DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Sparse Grid Density Estimation with the Combination Technique

Lukas Schulte

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Sparse Grid Density Estimation with the Combination Technique

Dünngitter Dichteschätzung mit der Kombinationstechnik

Author:	Lukas Schulte
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Michael Obersteiner, M.Sc.
	Paul Sarbu, M.Sc.
Submission Date:	16.03.2020

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

Munich, 16.03.2020 Lukas Schulte

Abstract

In the frame of this bachelor thesis, density estimation with the sparse grid combination technique was implemented and integrated into the sparseSpACE framework. The sparse grid combination technique is used to compute a sparse grid, whereby a specific sequence of small anisotropic full grids is combined linearly, to tackle the curse of dimensionality. The usage of mass lumping in the density estimation process is also explored, which still achieves relatively good results compared to the standard combination method. The density function is estimated for different data sets using the combination technique and compared with the full grid solution in regards to different error norms. The test results show that we achieve a good estimate of the density function while simultaneously reducing the number of grid points used.

Kurzfassung

Im Rahmen dieser Bachelorarbeit wurde die Dichteschätzung mit der Dünngitter Kombinationstechnik implementiert und in das sparseSpACE-Framework integriert. Die Kombinationstechnik wird zur Berechnung eines Dünngitters verwendet, wobei eine bestimmte Folge kleiner anisotroper Vollgitter linear kombiniert werden, um den sogenannten Fluch der Dimensionalität zu umgehen. Auch die Verwendung von einer konzentrierte Massenmatrix bei der Dichteschätzung wird untersucht, die im Vergleich zur Standardkombinationstechnik immer noch relativ gute Ergebnisse erzielt. Die Dichtefunktion wird für verschiedene Datensätze mit Hilfe der Dünngitter Kombinationstechnik geschätzt und mit der Vollgitterlösung im Hinblick auf verschiedene Fehlernormen verglichen. Die Testergebnisse zeigen, dass wir eine gute Schätzung der Dichtefunktion bei gleichzeitiger Reduzierung der Anzahl der verwendeten Gitterpunkte erreichen.

Contents

Abstract iii

Kurzfassung iv

1. Introduction 1

2. Theoretical Background 3
2.1. Grid Based Function Interpolation . . . 3
2.1.1. Interpolation on a Full Grid . . . 3
2.1.2. Nodal Basis . . . 4
2.1.3. Hierarchical Basis . . . 6
2.1.4. Sparse Grids . . . 8
2.1.5. Sparse Grid Combination Technique . . . 12
2.2. Data Mining . . . 14
2.2.1. Classification . . . 14
2.2.2. Clustering . . . 15
2.3. Density Estimation on Sparse Grids . . . 16
2.3.1. Kernel Based Density Estimation . . . 16
2.3.2. Grid Based Density Estimation . . . 16

3. Implementation 18
3.1. The sparseSpACE Framework . . . 18
3.2. Integration into the framework . . . 18
3.2.1. DensityEstimation Grid Operation . . . 19

4. Evaluation 26
4.1. The General Sparse Grid Toolbox SG++ . . . 26
4.1.1. The Data Mining Pipeline . . . 27
4.2. The Circles Data Set . . . 27
4.3. The Old Faithful Data Set . . . 29
4.4. A Multivariate Gaussian Distribution . . . 33

5. Conclusion and Future Work 35

List of Figures 36

List of Tables 38

Bibliography **39**

A. Apendix **40**
A.1. Configurations and Test Results 40
A.1.1. SG++ configuration 40
A.1.2. The Circle Data Set 41
A.1.3. The Old Faithful Data Set 41
A.1.4. The Two Moons Data Set 42
A.1.5. A Multivariate Gaussian Distribution 45

1. Introduction

Nowadays data mining tools become increasingly important due to the steady increase in the amount of data that is generated. This makes methods to extract knowledge from the constantly growing data sets necessary.
In this thesis the sparse grid combination technique is used to analyze the density of data as common full grids suffer from the curse of dimensionality, i.e. that the number of grid points grows exponentially with the number of dimensions. To overcome the curse of dimensionality, sparse grids are used, since full grids with more than three dimensions are not feasible. The basic idea of sparse grids is to omit some points of a full grid while retaining the subspaces that contribute most to the overall solution without compromising the overall accuracy. These sparse grids can be constructed by using the standard combination technique. The combination technique is a simpler method for calculating the surpluses of basis functions centered on grid points, rather than working directly on sparse grids based on the hierarchical basis, by linearly combining a certain sequence of small anisotropic full grids. This also enables us to individually evaluate each full grid in parallel as they are computed independently from each other. While we can achieve more accurate estimations of the density function by using a finer mesh, this also results in sparse grid with more grid points.
In future extensions of the implemented functionality, the number of grid points can be reduced further by using spatially adaptive sparse grids. Here, a coarse sparse grid is refined in several steps by adding points in areas of the domain where many points are located. Furthermore, the dimensional adaptive version of the combination technique, in which additional component grids can be included in the combination scheme, can also increase the accuracy and reduce the amount of grid points.
These estimated density functions could then be used for machine learning tasks, such as classification, i.e. training systems with data samples to classify previously unseen samples into classes, and clustering, i.e. dividing similar data into groups.

The chapter Theoretical Background provides, as the name suggests, an introduction to the theoretical background needed for sparse grid density estimation. It explains the common nodal basis and the hierarchical basis for grid based function interpolation. Then, the concept of sparse grids and the the standard combination technique is explained. This is followed by an introduction to density estimation use cases, using the examples of the machine learning tasks of classification and clustering, followed by an explanation of how a density function can be estimated on a grid.
The chapter Implementation explains the structure and functionalities of the sparseSpACE[1]

[1] `https://github.com/obersteiner/sparseSpACE`

framework and the implementation and integration process of the implemented sparse grid density estimation with the combination technique.
Then follows the chapter Evaluation, which contains comparisons of our implemented method on different data sets and for a variety of grid configurations in respect to three error norms. In addition, the results are compared with the density estimates performed with the SG++[2] framework developed mainly by Dirk Pflüger, which computes the density function directly on a sparse grid in the hierarchical basis.

[2] https://sgpp.sparsegrids.org/

2. Theoretical Background

This chapter describes the theoretical background of the implementation of the sparse grid density estimation using the combination technique. First, it explains how function approximation works on grids and how sparse grids can help to overcome the curse of dimensionality that occurs when using full grids. An introduction to the field of data mining and the machine learning tasks classification and clustering is then given, for which the use of density estimation is examined. On this basis, the standard combination technique and density estimation with the sparse grid combination technique are explained.

2.1. Grid Based Function Interpolation

A function $f \in V$ is usually represented as a linear combination of kernels associated with data points, e.g. in kernel density estimation. In a grid-based approach, on the other hand, the function is constructed as a sum of weighted basis functions placed on a grid. With this approach, unlike with the kernel-based approach, the number of basis functions does not increase with the number of data points. This makes grid based function interpolation a good method to approximate a function such as the density function. This section gives a brief introduction to the approximation of functions on a grid, the curse of dimensionality that arises, and how to use sparse grids and the combination technique to overcome it. For a detailed coverage of these topics please refer to the sources [1] and [2].

