_k) \prod_{l=1}^{d+1} \varphi_{j_l}(\vec{x}_l)\ d^{d+1}\vec{x} \tag{3.17}$$

W.l.o.g, it is assumed that the added dimension is dimension $d+1$. In the following equation $\vec{y}$ denotes the vector $(\vec{x}_1, \ldots, \vec{x}_n)$ and z denotes $\vec{x}_{d+1}$. Therefore Equation 2.2 can be transformed into:

$$\langle \varphi'_i, \varphi'_j \rangle_{L_2} = \int_0^1 \varphi_{i_{d+1}}(z)\varphi_{j_{d+1}}(z) \int_0^1 \prod_{k=1}^{d} \varphi_{i_k}(\vec{y}_k) \prod_{l=1}^{d} \varphi_{j_l}(\vec{y}_l)\, d^d\vec{y}\, dz \tag{3.18}$$

The inner integral is the L_2-inner-product of the d-dimensional basis functions φ_i, φ_j. Let $\vec{v}_{/1}$ denote the vector obtained by removing all elements of $\vec{v}$ equal to 1. In order to obtain the dimension factor, it is necessary that the following assumption holds for the previously introduced mapping from a grid point's level and index vector to its matrix index:

$$\forall \vec{l}, \vec{l'} \in \mathbb{N}^m \wedge \vec{i}, \vec{i'} \in \mathbb{N}^n : \vec{l}_{/1} = \vec{l'}_{/1} \wedge \vec{i}_{/1} = \vec{i'}_{/1} : h^{-1}(\vec{l}, \vec{i}) = h^{-1}(\vec{l'}, \vec{i'}) \tag{3.19}$$

Hence, level and index vector elements equal to 1 do not affect the index associated with a grid point. Note, that a 1 in the level vector implies a 1 in the corresponding index vector at the same position. Using this assumption, the inner integral from Equation 3.18 is equal to the base system matrix element $A_{i,j}$:

$$\langle \varphi'_i, \varphi'_j \rangle_{L_2} = \int_0^1 \varphi_{i_{d+1}}(z)\varphi_{j_{d+1}}(z) dz A_{i,j} \tag{3.20}$$

Thus, the blow-up factor ρ is given by:

$$\rho = \int_0^1 \varphi_{i_{n+1}}(z)\varphi_{j_{n+1}}(z)\, dz \tag{3.21}$$

If k level vector elements equal to 1 are added, the factor ρ^k has to be multiplied. Hence, the factor increases or decreases exponentially with respect to the number of added elements, if $\rho \neq 1$. This results in numerical problems described in Chapter 5. Note, that this is a property of sparse grid density estimation with the combination technique in general, and not of the blow-up method developed in this thesis. Therefore, the elements of a corresponding system matrix R increase or decrease with a growing number of elements equal to 1.
The value of ρ depends on the basis function used. We consider linear and modlinear basis functions.

3.2.1. Linear Basis Function

The linear basis function of level 1,

$$\varphi_{\text{lin}} = \max\{1 - |2x - 1|,\ 0\}$$

can be represented as:

$$\varphi_{\text{lin}} = \begin{cases} 2x & x \in [0,\ \frac{1}{2}] \\ 2(1-x) & x \in (\frac{1}{2},\ 1] \\ 0 & \textit{else} \end{cases} \tag{3.22}$$

This is shown in Figure 3.3 geometrically. Inserting Equation 3.22 into Equation 3.21 yields:

$$\rho_{\text{lin}} = \int_0^{\frac{1}{2}} 4x^2\, dx + \int_{\frac{1}{2}}^1 4(1-x)^2\, dx = \frac{1}{3} \tag{3.23}$$

3.2.2. Modlinear Basis Function

In the modlinear case the basis function of level 1 is (see Figure 3.3):

$$\varphi_{\text{modlin}} = \begin{cases} 1 & x \in [0,\ 1] \\ 0 & \textit{else} \end{cases}$$

Therefore, the blow-up factor ρ_{modlin} is trivial:

$$\rho_{\text{modlin}} = \int_0^1 1^2\, dx = 1 \tag{3.24}$$

3.3. Application to decomposed System Matrices

So far, we have only showed, that system matrices A of subproblems on equivalent component grids can be transformed into each other using the permutation approach. But within the offline step of sparse grid density estimation, the most time consuming part is to decompose the system matrix in order to speed up the online step drastically. Consequently, we are highly interested in applying the same approach to already decomposed matrices. This would avoid the need to compute the decomposition expensively for each subgrid. Hence, this section examines the applicability of the permutation and blow-up method to the matrix decomposition considered in this thesis, i.e. orthogonal and Cholesky decomposition.

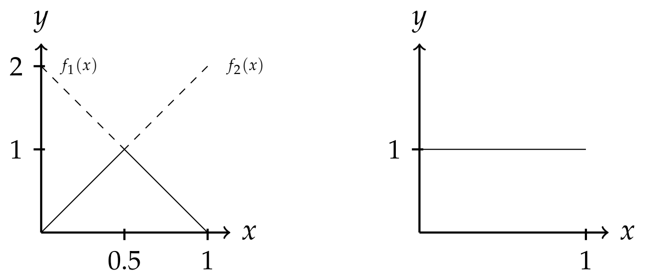


Figure 3.3.: The figure shows the linear (right) and modlinear one-dimensional basis functions. For the linear basis function, the dashed lines represent the functions $f_1(x) = -2x + 2$ and $f_2(x) = 2x$, used to calculate the integral value.

3.3.1. Orthogonal Decomposition

Suppose we apply the symmetric permutation to a system matrix A which has been decomposed into the orthogonal decomposition, hence:

$$PAP^T = PQTQ^TP^T = A' \tag{3.25}$$

Since, permutation matrices are orthogonal, the matrix $Q' = PQ$, which is obtained by applying the permutation to the rows of Q, is a composition of orthogonal matrices and likewise orthogonal. Consequently, $Q'T(Q')^T$ is an orthogonal decomposition of A'. It should be noted, that the decomposition resulting from this method is not necessarily identical to the one computed by the algorithm in Section 2.4.1.
The inverse of A' can be computed as follows:

$$(A')^{-1} = (Q')^T T^{-1} Q' \tag{3.26}$$

If T has already been inverted, no recomputation is necessary after the permutation.
If a dimension blow-up is necessary, it is most efficient to multiply the dimension factor to T and its inverse value to T^{-1}, respectively. Adjusting Q would require an additional square root.

3.3.2. Cholesky Decomposition

While transforming an orthogonal decomposition is rather easy, to apply the approach to a Cholesky decomposition is more complicated.

Suppose we again apply the symmetric decomposition to a decomposed system matrix A, but this time the matrix is decomposed into the Cholesky decomposition. Hence,

$$PAP^T = PLL^TP^T = BB^T = A' \tag{3.27}$$

where $B = PL$ is a row permutation of the Cholesky factor.
So far, we can exploit the symmetric properties of the Cholesky decomposition, to obtain a new factor B. But since L was in lower triangular form, and B is a row permutation of L, B is most likely not in lower triangular form anymore, unless P is the identity matrix. Even though forward and backward substitution could still be applied in theory, this would require modifications of existing implementations of the Cholesky decomposition and some extra effort when executing them, i.e. in the online step of sparse grid density estimation.
Due to this, it would be desirable to transform B into a lower triangular form, if this is possible with reasonable effort.
In [Sie16] a method is introduced, to compute a orthogonal transformation U, s.t. $BU = L'$ is in lower triangular form again. The approach used is similar to the computation of a QR-decomposition using Givens-Rotations [DR08].
The idea is, to set all nonzero elements $x_1, \ldots, x_n$ above the diagonal of B to 0, by applying Givens-Rotations U_i successively. The final orthogonal transformation matrix U is then given by:

$$U = U_1U_2 \ldots U_n \tag{3.28}$$

Since U is orthogonal, applying U to B yields a proper Cholesky factor L' of A':

$$L'(L')^T = BUU^TB^T = BIB^T = BB^T = A' \tag{3.29}$$

The number of nonzero elements above the diagonal is in $\mathcal{O}(n^2)$ in the worst case. Since a Givens-Rotation alters $2n$ elements at most, the worst case running time of this method is in $\mathcal{O}(n^3)$. This result is somewhat disappointing, because the complexity class does not change in comparison to the conventional Cholesky method. But the approach could still yield a performance improvement in practice, if the number of nonzero elements is low.

4. Implementation

The theoretical approach developed in Chapter 3, has been implemented in SG++ as part of this thesis. This chapter describes the implemented algorithms, analyzes their running time complexities and introduces the changes made to the library.
Due to the more promising results from Chapter 3, the approach has only been implemented for the orthogonal decomposition. But permutation and dimension blow-up are abstracted into a superclass, and thus reusable for future implementations of other decompositions such as Cholesky.