2.1.1. Interpolation on a Full Grid

In the following we will restrict our domain to the d-dimensional unit-hypercube $\Omega^d := [0,1]^d$ when interpolating a function $f : \Omega^d \to \mathbb{R}$. In addition, we confine ourselves to functions that are zero on the boundary of Ω^d.
In order to construct an interpolation u of function f, we discretize Ω^d into $(2^l)^d$ grid points $x_{l,i}$, with $l \in \mathbb{N}_0$ being the discretization level. The grid points are equally spaced with a distance (meshsize) of 2^{-l} between two adjacent points in each dimension. On these grid points we place suitable basis function $\varphi_{l,i}(\vec{x})$, which in this thesis is the standard hat function $\varphi : [0,1] \to \mathbb{R}$. However, there are numerous other possible choices for a basis function instead of the selected piecewise d-linear basis functions, such as d-polynomial, Mexican hat or B-spline basis functions. The interpolant u can therefore be constructed as a sum of basis functions $\varphi_{l,i}(\vec{x})$ weighted with the coefficients $\alpha_{l,i}$.

$$f(\vec{x}) \approx u(\vec{x}) := \sum_i \alpha_{l,i} \varphi_{l,i}(\vec{x}) \tag{2.1}$$

2.1.2. Nodal Basis

In this thesis we work primarily with small anisotropic full grids, which are represented in the conventional nodal basis, since they are utilized in the standard combination technique. This section first describes the nodal basis for the one-dimensional case, followed by the extended d-dimensional case.[2]

Nodal Basis in one dimension

In the one-dimensional space we work with functions defined on the unit interval $[0,1]$. This range Ω is divided into 2^l segments of equal size, where $l \in \mathbb{N}_0$ is the level of refinement. We get a total number of grid points $x_{l,i}$ of $2^l + 1$, including points on the border, and $2^l - 1$ grid points without boundary points. In the following we use grids without boundary points. The grid points are defined by

$$x_{l,i} := i \cdot h_l, \qquad i = 1, ..., 2^l - 1\,. \tag{2.2}$$

Here i denotes the index and $h_l := 2^{-l}$ is the distance or mesh size between the individual points. On each of the $2^l + 1$ grid points we place a basis function $\varphi_{l,i} : [0,1] \to \mathbb{R}$. In our case we use the standard hat function

$$\varphi_{l,i}(x) := \max\left(1 - |\frac{x}{h_l} - i|, 0\right)\,. \tag{2.3}$$

Individual one-dimensional hat base functions can be obtained by scaling and translation depending on the level l and the index i. We can see that the basis functions on one level have disjunctive pairwise supports and cover the entire domain. For the level l the node space V_l is defined as the linear span of all basis functions $\varphi_{l,i}$:

$$V_l := \text{span}\left\{\varphi_{l,i}(x) : i = 1, ..., 2^l - 1\right\}. \tag{2.4}$$

We can then formulate the interpolant u of a function $f : [0,1] \to \mathbb{R}$ as sum of basis hat functions $\varphi_{l,i}$ weighted with some $\alpha_{l,i} \in \mathbb{R}$:

$$u(x) = \sum_{i=1}^{2^l} \alpha_{l,i}\varphi_{l,i}(x)\,. \tag{2.5}$$

Figure 2.1 shows basis hat functions $\varphi_{l,i}$ and grid points $x_{l,i}$ and an example function interpolant $u \in V_3$ as a weighted sum of basis functions in the nodal basis with points on the boundary. In the later Section 2.1.5 we only consider anisotropic full grids without points on the boundary for the combination technique.

Nodal Basis in d-dimensions

We extend the nodal basis to the $d \in \mathbb{N}$ dimension by applying the tensor product approach. The domain now becomes $\Omega^d = [0,1]^d$ and the level $l = (l_1, ..., l_d)$ and the index $i =$

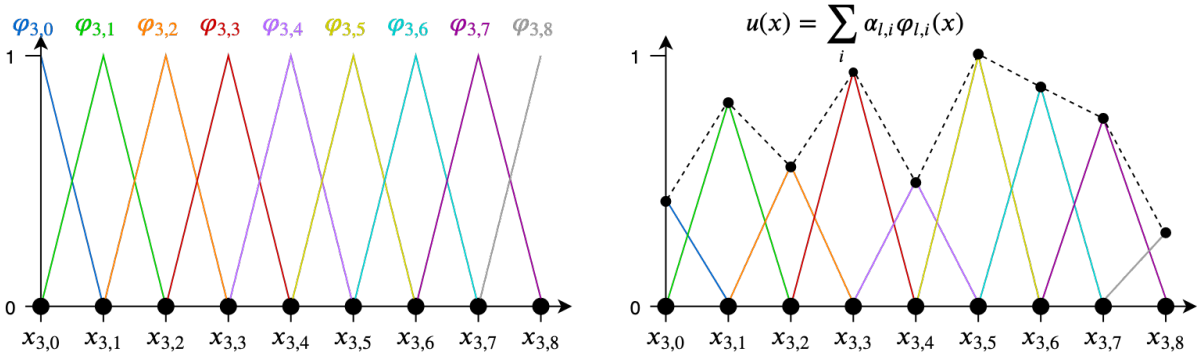


Figure 2.1.: Basis functions $\varphi_{l,i}$ and grid points $x_{l,i}$ of level 3 (left) and a function $u \in V_3$ as a weighted sum of hat functions in the nodal basis.

$(i_1, ..., i_d)$ turn into the vectors $\vec{l}$ and $\vec{i}$ respectively. Ω^d is partitionable into $2^{\|l\|_1}$ equal-sized hyperrectangles, where $\|l\|_1 := \sum_{t=1}^{d} |l_t|$ is the l_1-norm of the level vector. The grid points coordinates are given by

$$x_{\vec{l},\vec{i}} = \vec{i} \cdot h_{\vec{l}} = (i_1 h_{l_1}, ..., i_d h_{l_d}) \tag{2.6}$$

where $h_{\vec{l}} := (h_{l_1}, ..., h_{l_d})$ is the d-dimensional mesh size. In the following, addition, multiplication and exponentiation are performed element-wise when dealing with vectors. We define a nodal index set $I_{\vec{l}}$ to define the indices for each level:

$$I_{\vec{l}} := \left\{ \vec{i} : i_j = 1, ..., 2^{l_j} - 1,\ j = 1, ..., d \right\}. \tag{2.7}$$

It is important to note that this definition of the index set means that we have no points at the boundary of the grid. A basis hat function is again placed on each grid point. The tensor product approach extends the basis function to the d-dimensional case by multiplying the respective one-dimensional basis functions:

$$\varphi_{\vec{l},\vec{i}} : [\vec{0}, \vec{1}] \to \mathbb{R}, \quad \varphi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j). \tag{2.8}$$

Here $\vec{0}$ and $\vec{1}$ are defined as the vectors $(0, ..., 0)$ and $(1, ..., 1)$. Figure 2.2 shows the basis hat function $\varphi_{(2,1),(1,1)}$ on the grid for $\vec{l} = (2, 1)$.
The d-dimensional nodal space $V_{\vec{l}}$ is defined in the same manner as the one dimensional case:

$$V_{\vec{l}} := \operatorname{span}\left\{ \varphi_{\vec{l},\vec{i}}(\vec{x}) : \vec{i} \in I_{\vec{l}} \right\}. \tag{2.9}$$

The interpolant is also defined analogously to the one dimensional case:

$$u(\vec{x}) = \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}(\vec{x}). \tag{2.10}$$

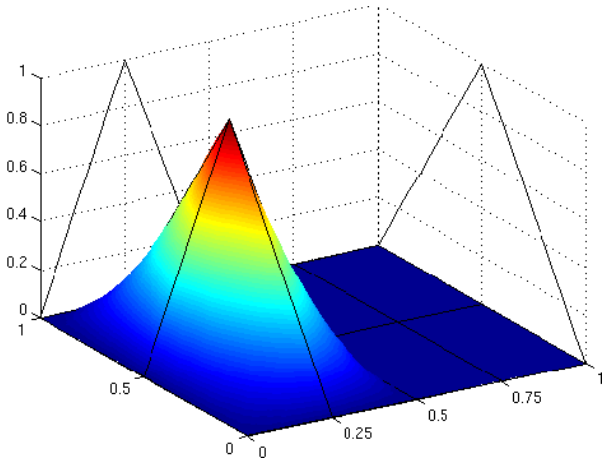


Figure 2.2.: Hat function $\varphi_{(2,1),(1,1)}$.[3]

2.1.3. Hierarchical Basis

Sparse Grids are based on a one-dimensional hierarchical system of basis functions, which can then be extended to the d-dimensional case using a tensor product approach. In this section the one directional case is first explained, followed by the extended d-dimensional case.

Hierarchical decomposition in one dimension

First we define the standard hat function as the basis function that is used.

$$\varphi(x) = \max(1 - |x|, 0) \tag{2.11}$$

Individual one-dimensional hat basis functions, that are centered at grid points $x_{l,i} = i \cdot h_l$, where $h_l = 2^{-l}$ denotes the mesh size of the grid and $i \in \mathbb{N}$ the index, can be obtained by scaling and translation depending on the level l and an index $i = 0, ..., 2^l$:

$$\varphi_{l,i}(x) := \varphi\left(\frac{x}{h_l} - i\right) \tag{2.12}$$

Furthermore we define a hierarchical index set I_l to define which level contains which indices:

$$I_l := \left\{ i = 1, ..., 2^l - 1,\ i \text{ odd} \right\}. \tag{2.13}$$

The set of the resulting hierarchical subspaces $W_l \in \Omega$ is defined by

$$W_l := \text{span}\left\{ \varphi_{l,i}(x) : i \in I_l \right\}. \tag{2.14}$$

The Figure 2.3 shows an illustration of W_l for $1 \leq l \leq 3$.
The space of the functions V_n can then be formulated on a full grid for a given level n as the

direct sum of W_l,

$$V_n = \bigoplus_{l \leq n} W_l \,, \tag{2.15}$$

resulting in the interpolant

$$u(x) = \sum_{l \leq n,\, i \in I_l} \alpha_{l,i} \varphi_{l,i}(x) \,. \tag{2.16}$$

Figure 2.4 shows $u(x) \in V_3$, a weighted sum of the hierarchical basis hat functions, and the corresponding weighted basis functions.

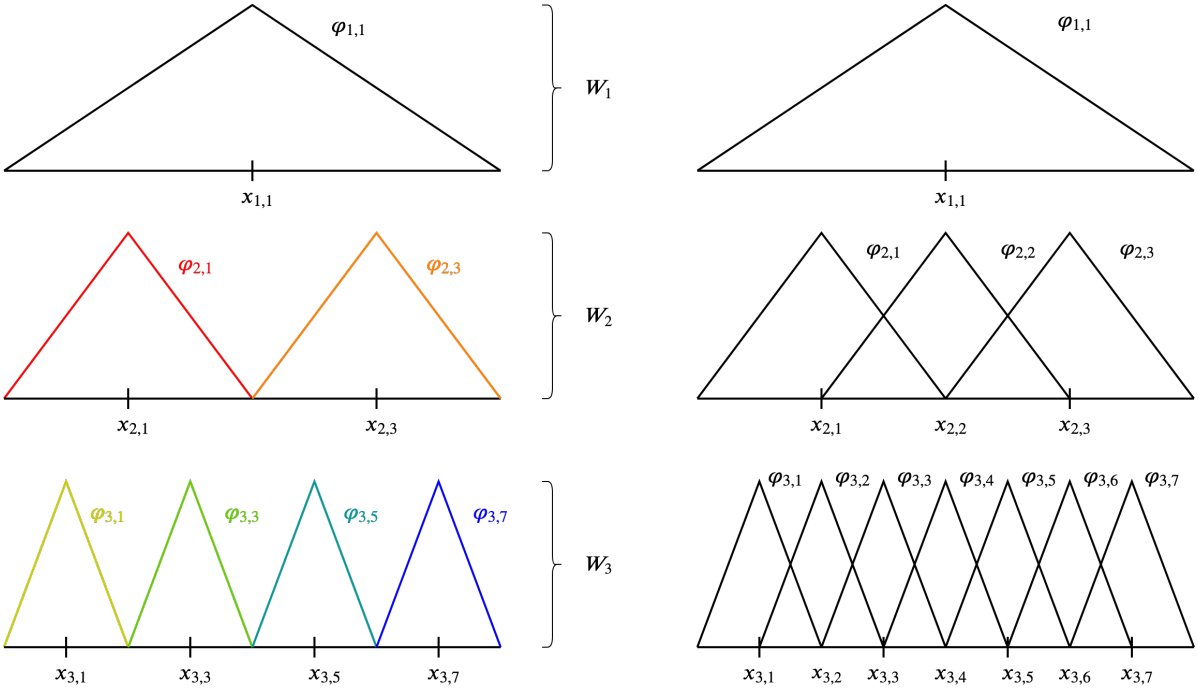


Figure 2.3.: Hat basis functions $\varphi_{l,i}$ centered on the respective grid points $x_{l,i}$ in the hierarchical basis for $l \leq 3$ (left) and in the nodal basis (right).