4.1. Grid point Numbering h in SG++

As stated in Chapter 3, an implementation dependent mapping $h_{\vec{l}} : [N] \mapsto \mathbb{N}^d \times \mathbb{N}^d$ from row and column indices of the system matrix to level and index vectors of grid points, is required by the introduced approach. In SG++ this function is implemented implicitly as part of the generation of the underlying grid, in the class *HashGridGenerator*. Indices are assigned iteratively, with one iteration per dimension of the level vector of $G_{\vec{l}}$. The numbering starts of in the first dimension, i.e. for each point with level and index vector elements equal to 1 in each dimension other than the first.
Indices are assigned according to the level and index of the point in dimension 1. For instance, grid point $x_{\mathbb{1}^T,\mathbb{1}^T}$ corresponds to index 1, $x_{(2,\mathbb{1}^T),\mathbb{1}^T}$ to index 2 and $x_{(3,\mathbb{1}^T),(3,\mathbb{1}^T)}$ to index 5. Formally, the numbering h_1 in the first dimension is given by:

$$h_1(i) = ((\lfloor \log_2(i) \rfloor, \mathbb{1}^T),\ (2(i - 2^{\lfloor \log_2(i) \rfloor}) - 1, \mathbb{1}^T))$$

And the inverse h_1^{-1}:

$$h_1^{-1}(\vec{l}, \vec{i}) = 2^{\vec{l}_1 - 1} + \frac{\vec{i}_1 + 1}{2}$$

After the first step, each point with a level vector $(l, \mathbb{1}^T)$ and an index vector $(i, \mathbb{1}^T)$ has an index assigned to it.
In the next iteration, all points $x_{(\vec{l},\vec{i})}$ with $\vec{l} = (l_1, l_2, \mathbb{1}^T)$ and $\vec{i} = (i_1, i_2, \mathbb{1}^T)$ are handled.

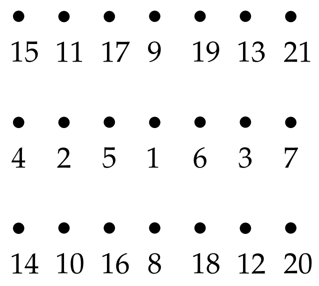


Figure 4.1.: This figure shows the grid points of $G_{(3,2)}$, with their corresponding system matrix index (below), according to the numbering method used in SG++.

This is done by numbering the one-dimensional subgrids in the second dimension containing points from the first iteration, in order of the indices of those points. The numbering of the one-dimensional subgrids follows the same approach used for the first dimension, with the slight difference that the point with $l_2, i_2 = 1$, i.e. the point already numbered, has no new index assigned.
In the following steps this exact process is repeated for all already numbered points of all previous iterations, in the dimension of the current iteration. Note, that the assumption made in 3.19 holds, because dimensions with a grid level of 1 can be skipped, since no points have new indices assigned. Therefore, such dimensions do not affect the numbering.
The principal is illustrated in Figure 4.1 for $G_{(3,2)}$. The indices $1 - 7$ are assigned to the subgrid in the first dimension. All second dimension subgrids are then numbered in order of the index of the grid point in the first dimension. If the grid had a third dimension, the third dimension subgrids would be numbered the exact same way, starting with the subgrid containing point 1.
The derived function h is not implemented explicitly. The permutation algorithm introduced in Section 4.2 uses the same iteration pattern described previously, and therefore implements the function implicitly. Hence, we forego a explicit definition.
The inverse function h^{-1} is required by the permutation implementation. Consequently, we will describe the function in detail and its implementation in more detail.
In the following $\vec{u}$ denotes the level vector of the grid $G_{\vec{u}}$, whereas $\vec{l}$ refers to the level vector of a grid point $x_{\vec{l},\vec{i}} \in G_{\vec{u}}$. The number l^* denotes the biggest index among elements of $\vec{l}$ unequal to 1. From the numbering algorithm used in SG++, $h_{\vec{u}}^{-1}$ can be formulated recursively, depending on the underlying grid $G_{\vec{u}}$:

$$h_{\vec{u}}^{-1}(\vec{l},\vec{i}) = \begin{cases} \prod_{x=1}^{l^*-1}(2^{\vec{u}_x}-1) + \\ (h_{\vec{u}}^{-1}((\vec{l}_1,\dots,\vec{l}_{l^*-1},\mathbb{1}^T),(\vec{i}_1,\dots,\vec{i}_{l^*-1},\mathbb{1}^T)) - 1)(2^{\vec{u}_{l^*}}-2) + \\ (2^{\vec{l}_{l^*}-1}-2) + (\vec{i}_{l^*}+1)/2 & l^* > 1 \\ (2^{\vec{l}_{l^*}-1}-1) + (\vec{i}_{l^*}+1)/2 & l^* = 1 \end{cases} \tag{4.1}$$

Let d denote the dimension of $\vec{l},\vec{i}$ and $\vec{u}$, respectively. The function is implemented iteratively and is shown in pseudo code in Algorithm 4.1.
The complexity class for each not trivially constant part of the algorithm is noted on the right side. The function *computeLStar()* determines l^*, by iterating over $\vec{l}$ descendingly, starting at its biggest index. Hence, the function takes at most d steps. One step of the second occurrence of *computeLStar()* reduces the amount of iterations of the while loop by one and vice versa. Therefore, the two complexities can be combined and the second occurrence is treated as if it was constant in the further analysis.
The first summand of the recursive representation of h^{-1} in Equation 4.1 only depends on the level vector $\vec{u}$ of the underlying grid. Since *hInverse()* is called repeatedly during permutation, it makes sense to precompute and store the terms, before invoking the permutation process. Therefore, *getPreComputed()* fetches the correct precomputed term in constant time.
Overall, the computational cost of the algorithm is in $\mathcal{O}(d)$.

4.2. Permutation Implementation

The implementation of the permutation approach follows the method introduced in Section 3.1.2. But instead of computing the permutation matrix explicitly and applying the permutation by matrix multiplication, the desired system matrix A' corresponding to a component grid $G_{\vec{v}}$ is constructed by copying the correct row/column from the base matrix A corresponding to the grid $\vec{u}$. Note, that we assume that both level vectors don't contain elements equal to 1. If this is not the case, the elements can simply be dropped and compensated during the dimension blow-up step. In the following d denotes the dimension of $\vec{u}$ and $\vec{v}$, respectively.
We will consider row permutation, column permutation is implemented analogously. The algorithm is depicted in Algorithm 4.2. The desired system matrix A' is constructed iteratively, analogous to the numbering method described in Section 4.1. The algorithm constructs the level and index vectors successively, in order of the matrix indices of their corresponding points. Hence, in iteration k, the level and index vector $h_{\vec{v}}(k)$ is constructed. The obtained level and index vector then gets permuted, and $h_{\vec{u}}^{-1}$ is

Algorithm 4.1.: Algorithm for h^{-1}

```
1: function hInverse($\vec{l}, \vec{i}, \vec{u}$)
2:     $r \leftarrow 1$
3:     $m \leftarrow 1$
4:     $l^* \leftarrow computeLStar(d)$                                  ▷ $\mathcal{O}(d)$
5:     while $l^* \geq 1$ do                                             ▷ $\mathcal{O}(d)$
6:         if $l^* = 1$ then
7:             $r \leftarrow (2^{\vec{l}_0 - 1} - 2 + (\vec{i}_0 + 1)/2) * m$
8:             $l^* \leftarrow -1$
9:         else
10:            $p \leftarrow getPreComputed(l^* - 1)$                     ▷ $\mathcal{O}(1)$
11:            $r \leftarrow r + (2^{\vec{l}_{l^*} - 1} - 3 + (\vec{i}_{l^*} + 1)/2) * m$
12:            $m \leftarrow m * (2^{\vec{u}_{l^*}} - 2)$
13:            $l^* \leftarrow computeLStar(l^* - 1)$                     ▷ $\mathcal{O}(1)^*$
14:        end if
15:    end while
16: end function
```

applied, using the algorithm introduced in Section 4.1. Furthermore, the row of the base system matrix A, that corresponds to the obtained index, is copied to row k of A'. The running time complexities of the interesting parts are again denoted on the right side. The precomputation has been discussed in Section 4.1. Furthermore, exactly n elements need to be copied in order to copy a row of A.
Since the algorithm constructs each point of the underlying grid successively, the innermost for loop is looped through exactly n times, which leads to the computational cost denoted at the outermost loop.
The function *computePermutation()* computes a proper permutation π with $\pi(\vec{v}) = \vec{u}$ and stores it in an array, where the index of the array corresponds to the vector index of $\vec{u}$ and the value at this index represents the index of $\vec{v}$. The algorithm used to compute the permutation, iterates over $\vec{u}$ and searches for a matching elements in $\vec{v}$, and therefore needs $\mathcal{O}(d^2)$ steps to terminate.
Hence, the running time complexity of Algorithm 4.2 is given by:

$$\mathcal{O}(d^2) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$$

This is correct due to the assumption, that $\vec{u}$ and $\vec{v}$ have no element equal to 1. Thus, d is in $\mathcal{O}(n)$.

4.3. Application to the Orthogonal Decomposition

The SG++ implementation of the offline object using the orthogonal decomposition stores the matrices Q and T^{-1}. The algorithm introduced in the previous section is used to permute the rows of Q. In the second step, the dimension blow-up is applied, by multiplying the inverse blow-up factor elementwise to T^{-1}. Therefore, the computational cost for the transformation is in $\mathcal{O}(n^2)$.