Hierarchical decomposition in d-dimensions

In the d-dimensional case the level $l = (l_1, ..., l_d)$ and index $i = (i_1, ..., i_d)$ become the vectors $\vec{l}$ and $\vec{i}$ respectively. Furthermore we define the l_∞-norm for this chapter.

$$|\vec{l}|_\infty := \max_{i \leq j \leq d} |l_j| \tag{2.17}$$

A tensor product approach extends the basis functions to the d-dimensional case by multiplying the respective one-dimensional basis functions.

$$\varphi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j) \tag{2.18}$$

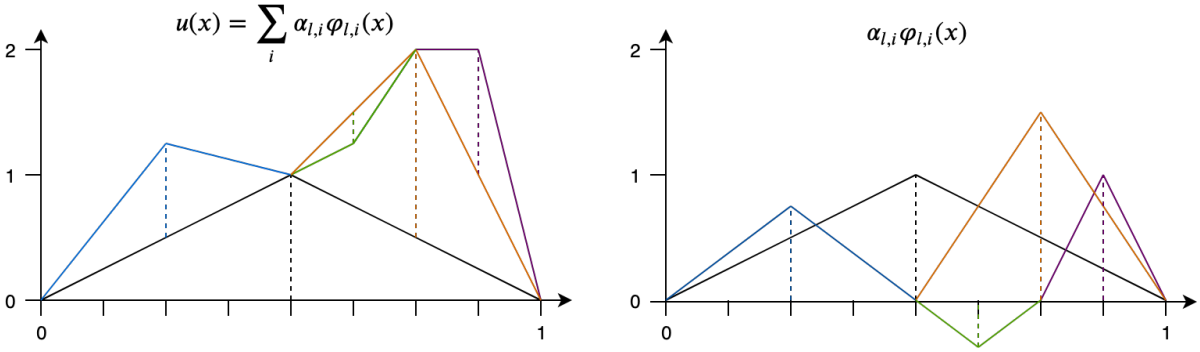


Figure 2.4.: An interpolant $u(x) \in V_3$ (left) and its corresponding weighted basis functions (right) in the hierarchical basis.

Moreover, the index set I_l becomes

$$I_{\vec{l}} := \left\{ \vec{i} : i_j = 1, ..., 2^{l_j} - 1,\ i_j \text{ odd},\ j = 1, ..., d \right\}, \tag{2.19}$$

and the subspace $W_{\vec{l}}$ follows accordingly

$$W_{\vec{l}} := \text{span}\left\{ \varphi_{\vec{l},\vec{i}}(\vec{x}) : \vec{i} \in I_{\vec{l}} \right\}. \tag{2.20}$$

Figure 2.5 shows two dimensional examples of $W_{\vec{l}}$ for $l_i \leq 3$. The space of the functions V_n with refinement level n is again accordingly the direct sum of $W_{\vec{l}}$.

$$V_n = \bigoplus_{|\vec{l}|_\infty \leq n} W_{\vec{l}} \tag{2.21}$$

This leads to the interpolant $u(\vec{x}) \in V_n$,

$$u(\vec{x}) = \sum_{|\vec{l}_\infty| \leq n, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}(\vec{x}). \tag{2.22}$$

2.1.4. Sparse Grids

Sparse grids are a numerical discretization technique used to accelerate the solution of a variety of computational problems, such as interpolation, classification, clustering or density estimation. Nowadays we often have to deal with an increased dimensionality and overall complexity of the data. This leads to the fact that we encounter the curse of dimensionality, i.e. the exponentially increased computing effort at higher dimensions. For instance, this is the case with conventional full grid interpolation schemes. We use sparse grids to overcome the curse of dimensionality to some extent, as they require a much smaller number of grid points than a full grid.

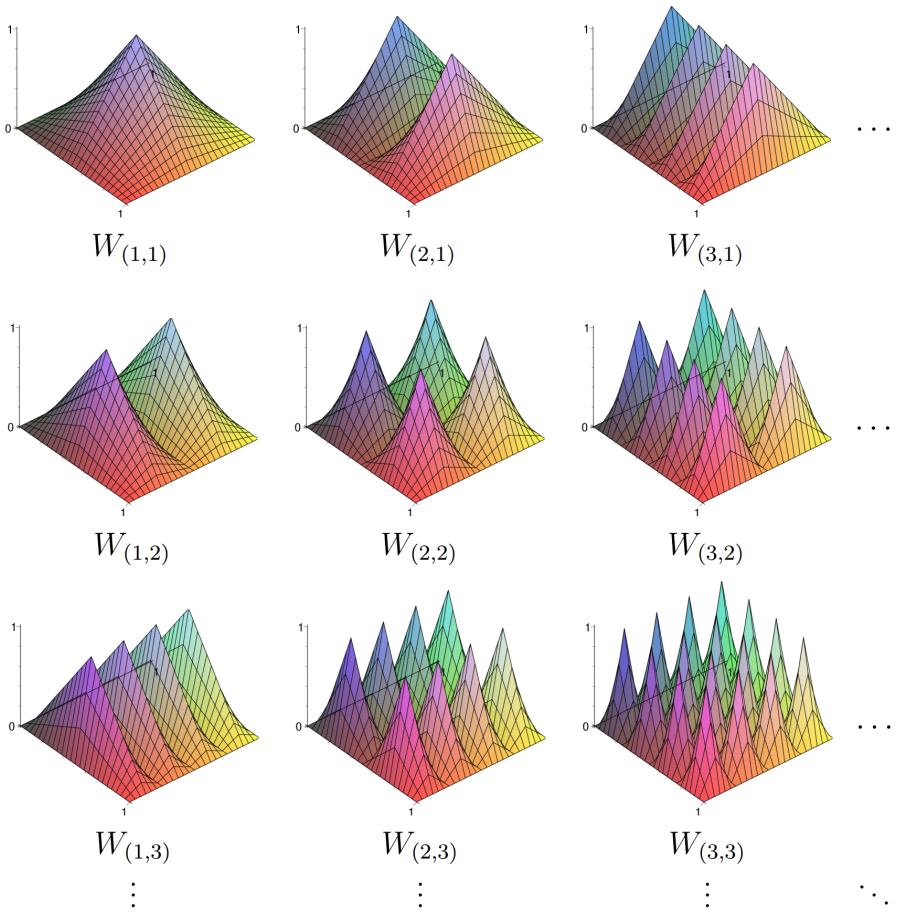


Figure 2.5.: Basis hat functions in two dimensions of the subspaces $W_{\vec{l}}$ for $l_j \leq 3$. [1]

The basic idea of sparse grids is to omit some points of a full grid and to keep the subspaces that contribute most to the overall solution without compromising the accuracy of the solved problem based on the information of the grid points. For a standard full grid, also called regular grid, the number of grid points is in the order of $\mathcal{O}(n^d)$, while for a sparse grid it is in $\mathcal{O}(n \cdot \log_2(n)^{d-1})$, where n specifies the number of grid points per dimension in the full grid. We see that the number of points is greatly reduced while at the same time the L^2-error $\|u - f\|_{L^2}$ increases only slightly from $\mathcal{O}(n^{-2})$ to $\mathcal{O}(n^{-2} \cdot \log_2(n)^{d-1})$.[3]
We start by defining the $|\vec{l}|_1$-norm:

$$|\vec{l}|_1 := \sum_{j=1}^{d} |l_j| \,. \tag{2.23}$$

When constructing a sparse grid, we omit certain subspaces $W_{\vec{l}}$ whose level sum $\|\vec{l}\|_1$ exceeds a certain value, resulting in a reduction of points. A d-dimensional sparse grid of the plane n is thus defined as

$$V_n^{(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}} \,. \tag{2.24}$$

The sparse grid interpolant $u(\vec{x}) \in V_n^{(1)}$ of level n is defined as

$$u(\vec{x}) = \sum_{|\vec{l}|_1 \leq n+d-1,\, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}(\vec{x}) \,. \tag{2.25}$$

Figure 2.6 shows the two-dimensional construction of $V_n^{(1)}$. There is no difference between the full grid and the sparse grid for the one-dimensional case. The selection of subspaces that make up the sparse grid of level $n = 3$ on the right are shown in black while the omitted subspaces are gray. Adding the gray subspaces to the selection would result in the full grid again.

With the help of spatial adaptivity the number of grid points can be further reduced. Since in this thesis, spatial adaptive sparse grids are not used in the implementation part, the concept of spatial adaptivity is only briefly introduced. By starting with a rather coarse sparse grid and adding points in those areas of the domain that are most important. This allows us to improve the accuracy in the regions of interest while keeping the number of grid points at an acceptable level. To decide where the most important area lays in the domain and where a new point should be added we need a certain criteria for adaptive refinement. One such criterion is the selection of refinement candidates on the basis of the highest absolute values of their coefficient $\alpha_{l,i}$ weighted with the functional value at the corresponding grid point.[4] Figure 2.7 shows an example of the spatial adaptive refinement process, in which additional grid points are added to high density areas in each step. Figure 2.8 shows examples of all the level 3 grid types mentioned in this section.

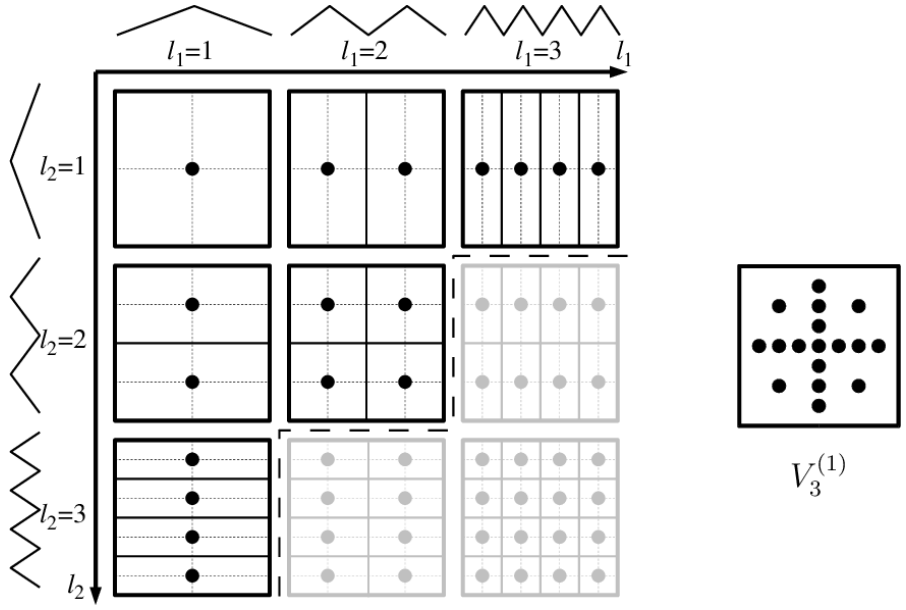


Figure 2.6.: The two-dimensional sparse grid space $V_n^{(1)}$ and its subspaces $W_{\vec{l}}$ up to level $n = 3$ (left) and the resulting sparse grid space (right).[1]

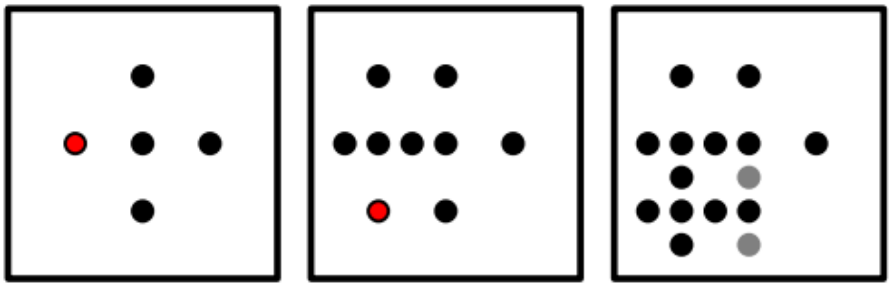


Figure 2.7.: Example of spatial adaptive refinement process.[4]

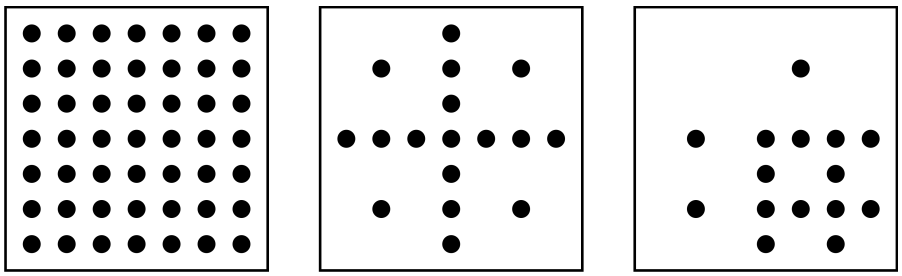


Figure 2.8.: An example full grid of level 3 without points on the boundary on the left, a regular sparse grid of leven 3 in the middle and a spatial adaptive sparse grid on the right.

2.1.5. Sparse Grid Combination Technique

When dealing with sparse grids, the most popular approach to calculate the surpluses of the basis functions centered on the grid points is the so called combination technique. In addition to working directly in the hierarchical base, the combination technique can be used to compute a sparse grid representation of a function, whereby a specific sequence of small anisotropic full grids represented in the conventional nodal basis, which are also called component grids, is combined linearly. The combination technique is widely used because it is much easier to implement compared to working directly with the hierarchical sparse grid basis, due to the fact that full grids are widely used in many applications. There is also a dimensionally adaptive variation of the standard sparse grid combination technique, in which arbitrary component grids are added gradually while an admissibility condition is satisfied until a global criterion is met. This approach can result in sparse grids with different degrees of discretization in each dimension, which can increase accuracy in the dimensions with a higher degree of discretization. [1]
Figure 2.9 shows the combination technique in two dimensions for a level 4 grid on the left and for a dimensionally adaptive sparse grid on the right. The component grids with $|\vec{l}|_1 = 5$ (blue) are added, while the component grids with $|\vec{l}|_1 = 4$ (red) are subtracted. The Figure 2.10 shows the output of the implementation for the standard combination technique of level 4 in two dimensions for the funnychess data set.