4.4. Integration into SG++ and the Data Mining Pipeline

For a general description of the SG++ library see [Pfl10b] and for a more detailed description of the data mining pipeline and the implementation of the combination technique in the library, [Rös19] is recommended.
The different offline objects in SG++ inherit from an abstract superclass *DBMatOffline*. The class has child classes for different decompositions and other solving techniques not covered in this thesis. The only one modified is *DBMatOfflineOrthoAdapt* for the orthogonal decomposition.
As part of this work, a new abstract child class *DBMatOfflinePermutable* has been implemented. The class provides the following protected methods:

- *preComputeSummands()*: precomputations for the inverse numbering function
- *computePermutation()*: stores the permutation of two vectors in an array
- *getMatrixIndexForPoint()*: inverse numbering function
- *permutateMatrix()*: permutes given matrix using the permutation algorithm
- *dimensionBlowUp()*: implementation of the dimension blow-up

Those methods can be uses to implement the approach for further decomposition types. Furthermore, the class provides a abstract method *permutateDecomposition()*, that has to be implemented by child classes. As of now, the only child class of *DBMatOfflinePermutable* is *DBMatOfflineOrthoAdapt*.
Additionally, the model fitting component for the combination technique, which is implemented in the class *ModelFittingDensityEstimationCombi*, has been adjusted. When methods with online / offline splitting are configured, the combination fitter aggregates one *ModelFittingDensityEstimationOnOff* object for each component grid. The class

ModelFittingDensityEstimationOnOff wires the online and offline object corresponding to a grid. It provides functionality to fit data sets and to perform different adaptivity tasks. To make use of the permutation technique, a class *ObjectStore* has been implemented, that can be passed to the constructor of *ModelFittingDensityEstimationCombi*, which passes the object to its instances of *ModelFittingDensityEstimationOnOff*. When a component initially has to build and decompose its offline matrix, it tries to fetch a base object from the store first. If a suitable base object is present, the component fitter copies the object and calls *permutateDecomposition()*. Otherwise, the matrix is built and decomposed from scratch, to be stored.

Algorithm 4.2.: Permutation algorithm

```
1: function $permutateMatrix(\vec{u}, \vec{v}, A, d)$
2:     $computePermutation()$                                   $\triangleright\ \mathcal{O}(d^2)$
3:     $preCompute(\vec{u})$                                    $\triangleright\ \mathcal{O}(d)$
4:     $A'_1 \leftarrow copyRow(A_1)$                           $\triangleright\ \mathcal{O}(n)$
5:     $X \leftarrow \{x_{\mathbb{1},\mathbb{1}}\}$
6:     $r \leftarrow 2$
7:     for $k \leftarrow 1, d$ do                               $\triangleright\ \mathcal{O}(n)$
8:         for $x_{\vec{l},\vec{i}} \in X$ do
9:             $first \leftarrow$ true
10:            for $l \leftarrow 1, \vec{v}_k$ do
11:                for $i \in \{1, 3, \ldots 2^l - 1\}$ do
12:                    if $first =$ true then
13:                        $first \leftarrow$ false
14:                    else
15:                        $\vec{l'} \leftarrow (\vec{l}_1, \ldots, \vec{l}_{k-1}, l, 1, \ldots)$
16:                        $\vec{i'} \leftarrow (\vec{i}_1, \ldots, \vec{i}_{k-1}, i, 1, \ldots)$
17:                        $X \leftarrow X \cup x_{\vec{l'},\vec{i'}}$
18:                        $l_\pi = permutate(\vec{l'}, \vec{u}, \vec{v})$           $\triangleright\ \mathcal{O}(d)$
19:                        $i_\pi = permutate(\vec{i'}, \vec{u}, \vec{v})$           $\triangleright\ \mathcal{O}(d)$
20:                        $r' \leftarrow hInverse(l_\pi, i_\pi, \vec{u})$           $\triangleright\ \mathcal{O}(d)$
21:                        $A'_r \leftarrow copyRow(A_{r'})$                         $\triangleright\ \mathcal{O}(n)$
22:                        $r \leftarrow r + 1$
23:                    end if
24:                end for
25:            end for
26:        end for
27:    end for
28: end function
```

5. Evaluation

In this chapter, we want to analyze the performance improvement of the new implementation of the offline step introduced in Chapter 4 compared to the old implementation in the SG++ project, as well as the error, i.e. the difference of the computed results, introduced by the new approach.
Therefore, different scenarios are examined. At first, only anisotropic component grids are considered, to verify the quadratic complexity of the approach derived in the implementation chapter, compared to the cubic complexity of decomposing the system matrices. Furthermore, we will show, that the introduced error is negligible. Note that error in this context means the deviation from the results computed by using conventional decomposition. The performance and error tests are examined for permutation and the dimension blow-up in isolation.
Finally, we compare the performance of the combination grid approach on a sparse grid of level n and dimension d, using the new implementation, compared to the conventional implementation.

5.1. Anisotropic Component Grids

This section compares the practical running times of building and decomposing a system matrix and obtaining it by permutation and dimension blow-up from an existing base grid. Furthermore, the error introduced by the new approach is examined. For this measurements, the results from permutation and dimension blow-up are evaluated in isolation. The combined scenario did not yield new effects and is therefore omitted in this section.

5.1.1. Method

To compare the two implementations, evaluation data sets have been generated, using the following approach.
At first, a set of base level vectors X has been chosen. The chosen vectors $\vec{v} \in B$ do not contain elements equal to 1. For each such $\vec{l}$, a set of objective vectors $\vec{l'} \in Y_X$ has been generated, depending on the corresponding scenario. For the blow-up only scenario, the objective vectors are gathered by inserting a specified number of 1 elements, while

conserving the sequence of elements of $\vec{l}$. For the permutation scenario different permutations of the base vectors are considered.
Hence, the scenario dependant sample set S consists of pairs of base vectors and objective vectors. For each pair, the offline object on the component grid $G_{\vec{l}}$ is built and decomposed. Subsequently, the decomposed offline object corresponding to $G_{\vec{l}'}$ is constructed, by building and decomposing it from scratch, and by transforming the base object. The computation time of both methods is measured, which is the time metric examined in the following sections. Note, that the time spent to build the base object is not included, we want to simulate the scenario where a base object already exists.
In the next step, two online objects are built from the two versions of the objective offline object. Those are used to compute two surplus vectors $\vec{\alpha}$ and $\vec{\alpha'}$, by solving the linear system for a randomly generated set of data points.
Finally, for the error analysis the mean relative *mre* and mean total error *mte*,

$$mte = \frac{1}{n}\sum_{i=1}^{n} |\vec{\alpha}_i - \vec{\alpha'}_i| \quad mre = \frac{1}{n}\sum_{i=1}^{n} \frac{|\vec{\alpha}_i - \vec{\alpha'}_i|}{\vec{\alpha}_i}$$

where n denotes the number of grid points, are determined from the surplus vectors.

5.1.2. Dimension Blow-Up only

The first scenario we will consider, is the one where only the dimension blow-up factor has to be applied. Thereby, we can analyse the effects of multiplying the factor in isolation.
Permutation is not necessary, if the level vectors corresponding to the underlying grids, have equal sequences of elements unequal to 1. As explained in Chapter 3, system matrices on such grids can be transformed into each other, by multiplying the dimension blow-up factor.
In the following Δ_d denotes the difference in dimension between a base grid and a objective grid, i.e. the number of dimensions with level 1.
For linear basis functions, the dimension permutation factor ρ is

$$\frac{1}{3^{\Delta_d}} \quad \text{and} \quad 3^{\Delta_d} \text{ for inverse matrices.}$$

In practice, the exponential growth of ρ implies a upper bound for Δ_d, due to the maximum value of floating point numbers with a fixed length. In C++, a double has a maximum value of 1.8×10^{308}, which approximately corresponds to 3^{650}. During evaluation, the maximum feasible value for Δ_d was roughly around 450. Even though

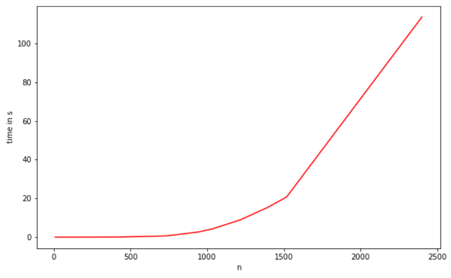
Table 5.1.: Error metrics dependent on the number of grid points n

n	avg(mre)	avg(mte)	max(mre)	max(mte)
$n < 500$	10^{-8}	10^{22}	10^{-7}	10^{24}
$500 \leq n < 1000$	10^{-10}	10^{16}	10^{-9}	10^{18}
$1000 \leq n < 1500$	10^{-5}	10^{23}	10^{-4}	24
$n \geq 1000$	10^{-5}	10^{23}	10^{-5}	10^{24}

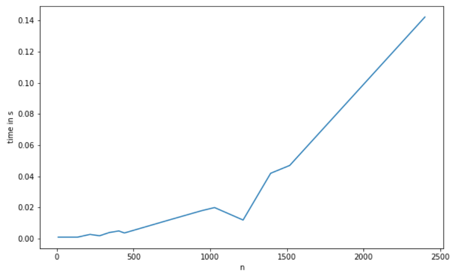
this boundary can be stretched by using larger floating point implementations, the exponential growth will catch up eventually. Furthermore, using larger floating point numbers increases the memory usage drastically. As stated in Chapter 3, this is not a problem of the blow-up method, but of sparse grid density estimation with linear basis functions in general. When using modlinear basis functions, no dimension blow-up is necessary and thus, the problem is avoided.
For the analysis, we first evaluate the error resulting from applying the blow-up method. In general, it turned out, that the error is influenced by Δ_d and the the number of grid points n.
Table 5.1 shows the average and maximum mse and mte depending on n. For higher values of n, the relative error increases slightly, but is still insignificant even for large component grids. Note, that in practice component grids are usually much smaller than what was considered for this analysis. For instance, the grid corresponding to the level vector $(3,3,3,3)$ with 2041 grid points, appears in the combination scheme of a regular sparse grid of level 12.
The astronomically high total error numbers seem to be concerning, but those can be relativised, when considering the exponential growth of ρ and the resulting impact of rounding errors. This is shown in Table 5.2, which displays the error metrics dependant on Δ_d. Even though the total error increases exponentially with increasing Δ_d, the relative error is unaffected.
The running time results confirm the theoretical improvement from cubic time for the matrix decomposition, to quadratic time for the elementwise multiplication of the blow-up factor. Figure 5.1 shows plots of the total running time of both methods dependant on the number of grid points n, and illustrated the asymptotic running time behaviour of both methods nicely. The dimension difference Δ_d has no significant effect on the running time.