An anisotropic grid is a grid with uniform mesh sizes $h_t = 2^{(-l_t)}$ in each of the t-dimensions, so each grid usually has a different mesh size for the respective coordinate direction. To evaluate the sparse grid function, the component grid functions are calculated individually and independently of each other and then added together. This enables the calculation to be performed in parallel. The combination technique exploits that a sparse grid interpolant $u(\vec{x})$ of a d-dimensional function f can be represented as the sum

$$u(\vec{x}) := \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{k} \sum_{|\vec{l}|_1 = n+(d-1)-q} u_{\vec{l}}(\vec{x}) \,, \tag{2.26}$$

where $u_{\vec{l}}$ are the different partial functions from the different grids. This combined representation of the sparse grid interpolant is identical with the hierarchical sparse grid interpolant. [3]

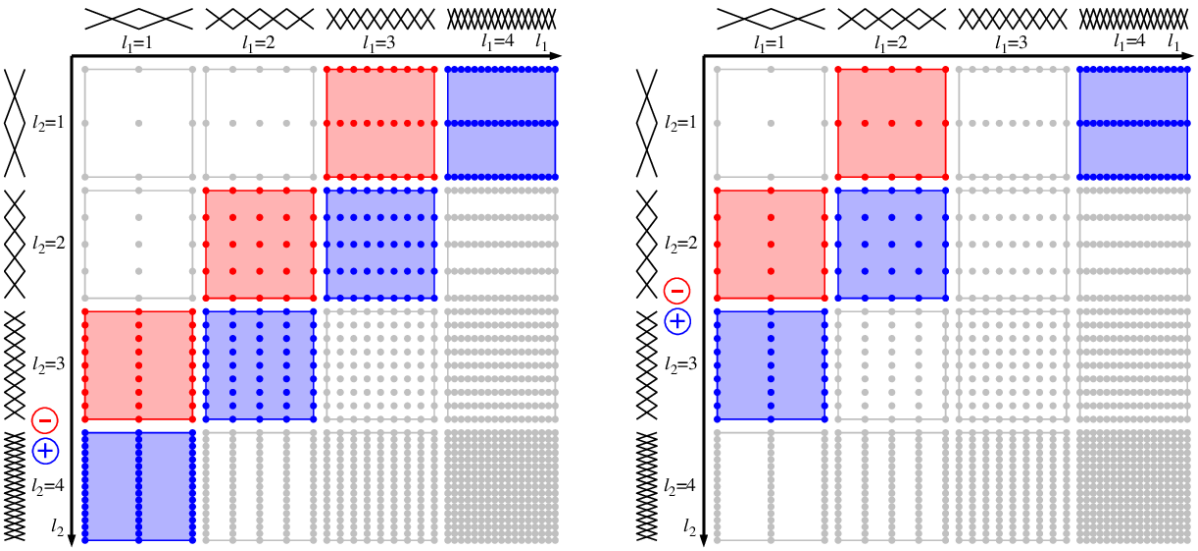
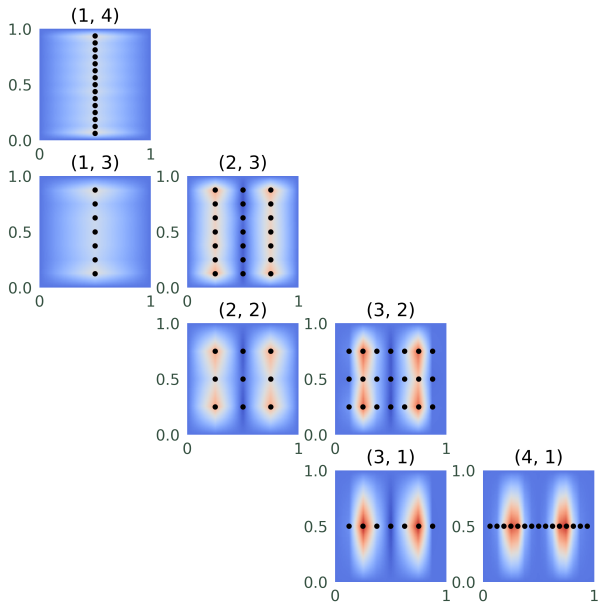
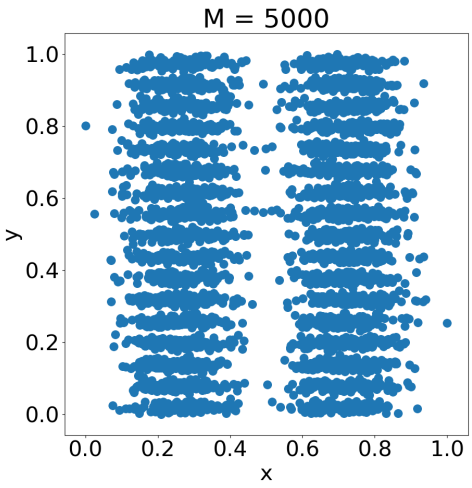


Figure 2.9.: Combination technique in two dimensions with points on the boundary. By adding the blue subspaces and subtracting the red ones, we obtain a level 4 sparse grid on the left and dimensionally adaptive sparse grid on the right. [1]



(a) Combination technique of level 4 in two dimensions with densities of each component grid for the funnychess data set.



(b) The funnychess data set from the SG++ library.

Figure 2.10.: Example of the standard combination technique for the funnychess data set.

2.2. Data Mining

Nowadays Data Mining plays an important role in many sciences because the amount of data is increasing rapidly and it is no longer possible to analyse this huge amount of data with traditional methods. You are confronted with huge amounts of data in almost all areas today, but we are just now actively trying to convert them into valuable information and knowledge with the help of data mining.
This section gives an introduction to the data mining tasks classification and clustering and then illustrates the use of density estimation for these tasks. There are three phases to the data mining process: Data pre-processing, data modeling, and data post-processing. In the context of this work, we are only interested in the data modeling part of the data mining process that for example deals with training systems with data samples in order to generalize unseen samples, i.e. classification, and dividing similar data into groups, i.e. clustering.
In this thesis we use density estimation to provide visualization and retrieval of information from data, but it can also be a tool that can be used for other commonly used data mining tasks such as clustering and classification.

2.2.1. Classification

Classification is used to predict the labels of objects so that you can distinguish between them. Prediction methods are used to estimate the future value of an object in terms of patterns in the data. Classification is a case of supervised learning. Supervised algorithms take already labeled data as input and then try to predict the output.
Suppose we have a function $p : \mathbb{R}^d \to K$ that assigns class labels from the the set of classes $K = \{1, ..., k\} \subset \mathbb{N}$ to data points in the d-dimensional space $\mathbb{R}^d$. Furthermore, we have a data sample $\{x_1, ..., x_M\} \subset \mathbb{R}^d$, which can be expressed as a vector $\vec{x}$. The goal would then be to find a function $\hat{p} : \mathbb{R}^d \to K$ which approximates the function p so that it correctly assigns the correct class $y \in K$ to every sample $x \in \mathbb{R}^d$. Since supervised algorithms take already labelled data as input, we get the training data set S using the data samples and the corresponding classes: [5]

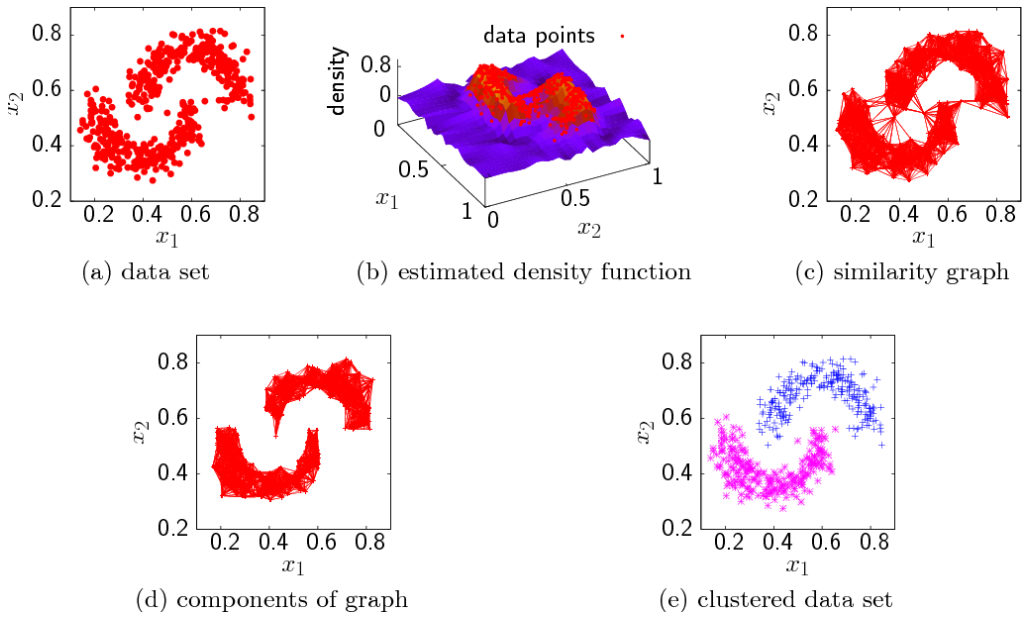
$$S = \{(\vec{x}, y) | \vec{x} \in \mathbb{R}^d, y \in K, f(\vec{x}) = y\}. \tag{2.27}$$

For classification based on the sparse grid density estimation, the density function must first be obtained separately for each class of the data set by dividing the data set into classes and then estimating the respective density functions on the sparse grids. A new data point's class is identified by the value of the density functions at the given point. The class label associated with the density function that returns the highest value for the given data point is also the class label for that point. The confidence in this class assignment is high if the selected density function has a significantly higher value than the other density functions, and low if several density functions yield approximately the same value, because this means that the classes overlap at this point. [6]
Figure 2.11 shows an illustration of the classification process based on density estimation for the two moon data set.

2.2.2. Clustering

Broadly speaking, clustering is used to divide similar objects into groups. Clustering is the corresponding unsupervised process to classification, meaning that it works only with unlabeled input data and then tries to find similarities in the data.
Suppose we have a specific training data set $S = \{x_1, ..., x_M\} \subset \mathbb{R}^d$, then clustering divides S into groups of similar data points with respect to a defined similarity measure. Note that the training data set here does not include any target values like in Section 2.2.1, as it is a unsupervised method. [5]

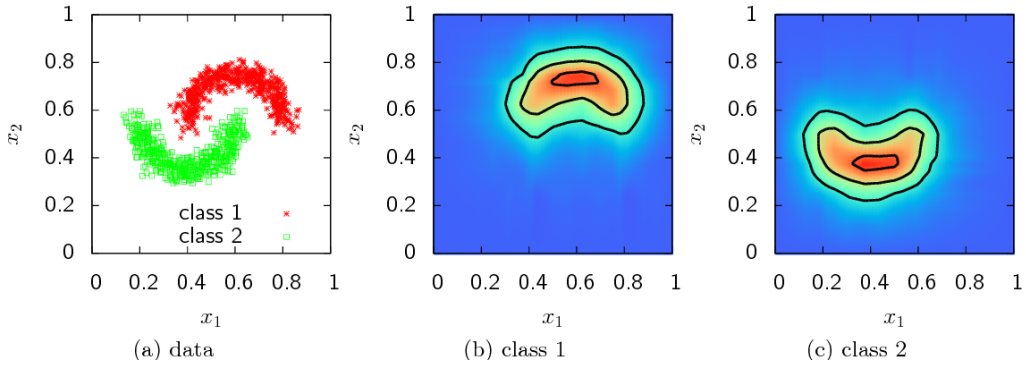
In clustering based on the sparse grid density estimation, a cluster is defined as a region with a high number of data points (high density) surrounded by a region with relatively few data points (low density). A similarity graph is first constructed from the data and then the density estimation method described in Section 2.3 is used to approximate the density function. Then all vertices of the similarity graph that have a density below a certain threshold are deleted, resulting in the similarity graph being split into several components representing the cluster centers. [7]
Figure 2.12 shows an illustration of the clustering process based on density estimation for the two moon data set.



(a) data set (b) estimated density function (c) similarity graph

(d) components of graph (e) clustered data set

Figure 2.11.: Example of the classification process based on density estimation for the two moon data set. [5]



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

Figure 2.12.: Example of the clustering process based on density estimation for the two moon data set. [5]

2.3. Density Estimation on Sparse Grids

The task of density estimation is to construct an estimated density function $\hat{f}$ of f based on a data set $S = \{x_1, ..., x_M\} \subset \mathbb{R}^d$. Density estimation methods are split into parametric and nonparametric methods. A method of estimating parametric density assumes that the underlying distribution form is known and only a very limited number of parameters need to be estimated. In contrast, nonparametric density estimation uses only the available data samples for density estimation and does not rely on additional information about the data. [1]

2.3.1. Kernel Based Density Estimation

The estimated density function $\hat{f}$ can for example be a kernel density function. We construct it by placing the non-negative kernel function K onto the data points x_i:

$$\hat{f}(x) = \frac{1}{M} \sum_{i=1}^{M} K(\frac{x - x_i}{h}) . \tag{2.28}$$

The parameter h is the so called smoothing coefficient, also called the bandwidth. The performance of the kernel density estimator is influenced by the choice of the kernel function K and the bandwidth h. Furthermore, the evaluation of $\hat{f}$ depends on the number of data points M in the data set S, so all M kernel functions on the data points x_i have to be evaluated in order to evaluate the function $\hat{f}$ in contrast to sparse grid based density estimation where the estimated density function is only evaluated at the basis functions centered at the grid points. A commonly chosen kernel function is the Gaussian kernel [4]

$$K(x) = (2\pi)^{-1/2} e^{-x^2/2} . \tag{2.29}$$

2.3.2. Grid Based Density Estimation

In grid based density estimation, we begin with a initial estimate f_ϵ of the density function of the data set S, which is then gradually smoothed with a spline smoothing approach until we

obtain an approximated density function $\hat{f}$ such that

$$\hat{f} = \arg\min_{f \in V} \int_{\Omega} (f(x) - f_{\epsilon}(x))^2 dx + \lambda \left\| \Lambda f \right\|_{L^2}^2 . \tag{2.30}$$