Table 5.2.: Error metrics dependent on the dimension difference Δ_d

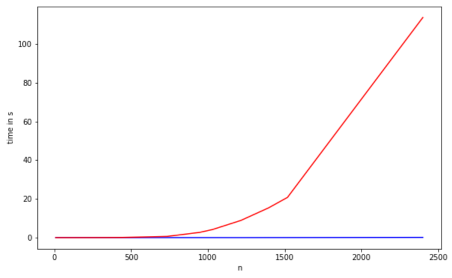
Δ_d	avg(mre)	avg(mte)	max(mre)	max(mte)
$\Delta_d < 50$	10^{-6}	10^{-7}	10^{-4}	10^{-5}
$50 \leq \Delta_d < 100$	10^{-6}	10^{-2}	10^{-4}	10^{0}
$100 \leq \Delta_d < 200$	10^{-6}	10^{0}	10^{-4}	10^{2}
$\Delta_d \geq 200$	10^{-6}	10^{23}	10^{-4}	10^{24}



(a) Old implementation.



(b) New implementation.



(c) Combined plot.

Figure 5.1.: This figure shows the plots of the runnig time evaluations for the blow-up only scenario of the old and new implementation. The plots show the number of grid points n on the x-axis and the running time in seconds on the y-axis. Plot (a) depictes the results for the old implementation and plot (b) represents the new implementation. Plot (c) shows the running times in comparison with the old implementation in red and the new one in blue.

Table 5.3.: Error metrics depending on the number of grid points n for linear basis functions

n	avg(mre)	avg(mte)	max(mre)	max(mte)
$n < 500$	0	0	0	0
$500 \leq n < 1000$	10^{-11}	10^{-12}	10^{-11}	10^{-12}
$1000 \leq n < 1500$	10^{-5}	10^{-7}	10^{-4}	10^{-6}
$n \geq 1500$	10^{-11}	10^{-11}	10^{-10}	10^{-10}

Table 5.4.: Error metrics depending on the number of grid points n for modlinear basis functions.

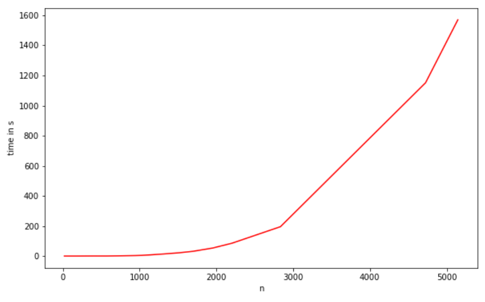
n	avg(mre)	avg(mte)	max(mre)	max(mte)
$n < 500$	10^{-11}	10^{-9}	10^{-10}	10^{-8}
$500 \leq n < 1000$	10^{-10}	10^{-8}	10^{-10}	10^{-7}
$1000 \leq n < 1500$	10^{-10}	10^{-8}	10^{-9}	10^{-7}
$n \geq 1500$	10^{-9}	10^{-7}	10^{-8}	10^{-6}

5.1.3. Permutation only

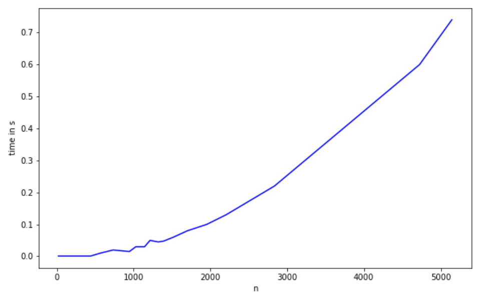
In the next step, we examine scenarios where only permutation has to be applied, i.e. where the level vectors of the base and objective grids are permutations of each other. Both linear and modlinear basis functions are considered.

For both basis functions, the total and relative error numbers increase very slightly with growing values of n, but even for larger grids, the error is insignificant. Table 5.3 and Table 5.4 show the order of magnitude for the error metrics for the two basis functions, for different intervals of n.

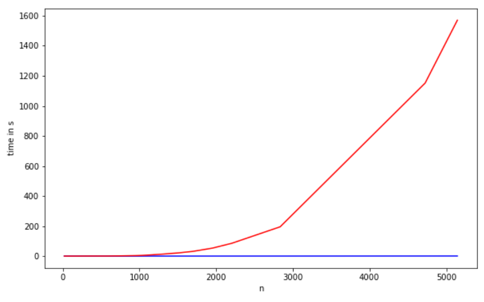
The running time analysis again confirms the theoretical results regarding running time complexity of the approach from Chapter 4. The asymptotic running time for the matrix permutation is quadratic, compared to the cubic one for matrix decomposition. Figure 5.2 and Figure 5.3 compare the running times for linear and modlinear basis functions, and illustrate this clearly.



(a) Old implementation.

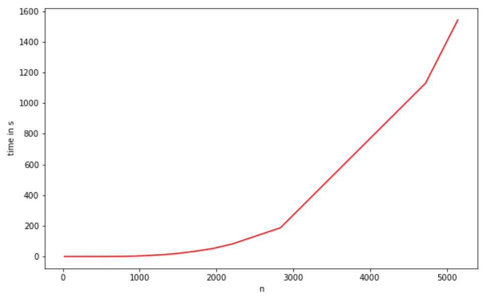


(b) New implementation.

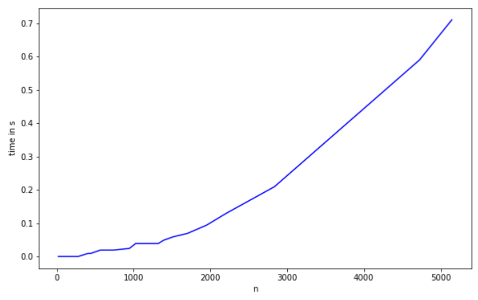


(c) Combined plot.

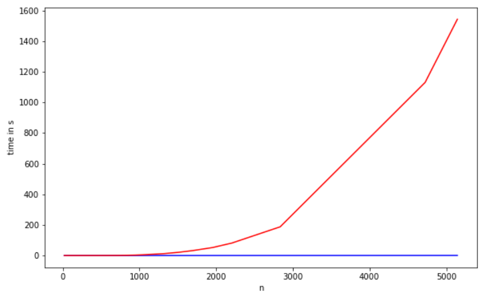
Figure 5.2.: This figure shows the plots of the runnig time evaluations for the permutation only scenario with linear basis functions of the old and new implementation. The plots show the number of grid points n on the x-axis and the running time in seconds on the y-axis. Plot (a) depictes the results for the old implementation and plot (b) represents the new implementation. Plot (c) shows the running times in comparison with the old implementation in red and the new one in blue.



(a) Old implementation.



(b) New implementation.



(c) Combined plot.

Figure 5.3.: This figure shows the plots of the runnig time evaluations for the permutation only scenario with modlinear basis functions of the old and new implementation. The plots show the number of grid points n on the x-axis and the running time in seconds on the y-axis. Plot (a) depictes the results for the old implementation and plot (b) represents the new implementation. Plot (c) shows the running times in comparison with the old implementation in red and the new one in blue.

5.2. Complete Combination Scheme

Finally, we want to evaluate the permutation and dimension blow-up method applied to a density estimation problem on a regular sparse grid using the combination technique, implemented as described in Chapter 4.

5.2.1. Method

First, we introduce the method used for evaluation. For the comparison between the old and new implementation, the combination technique model fitter implemented in the class *ModelFittingDensityEstimationCombi*, is applied to a randomly generated data set S with $|S| = 1000$, using the permutation approach, and using the conventional implementation. Again, both linear and modlinear basis functions are considered. For the running time comparison, the CPU time of both approaches is measured. Subsequently, the two fitted models are evaluated in $n = 1000$ randomly generated points. The results, i.e. the values of the density function at the evaluation points, are stored in two data sets X, corresponding to the old implementations, and Y, corresponding to the new implementation. Those sets are used to determine the mean relative error *mre* and the mean total error *mte*:

$$mte = \frac{1}{n}\sum_{i=1}^{n}|X_i - Y_i| \quad mre = \frac{1}{n}\sum_{i=1}^{n}\frac{|X_i - Y_i|}{X_i}$$

In the following, the two fitter implementations are evaluated for sparse grids with different level l and dimension d, using this method.

5.2.2. Error Analysis

Table 5.5 and Table 5.6 show the error metrics for both basis functions. The tables show, that both total and relative error, have been in a order of magnitude smaller than 10^{-10} for nearly all test runs. Due to the results of the isolated error analysis, this can be explained by the comparably small grid sizes of the occurring component grids, and the resulting small component errors. Hence, the permutation and blow-up method introduces no relevant error compared to the conventional method, atleast within the considered level range. Furthermore, higher dimensions are beneficial regarding the introduced error, because more small component grids are involved in the combination scheme. Therefore, considering the results from the isolated analysis, these smaller grids compensate higher error numbers by larger component grids.

Table 5.5.: Level dependant error metrics for linear basis functions.

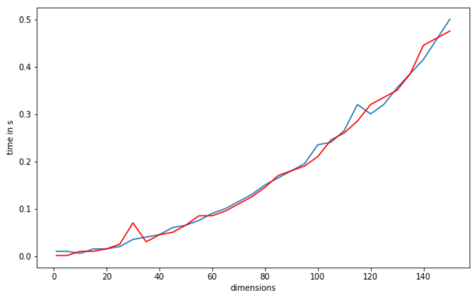
l	avg(mre)	avg(mte)	max(mre)	max(mte)
2	10^{-15}	10^{-17}	10^{-14}	10^{-15}
3	10^{-14}	10^{-14}	10^{-14}	10^{-13}
4	10^{-14}	10^{-13}	10^{-13}	10^{-12}
5	10^{-13}	10^{-11}	10^{-12}	10^{-11}
6	10^{-12}	10^{-10}	10^{-12}	10^{-9}

Table 5.6.: Level dependant error metrics for modlinear basis functions.

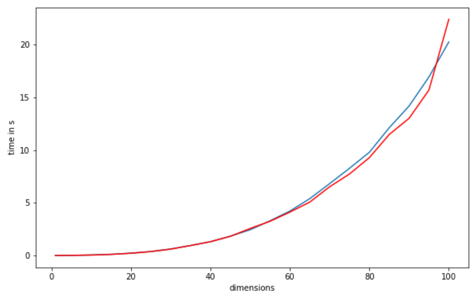
l	avg(mre)	avg(mte)	max(mre)	max(mte)
2	10^{-15}	10^{-17}	10^{-14}	10^{-15}
3	10^{-14}	10^{-14}	10^{-14}	10^{-13}
4	10^{-14}	10^{-13}	10^{-13}	10^{-12}
5	10^{-13}	10^{-11}	10^{-12}	10^{-11}
6	10^{-12}	10^{-10}	10^{-12}	10^{-9}

5.2.3. Running Time Analysis

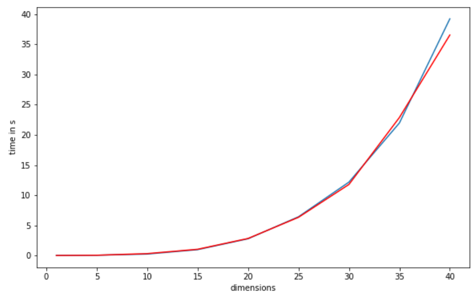
The total running times for both methods are depicted in Figure 5.4 for linear basis functions and in Figure 5.5 for modlinear basis functions. For regular sparse grids with levels up to 4, the results are somewhat disappointing. The plots show, that for combination schemes corresponding to such low level grids, the performance is identical if not slightly worse than the conventional implementation. This can be explained by the small sizes of the component grids in those schemes. Therefore, the cost for decomposing the component system matrices is small, and the overhead of the implementation of the permutation method, i.e. searching the object store for a suitable base object, copying and transforming it, outweighs the theoretical benefits. For instance, in the combination scheme corresponding to the regular sparse grid of level 2, the largest sub grid has 3 points, for level 3 the maximum number of points is 9. Considering, that C++ libraries for matrix decompositions are highly optimized, these number of grid points are to small to benefit from the improved asymptotic running time.
However, for level 5 the new implementation is consistently faster, and the performance improvement increases in higher dimensions. For level 6, this effect becomes more apparent. As mentioned, for higher levels, the component grids increase in size. Thus, the running time savings resulting from the new method can be expected to increase even more the larger the level gets.



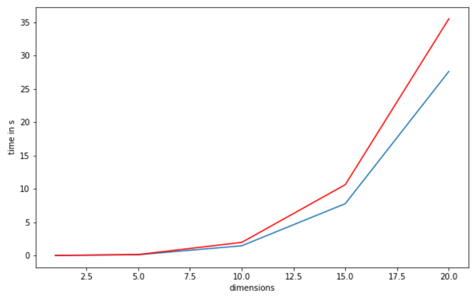
(a) $l = 2$



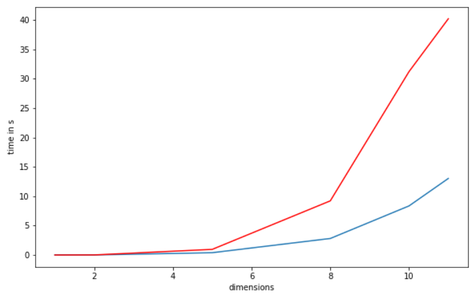
(b) $l = 3$



(c) $l = 4$

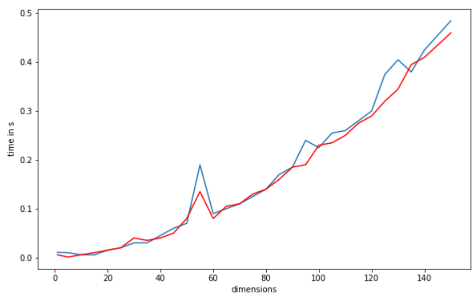


(d) $l = 5$

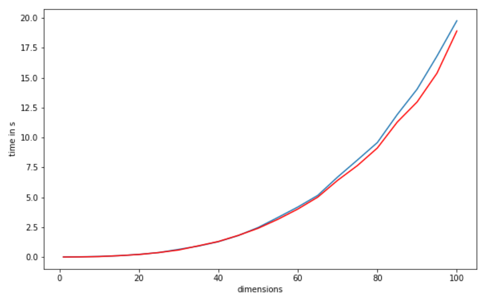


(e) $l = 6$

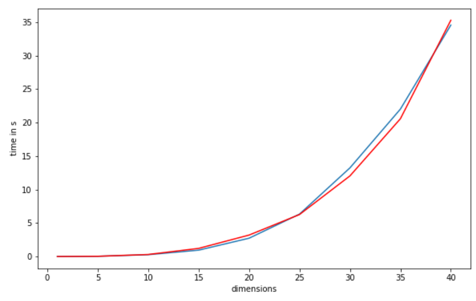
Figure 5.4.: This figure shows plots of the total running time in seconds for different regular sparse grids of different levels l with linear basis functions. The old implementation is represented in red and the new implementation in blue. The running time is shown depending on the dimension of the underlying sparse grid. Note that the dimension ranges differ for each plot.



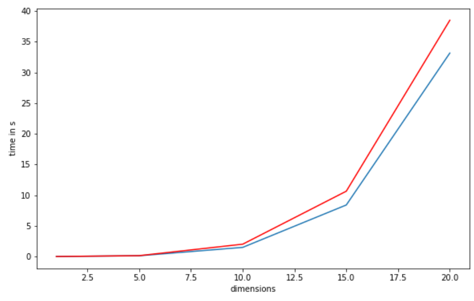
(a) $l = 2$



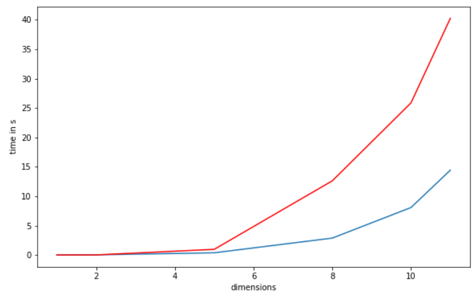
(b) $l = 3$



(c) $l = 4$



(d) $l = 5$



(e) $l = 6$

Figure 5.5.: This figure shows plots of the total running time in seconds for different regular sparse grids of different levels l with modlinear basis functions. The old implementation is represented in red and the new implementation in blue. The running time is shown depending on the dimension of the underlying sparse grid. Note that the dimension ranges differ for each plot.

6. Conclusion

For combination schemes of lower level regular sparse grids, the developed permutation and dimension blow-up method yields no performance improvement. However, for grids with a level larger than 5, the results of the approach are promising. Furthermore, the error compared to building and decomposing each component matrix from scratch is insignificant. So far, the approach has only been implemented for the orthogonal decomposition in SG++, but the implementation aims at the reusability of the main concepts. Therefore, in the future the method may be applied to other matrix decompositions.

The concepts developed in this thesis could possibly be used to massively reduce the memory usage of sparse grid density estimation with the combination technique. To do so, rather than storing the system matrix of each offline object, only one matrix per equivalence class could be stored. The online step could then be adjusted to apply permutation and blow-up before as part of solving the system of equations. This would add additional computational cost to the online step, but may be worth it in memory consuming scenarios.

A. Appendix

The Appendix contains the evaluation data and the hardware specifications of the laptop it was generated. The blow-up dataset is quite large and is therefore omitted.

A.1. Hardware Specification

CPU: Intel(R) Core i7-7500U 4 $\times$ 2.7 GHz
RAM: 16GB DDR4

A.2. Data

Evaluation data for permutation scenario with linear basis functions.

n	mre	mte	tDec	tPerm	tFac
21	0.0	0.0	0	1	0
105	0.0	0.0	0	1	0
135	0.0	0.0	10	1	10
189	0.0	0.0	20	1	20
217	0.0	0.0	40	1	40
217	0.0	0.0	130	1	130
279	0.0	0.0	250	1	250
405	0.0	0.0	420	1	420
441	0.0	0.0	390	1	390
567	0.0	0.0	310	10	31
735	0.0	0.0	1440	20	72
945	0.0	0.0	3330	10	333
945	9.26945e-11	7.88421e-12	3170	20	158
1029	0.0	0.0	4510	30	150
1143	0.0	0.0	7760	30	258
1215	0.0	0.0	10540	50	210
1323	6.00686e-11	7.20842e-10	12650	40	316
1323	0.0	0.0	15340	50	306

n	mre	mte	tDec	tPerm	tFac
1395	0.0	0.0	16620	60	277
1395	0.000256915	2.282859e-10	16010	40	400
1395	0.000219385	1.1808399e-08	16900	40	422
1395	0.000178165	9.83513e-10	18230	50	364
1395	0.0001668359	2.372150e-06	16970	50	339
1519	0.0	0.0	21800	60	363
1701	0.0	0.0	31740	80	396
1953	4.10472e-11	7.45106e-15	51710	100	517
1953	0.0	0.0	54740	100	547
2205	1.04814e-10	1.900659e-12	85380	130	656
2835	0.0	0.0	195070	220	886
4725	4.96075e-11	4.08402e-10	1152010	600	1920
5145	7.06989e-11	1.35761e-13	1570330	740	2122

Evaluation data for permutation scenario with modlinear basis functions.

n	mre	mte	tDec	tPerm	tFac
21	0.0	0.0	0	1	0
105	0.0	0.0	10	1	10
135	0.0	0.0	10	1	10
189	0.0	0.0	10	1	10
217	0.0	0.0	20	1	20
217	0.0	0.0	10	1	10
279	0.0	0.0	30	1	30
405	1.9855900e-10	3.7676e-08	80	10	8
441	1.406170e-10	2.7031599e-10	100	10	10
567	0.0	0.0	250	20	12
735	2.542e-10	9.7313699e-10	670	20	33
945	5.06019e-10	1.45953e-07	2750	30	91
945	1.6024299e-10	6.30737e-11	2730	20	136
1029	2.9366e-09	8.99701e-09	4450	40	111
1143	2.222559e-10	2.34664e-10	7500	40	187
1215	5.594339e-10	5.27464e-07	9410	40	235
1323	2.05176e-10	4.78684e-11	12900	40	322
1323	1.29506e-10	2.45086e-08	12080	40	302
1395	2.40102e-10	2.14388e-10	15680	50	313
1519	3.6682199e-10	1.2125e-08	21470	60	357
1701	1.47407e-10	6.48257e-10	32680	70	466

n	mre	mte	tDec	tPerm	tFac
1953	7.66013e-11	7.68949e-12	53200	100	532
1953	5.31162e-10	2.09486e-07	50920	90	565
2205	2.0804e-09	4.753340e-09	81460	130	626
2835	9.01378e-09	4.93587e-06	187590	210	893
4725	1.0785e-08	1.49787e-07	1131410	590	1917
5145	1.20279e-08	7.35616e-07	1544070	710	2174

Evalutation data for complete combination scheme with linear basis functions.

l	d	mre	mte	tOld	tNew	tFac
2	1	8.19732e-16	1.84365e-16	1	10	0.1
2	5	1.13874e-15	2.02248e-15	1	10	0.1
2	10	5.99102e-16	4.01124e-16	10	1	10
2	15	1.70625e-15	2.59873e-16	10	20	0.5
2	20	1.03834e-16	5.16723e-18	10	20	0.5
2	25	2.74429e-16	2.31886e-18	30	20	1.5
2	30	1.00354e-15	3.57474e-18	110	40	2.75
2	35	2.07421e-15	1.00368e-18	30	40	0.75
2	40	2.31926e-15	1.81218e-19	50	50	1
2	45	1.23769e-15	1.24541e-21	50	60	0.833333
2	50	2.51241e-15	2.07596e-22	70	70	1
2	55	2.51179e-15	3.14835e-23	90	80	1.125
2	60	3.63895e-15	1.48127e-24	80	90	0.888889
2	65	3.92646e-15	4.44052e-26	100	100	1
2	70	4.26825e-15	5.19125e-27	110	110	1
2	75	4.23058e-15	8.67938e-28	120	130	0.923077
2	80	3.72795e-15	1.51195e-29	160	150	1.06667
2	85	4.10133e-15	7.79869e-31	180	170	1.05882
2	90	6.82486e-15	2.10912e-30	180	180	1
2	95	5.01322e-15	2.54915e-32	190	190	1
2	100	7.89596e-15	7.38066e-35	210	220	0.954545
2	105	8.2741e-15	4.45305e-37	260	240	1.08333
2	110	7.61429e-15	2.94148e-37	270	270	1
2	115	8.71797e-15	3.47285e-37	270	360	0.75
2	120	9.39412e-15	6.12837e-38	330	300	1.1
2	125	1.06076e-14	1.26677e-39	350	320	1.09375
2	130	1.21577e-14	3.59716e-40	360	350	1.02857
2	135	1.23338e-14	1.62653e-44	390	380	1.02632

l	d	mre	mte	tOld	tNew	tFac
2	140	1.15356e-14	5.18026e-45	490	410	1.19512
2	150	1.36859e-14	6.79669e-43	470	510	0.921569
3	1	7.28558e-16	6.47381e-17	1	1	1
3	5	1.73319e-15	9.67545e-15	20	10	2
3	10	3.07591e-14	1.34416e-13	40	60	0.666667
3	15	2.94806e-14	9.7006e-14	110	120	0.916667
3	20	1.12952e-15	1.05826e-15	210	230	0.913043
3	25	2.16503e-14	1.33427e-15	380	380	1
3	30	7.38272e-15	1.09012e-16	590	640	0.921875
3	35	9.16336e-15	3.35564e-17	950	950	1
3	40	1.88936e-14	3.04666e-17	1310	1300	1.00769
3	45	9.83712e-15	3.33992e-19	1820	1830	0.994536
3	50	6.38466e-15	4.9561e-21	2510	2430	1.03292
3	55	2.91462e-14	2.67438e-20	3240	3320	0.975904
3	60	2.69996e-14	3.41892e-22	4130	4230	0.976359
3	65	4.25256e-14	9.25126e-23	5070	5290	0.958412
3	70	1.36519e-14	1.67307e-25	6530	6820	0.957478
3	75	1.73033e-14	5.80896e-26	7750	8300	0.933735
3	80	1.0355e-14	3.28235e-26	9260	9800	0.944898
3	85	2.90661e-14	8.15527e-23	11540	12120	0.952145
3	90	2.16579e-14	1.59033e-29	13070	14190	0.921071
3	95	1.9259e-14	9.18428e-31	16030	17080	0.938525
3	100	1.10351e-14	1.67147e-31	26340	20680	1.27369
2	1	0	0	1	10	0.1
2	5	1.198e-15	2.05288e-15	1	10	0.1
2	10	6.30738e-16	3.99431e-16	10	10	1
2	15	1.67536e-15	2.57352e-16	10	10	1
2	20	1.24142e-16	6.19817e-18	20	10	2
2	25	4.12562e-16	3.74675e-18	20	20	1
2	30	7.70068e-16	2.89707e-18	30	30	1
2	35	2.07421e-15	1.00368e-18	30	40	0.75
2	40	2.31948e-15	1.812e-19	40	40	1
2	45	1.07717e-15	1.18555e-21	50	60	0.833333
2	50	2.50884e-15	2.05969e-22	60	60	1
2	55	2.51113e-15	3.14864e-23	80	70	1.14286
2	60	3.48607e-15	1.41475e-24	90	90	1
2	65	3.93234e-15	4.44052e-26	90	100	0.9
2	70	4.26825e-15	5.19125e-27	110	120	0.916667

l	d	mre	mte	tOld	tNew	tFac
2	75	4.12178e-15	8.66991e-28	130	130	1
2	80	3.79437e-15	1.44963e-29	130	150	0.866667
2	85	4.22579e-15	7.89034e-31	160	160	1
2	90	6.82486e-15	2.10912e-30	180	180	1
2	95	5.07459e-15	2.55071e-32	190	200	0.95
2	100	7.89596e-15	7.38066e-35	210	250	0.84
2	105	8.24966e-15	4.45061e-37	230	240	0.958333
2	110	7.61153e-15	2.94047e-37	250	260	0.961538
2	115	8.57878e-15	3.39711e-37	300	280	1.07143
2	120	9.33276e-15	6.09365e-38	310	300	1.03333
2	125	1.06089e-14	1.26677e-39	320	320	1
2	130	1.21577e-14	3.59716e-40	340	360	0.944444
2	135	1.23361e-14	1.61713e-44	380	390	0.974359
2	140	1.15689e-14	5.16115e-45	400	420	0.952381
2	150	1.35933e-14	6.7967e-43	480	490	0.979592
3	1	1.14219e-14	3.00386e-16	1	10	0.1
3	5	1.71873e-15	9.66649e-15	10	10	1
3	10	3.08255e-14	1.34639e-13	50	50	1
3	15	2.94892e-14	9.68265e-14	110	110	1
3	20	1.08698e-15	1.12177e-15	220	230	0.956522
3	25	2.16467e-14	1.3329e-15	370	380	0.973684
3	30	7.45065e-15	1.08267e-16	620	610	1.01639
3	35	9.0647e-15	3.27197e-17	950	950	1
3	40	1.88419e-14	3.05514e-17	1310	1330	0.984962
3	45	9.83583e-15	3.33975e-19	1820	1840	0.98913
3	50	6.58943e-15	4.85145e-21	2620	2460	1.06504
3	55	2.92711e-14	2.67214e-20	3260	3270	0.996942
3	60	2.69991e-14	3.41892e-22	4110	4210	0.976247
3	65	4.25304e-14	9.22472e-23	5070	5500	0.921818
3	70	1.37474e-14	1.72314e-25	6520	6830	0.954612
3	75	1.72999e-14	5.80753e-26	7730	8260	0.935835
3	80	1.04049e-14	3.30326e-26	9320	9800	0.95102
3	85	2.89733e-14	8.13257e-23	11440	12140	0.942339
3	90	2.1591e-14	1.58429e-29	12980	14200	0.914085
3	95	1.94676e-14	9.14824e-31	15400	16830	0.915033
3	100	1.10677e-14	1.73516e-31	18520	19880	0.93159
4	1	2.47251e-14	6.49842e-16	1	20	0.05
4	5	4.94711e-14	8.89611e-13	40	30	1.33333

l	d	mre	mte	tOld	tNew	tFac
4	10	1.34644e-13	3.19066e-12	350	230	1.52174
4	15	4.29087e-14	4.8691e-13	1030	940	1.09574
4	20	7.82605e-14	8.74706e-13	2830	2730	1.03663
4	25	5.88256e-14	1.11904e-13	6600	6690	0.986547
4	30	2.10737e-14	3.93584e-14	11310	11720	0.965017
4	35	3.23277e-14	1.05567e-14	21270	22420	0.948707
4	40	1.65258e-13	5.22416e-15	40770	44980	0.906403
4	1	9.99338e-15	2.4309e-16	1	10	0.1
4	5	4.92243e-14	8.86125e-13	40	40	1
4	10	1.35062e-13	3.20137e-12	260	240	1.08333
4	15	4.28928e-14	4.86686e-13	1020	990	1.0303
4	20	7.83752e-14	8.7524e-13	2810	2810	1
4	25	5.88301e-14	1.11902e-13	6060	6150	0.985366
4	30	2.11132e-14	3.95877e-14	12250	12620	0.970681
4	35	3.23212e-14	1.05309e-14	24500	21530	1.13795
4	40	1.6524e-13	5.22448e-15	32370	33530	0.965404
5	1	4.95884e-14	7.306e-16	1	10	0.1
5	5	2.57431e-13	1.34106e-11	140	110	1.27273
5	10	1.5988e-13	1.72575e-11	1950	1550	1.25806
5	15	1.11956e-12	7.81683e-11	10340	7710	1.34112
5	20	2.84815e-13	2.62528e-11	35470	27370	1.29594
5	1	4.6827e-14	4.60997e-16	1	30	0.0333333
5	5	2.57529e-13	1.34127e-11	140	110	1.27273
5	10	1.59567e-13	1.72216e-11	1980	1360	1.45588
5	15	1.11949e-12	7.81682e-11	10920	7840	1.39286
5	20	2.8517e-13	2.62764e-11	35580	27890	1.27573
6	1	1.4689e-14	5.88773e-16	10	60	0.166667
6	2	4.98589e-14	8.77584e-13	20	20	1
6	5	3.50669e-13	5.27144e-11	1100	390	2.82051
6	8	2.70366e-12	9.01373e-10	10360	2820	3.67376
6	10	7.71113e-14	2.93978e-11	26350	8740	3.01487
6	11	6.46159e-12	3.2969e-09	40520	13260	3.05581
6	1	2.91401e-14	9.06104e-16	1	10	0.1
6	2	5.00176e-14	8.81542e-13	20	20	1
6	5	3.49826e-13	5.26396e-11	920	410	2.2439
6	8	2.70354e-12	9.0138e-10	8630	2820	3.06028
6	10	7.70842e-14	2.94134e-11	25420	8130	3.12669
6	11	6.46155e-12	3.29688e-09	39920	12780	3.12363

l	d	mre	mte	tOld	tNew	tFac
6	1	2.28807e-14	7.32976e-16	10	10	1
6	2	5.02207e-14	8.84609e-13	30	20	1.5
6	5	3.49313e-13	5.25905e-11	890	410	2.17073
6	8	2.70322e-12	9.01326e-10	8670	2820	3.07447
6	10	7.73506e-14	2.94629e-11	41810	8200	5.09878
6	11	6.4617e-12	3.29694e-09	40140	13060	3.07351

Evalutation data for complete combination scheme with modlinear basis functions.

l	d	mre	mte	tOld	tNew	tFac
2	1	8.19732e-16	1.84365e-16	1	10	0.1
2	5	1.13874e-15	2.02248e-15	1	10	0.1
2	10	5.99102e-16	4.01124e-16	10	1	10
2	15	1.70625e-15	2.59873e-16	10	20	0.5
2	20	1.03834e-16	5.16723e-18	10	20	0.5
2	25	2.74429e-16	2.31886e-18	30	20	1.5
2	30	1.00354e-15	3.57474e-18	110	40	2.75
2	35	2.07421e-15	1.00368e-18	30	40	0.75
2	40	2.31926e-15	1.81218e-19	50	50	1
2	45	1.23769e-15	1.24541e-21	50	60	0.833333
2	50	2.51241e-15	2.07596e-22	70	70	1
2	55	2.51179e-15	3.14835e-23	90	80	1.125
2	60	3.63895e-15	1.48127e-24	80	90	0.888889
2	65	3.92646e-15	4.44052e-26	100	100	1
2	70	4.26825e-15	5.19125e-27	110	110	1
2	75	4.23058e-15	8.67938e-28	120	130	0.923077
2	80	3.72795e-15	1.51195e-29	160	150	1.06667
2	85	4.10133e-15	7.79869e-31	180	170	1.05882
2	90	6.82486e-15	2.10912e-30	180	180	1
2	95	5.01322e-15	2.54915e-32	190	190	1
2	100	7.89596e-15	7.38066e-35	210	220	0.954545
2	105	8.2741e-15	4.45305e-37	260	240	1.08333
2	110	7.61429e-15	2.94148e-37	270	270	1
2	115	8.71797e-15	3.47285e-37	270	360	0.75
2	120	9.39412e-15	6.12837e-38	330	300	1.1
2	125	1.06076e-14	1.26677e-39	350	320	1.09375
2	130	1.21577e-14	3.59716e-40	360	350	1.02857
2	135	1.23338e-14	1.62653e-44	390	380	1.02632

l	d	mre	mte	tOld	tNew	tFac
2	140	1.15356e-14	5.18026e-45	490	410	1.19512
2	150	1.36859e-14	6.79669e-43	470	510	0.921569
3	1	7.28558e-16	6.47381e-17	1	1	1
3	5	1.73319e-15	9.67545e-15	20	10	2
3	10	3.07591e-14	1.34416e-13	40	60	0.666667
3	15	2.94806e-14	9.7006e-14	110	120	0.916667
3	20	1.12952e-15	1.05826e-15	210	230	0.913043
3	25	2.16503e-14	1.33427e-15	380	380	1
3	30	7.38272e-15	1.09012e-16	590	640	0.921875
3	35	9.16336e-15	3.35564e-17	950	950	1
3	40	1.88936e-14	3.04666e-17	1310	1300	1.00769
3	45	9.83712e-15	3.33992e-19	1820	1830	0.994536
3	50	6.38466e-15	4.9561e-21	2510	2430	1.03292
3	55	2.91462e-14	2.67438e-20	3240	3320	0.975904
3	60	2.69996e-14	3.41892e-22	4130	4230	0.976359
3	65	4.25256e-14	9.25126e-23	5070	5290	0.958412
3	70	1.36519e-14	1.67307e-25	6530	6820	0.957478
3	75	1.73033e-14	5.80896e-26	7750	8300	0.933735
3	80	1.0355e-14	3.28235e-26	9260	9800	0.944898
3	85	2.90661e-14	8.15527e-23	11540	12120	0.952145
3	90	2.16579e-14	1.59033e-29	13070	14190	0.921071
3	95	1.9259e-14	9.18428e-31	16030	17080	0.938525
3	100	1.10351e-14	1.67147e-31	26340	20680	1.27369
2	1	0	0	1	10	0.1
2	5	1.198e-15	2.05288e-15	1	10	0.1
2	10	6.30738e-16	3.99431e-16	10	10	1
2	15	1.67536e-15	2.57352e-16	10	10	1
2	20	1.24142e-16	6.19817e-18	20	10	2
2	25	4.12562e-16	3.74675e-18	20	20	1
2	30	7.70068e-16	2.89707e-18	30	30	1
2	35	2.07421e-15	1.00368e-18	30	40	0.75
2	40	2.31948e-15	1.812e-19	40	40	1
2	45	1.07717e-15	1.18555e-21	50	60	0.833333
2	50	2.50884e-15	2.05969e-22	60	60	1
2	55	2.51113e-15	3.14864e-23	80	70	1.14286
2	60	3.48607e-15	1.41475e-24	90	90	1
2	65	3.93234e-15	4.44052e-26	90	100	0.9
2	70	4.26825e-15	5.19125e-27	110	120	0.916667

l	d	mre	mte	tOld	tNew	tFac
2	75	4.12178e-15	8.66991e-28	130	130	1
2	80	3.79437e-15	1.44963e-29	130	150	0.866667
2	85	4.22579e-15	7.89034e-31	160	160	1
2	90	6.82486e-15	2.10912e-30	180	180	1
2	95	5.07459e-15	2.55071e-32	190	200	0.95
2	100	7.89596e-15	7.38066e-35	210	250	0.84
2	105	8.24966e-15	4.45061e-37	230	240	0.958333
2	110	7.61153e-15	2.94047e-37	250	260	0.961538
2	115	8.57878e-15	3.39711e-37	300	280	1.07143
2	120	9.33276e-15	6.09365e-38	310	300	1.03333
2	125	1.06089e-14	1.26677e-39	320	320	1
2	130	1.21577e-14	3.59716e-40	340	360	0.944444
2	135	1.23361e-14	1.61713e-44	380	390	0.974359
2	140	1.15689e-14	5.16115e-45	400	420	0.952381
2	150	1.35933e-14	6.7967e-43	480	490	0.979592
3	1	1.14219e-14	3.00386e-16	1	10	0.1
3	5	1.71873e-15	9.66649e-15	10	10	1
3	10	3.08255e-14	1.34639e-13	50	50	1
3	15	2.94892e-14	9.68265e-14	110	110	1
3	20	1.08698e-15	1.12177e-15	220	230	0.956522
3	25	2.16467e-14	1.3329e-15	370	380	0.973684
3	30	7.45065e-15	1.08267e-16	620	610	1.01639
3	35	9.0647e-15	3.27197e-17	950	950	1
3	40	1.88419e-14	3.05514e-17	1310	1330	0.984962
3	45	9.83583e-15	3.33975e-19	1820	1840	0.98913
3	50	6.58943e-15	4.85145e-21	2620	2460	1.06504
3	55	2.92711e-14	2.67214e-20	3260	3270	0.996942
3	60	2.69991e-14	3.41892e-22	4110	4210	0.976247
3	65	4.25304e-14	9.22472e-23	5070	5500	0.921818
3	70	1.37474e-14	1.72314e-25	6520	6830	0.954612
3	75	1.72999e-14	5.80753e-26	7730	8260	0.935835
3	80	1.04049e-14	3.30326e-26	9320	9800	0.95102
3	85	2.89733e-14	8.13257e-23	11440	12140	0.942339
3	90	2.1591e-14	1.58429e-29	12980	14200	0.914085
3	95	1.94676e-14	9.14824e-31	15400	16830	0.915033
3	100	1.10677e-14	1.73516e-31	18520	19880	0.93159
4	1	2.47251e-14	6.49842e-16	1	20	0.05
4	5	4.94711e-14	8.89611e-13	40	30	1.33333

l	d	mre	mte	tOld	tNew	tFac
4	10	1.34644e-13	3.19066e-12	350	230	1.52174
4	15	4.29087e-14	4.8691e-13	1030	940	1.09574
4	20	7.82605e-14	8.74706e-13	2830	2730	1.03663
4	25	5.88256e-14	1.11904e-13	6600	6690	0.986547
4	30	2.10737e-14	3.93584e-14	11310	11720	0.965017
4	35	3.23277e-14	1.05567e-14	21270	22420	0.948707
4	40	1.65258e-13	5.22416e-15	40770	44980	0.906403
4	1	9.99338e-15	2.4309e-16	1	10	0.1
4	5	4.92243e-14	8.86125e-13	40	40	1
4	10	1.35062e-13	3.20137e-12	260	240	1.08333
4	15	4.28928e-14	4.86686e-13	1020	990	1.0303
4	20	7.83752e-14	8.7524e-13	2810	2810	1
4	25	5.88301e-14	1.11902e-13	6060	6150	0.985366
4	30	2.11132e-14	3.95877e-14	12250	12620	0.970681
4	35	3.23212e-14	1.05309e-14	24500	21530	1.13795
4	40	1.6524e-13	5.22448e-15	32370	33530	0.965404
5	1	4.95884e-14	7.306e-16	1	10	0.1
5	5	2.57431e-13	1.34106e-11	140	110	1.27273
5	10	1.5988e-13	1.72575e-11	1950	1550	1.25806
5	15	1.11956e-12	7.81683e-11	10340	7710	1.34112
5	20	2.84815e-13	2.62528e-11	35470	27370	1.29594
5	1	4.6827e-14	4.60997e-16	1	30	0.0333333
5	5	2.57529e-13	1.34127e-11	140	110	1.27273
5	10	1.59567e-13	1.72216e-11	1980	1360	1.45588
5	15	1.11949e-12	7.81682e-11	10920	7840	1.39286
5	20	2.8517e-13	2.62764e-11	35580	27890	1.27573
6	1	1.4689e-14	5.88773e-16	10	60	0.166667
6	2	4.98589e-14	8.77584e-13	20	20	1
6	5	3.50669e-13	5.27144e-11	1100	390	2.82051
6	8	2.70366e-12	9.01373e-10	10360	2820	3.67376
6	10	7.71113e-14	2.93978e-11	26350	8740	3.01487
6	11	6.46159e-12	3.2969e-09	40520	13260	3.05581
6	1	2.91401e-14	9.06104e-16	1	10	0.1
6	2	5.00176e-14	8.81542e-13	20	20	1
6	5	3.49826e-13	5.26396e-11	920	410	2.2439
6	8	2.70354e-12	9.0138e-10	8630	2820	3.06028
6	10	7.70842e-14	2.94134e-11	25420	8130	3.12669
6	11	6.46155e-12	3.29688e-09	39920	12780	3.12363

l	d	mre	mte	tOld	tNew	tFac
6	1	2.28807e-14	7.32976e-16	10	10	1
6	2	5.02207e-14	8.84609e-13	30	20	1.5
6	5	3.49313e-13	5.25905e-11	890	410	2.17073
6	8	2.70322e-12	9.01326e-10	8670	2820	3.07447
6	10	7.73506e-14	2.94629e-11	41810	8200	5.09878
6	11	6.4617e-12	3.29694e-09	40140	13060	3.07351

List of Figures

List of Tables

Bibliography

[BG04] H.-J. Bungartz and M. Griebel. “Sparse Grids.” In: *Acta Numerica* (2004), pp. 1–123.

[Bos17] D. Boschko. “Orthogonal Matrix Decomposition for Adaptive Sparse Grid Density Estimation Methods.” Bachelor’s thesis. Technical University of Munich, Sept. 2017.

[DR08] W. Dahmen and A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. 2nd ed. Springer, 2008.

[Fuc18] D. Fuchsgruber. “Integration of SGDE-based Classification into the SG++ Datamining Pipeline.” Bachelorarbeit. Technical University of Munich, Aug. 2018.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. “A combination technique for the solution of sparse grid problems.” In: *Proceedings of the IMACS International Symposium onIterative Methods in Linear Algebra* (1992), pp. 263, 281.

[HHR00] M. Hegland, G. Hooker, and S. Roberts. *Finite element thin plate splines in density estimation*. ANZIAM Journal, 42:C712–C734, 2000.

[Peh13] B. Peherstorfer. *"Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques"*. Technische Universität München, 2013.

[Pfl10a] D. Pflüger. *"Spatially Adaptive Sparse Grids forHigh-Dimensional Problems"*. Technische Universität München, 2010.

[Pfl10b] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010. ISBN: 9783868535556.

[Rös19] N. Rösel. “Combigrid Based Dimensional Adaptivity for Sparse Grid Density Estimation and Classification.” Bachelor’s Thesis. Technical University of Munich, Apr. 2019.

[Sie16] A. Sieler. “Refinement and Coarsening of Online-Offline Data Mining Methods with Sparse Grids.” Bachelor’s thesis. Technical University of Munich, Mar. 2016.

[SK09] H. R. Schwarz and N. Köckler. *Numerische Mathematik, 7. Auflage*. Springer, 2009.

[Wae17] T. Waegemans. "Image Classification with Geometrically Aware Sparse Grids." Bachelor's thesis. Technical University of Munich, Oct. 2017.