Here the regularization term $\|\Lambda f\|_{L^2}^2$ is used to control the smoothness of the resulting density function, while the coefficient $\lambda > 0$ controls the balance between accuracy and smoothness. Λ represents a differential operator. For the initial guess of the density function f_{ϵ} the approach that places a Dirac delta function δ_{x_i} at at every data point $x_i \in S$ is used:

$$f_{\epsilon} := \frac{1}{M} \sum_{i=1}^{M} \delta_{x_i} . \tag{2.31}$$

Furthermore, let $W^{(1)}$ be the set of basis functions of the sparse grid space $V^{(1)}$. We are then looking for $f(x) \in V^{(1)}$ such that

$$\int_{\Omega} f(x)\varphi(x)dx + \lambda \int_{\Omega} \Lambda f(x) \cdot \Lambda \varphi(x) dx = \frac{1}{M} \sum_{i=1}^{M} \varphi(x_i) \tag{2.32}$$

holds for all $\varphi \in W^{(1)}$. This equation can be expressed as the system of linear equations

$$(R + \lambda C)\vec{\alpha} = \vec{b} , \tag{2.33}$$

with $R_{ij} = (\varphi_i, \varphi_j)_{L_2}$, $C_{ij} = (\Lambda\varphi_i, \Lambda\varphi_j)_{L_2}$ and $b_i = \frac{1}{M} \sum_{j=1}^{M} \varphi_i(x_j)$. With $(\cdot, \cdot)_{L_2}$ we denote the standard L_2-inner product of two basis function on the domain Ω^d:

$$(\varphi_{\vec{l},\vec{i}}, \varphi_{\vec{l}',\vec{i}'})_{L_2} = \int_{\Omega^d} \varphi_{\vec{l},\vec{i}}(\vec{x}) \varphi_{\vec{l}',\vec{i}'}(\vec{x}) d\vec{x} . \tag{2.34}$$

The system of linear equations can further be simplified by replacing the matrix C with the identity matrix I:

$$(R + \lambda I)\vec{\alpha} = \vec{b} . \tag{2.35}$$

This is done by choosing a simpler and computationally more efficient regularization operator Λ, which greatly simplifies the optimization problem in Equation 2.3.2. This regularization operator no longer preserves moments, but these properties are not needed for estimating density functions. [4]
By solving this system we obtain the vector $\vec{\alpha}$ which contains the coefficients of the corresponding basis functions which together form the sparse grid interpolant, seen in Equation 2.1.4. Since the calculation of the R-matrix depends solely on the sparse grid and is independent of the data set, Offline/Online splitting was proposed as a method to considerably accelerate the calculation of the system in Equation 2.35 by splitting it into an Offline and an Online phase. In the offline phase, the matrix $R + \lambda I$ is precalculated and stored for a variety of grids. It can then be loaded in the Online phase to solve the linear system of equations in Equation 2.35 in $\mathcal{O}(N^2)$ instead of $\mathcal{O}(N^3)$. [5]

3. Implementation

This chapter first explains the structure of the sparseSpACE framework, whose offered functionalities are extensively used in our implementation. Subsequently, the second section deals with the implementation of the sparse grid density estimation with the combination technique and the integration into the framework.

3.1. The sparseSpACE Framework

The sparse grid density estimation is implemented in an existing sparse grid framework called sparseSpACE. The Sparse Grid Spatially Adaptive Combination Environment (sparseSpACE), created by Michael Obersteiner, offers a variety of spatially adaptive combination methods and supports the implementation of arbitrary grid operations. For instance, the Extend-Split strategy is implemented, which is a variant of the Sparse Grid Combination Technique that provides a new method for spatially fitting a full grid to a function with very local behavior [8]. In this thesis only the standard combination technique is used, which is employed for the calculation of the set of level vectors and their respective coefficients, as described in Section 2.1.5. The framework supports several different grid types like `TrapezoidalGrids`, `ClenshawCurtisGrids` or `GaussLegendreGrids`. These grid implementations provide useful methods that we can use when implementing density estimation, such as checking whether a point lies on the boundary of the grid or obtaining the coordinates of a point. In the implementation, a trapezoidal grid without grid points on the boundary is utilised to make use of these methods.

The classes in `GridOperation` define possible operations on sparse grids. For instance, `UncertaintyQuantification` implements a single-dimension spatially adaptive sparse grid refinement strategy for uncertainty quantification and `Interpolation` provides functionality to interpolate a grid at a given list of evaluation points using bilinear interpolation.
The Standard Combination Technique is implemented in `StandardCombi` and provides functionalities for interpolating points using the Combination Technique and to display the resulting combination scheme and sparse grid. The method `perform_operation` performs the standard combination technique for the specified `GridOperation`.

3.2. Integration into the framework

To support density estimation in sparseSpACE, the new `GridOperation` class `DensityEstimation` was added.

3.2.1. DensityEstimation Grid Operation

In order to use the new grid operation, the user must pass a `DensityEstimation` object to the `StandardCombi` constructor. It is possible to pass either a path to a `.csv` file to specify the data set used for density estimation, or to pass a `NumPy` array directly when the grid operation is created. Different data sets can be created with the `scikit-learn` package `sklearn.datasets` or with the `NumPy` package `random`, which provides random sampling of various different distributions. The data is scaled to the range $(0, 1)$ in the `initialize()` function that is called when the grid operation is performed. This has to be done because the implementation can only handle values between zero and one.

The user can also specify a λ value that controls the smoothness of the density estimation and helps prevent overfitting when an appropriate value is chosen for the specific data set. In addition, the user can specify whether mass lumping should be used in the R-matrix calculation in the Equation 2.35. Here we omit all cases where the basis functions only partially overlap, resulting in R being a diagonal matrix. This is feasible because we use the nodal basis for the component grids of the combination scheme, for which the linear system seen in Equation 2.35 is solved, where the R matrix is already sparse because we have less overlapping basis functions compared to the hierarchical basis. This speeds up the calculation because only the diagonal values of the matrix are calculated and since the values on the diagonal are all the same, only one value on the diagonal needs to be evaluated. This accelerates the solution of the linear system $(R + \lambda I)\vec{\alpha} = \vec{b}$, since R becomes a scalar matrix but also decreases the accuracy as seen in chapter 4.

After performing the density estimation operation by calling the `perform_operation` method of the `StandardCombi` object, the surpluses of each component grid are calculated and stored in a dictionary. These calculated surpluses can then be used to interpolate the density function or when plotting the resulting density estimation in 3D or as a contour plot. In Figure 3.1 you can see the resulting density estimation and its contour plot for the data set for a regular sparse grid of level 5 and $\lambda = 0.001$.

In Figure 3.2 you can see an example of the output of the implementation for the the combination scheme with the contour plot of each component grid.

The Figure 3.3 shows the basic use case of the implemented density estimation method, where the class `StandardCombi` uses several functions of the new `GridOperation` class `DensityEstimation`. The function `plot_dataset()` of the `GridOperation` can be used directly by the user. There is also a Python script for comparing the result with the SG++ toolbox and plotting the differences between the two if one has a `.csv` file containing the SG++ evaluations for the data set.

For estimating the density function, the main effort is the solution of the linear system $(R + \lambda I)\vec{\alpha} = \vec{b}$ from the Equation 2.35. The R matrix is obtained by calculating the L_2-inner product of pairs of hat basis function φ_i and φ_j:

$$R_{ij} = (\varphi_i, \varphi_j)_{L_2} . \tag{3.1}$$

For the nodal basis, which is used in the standard combination technique for each full grid, there are three different cases when the L_2-scalar product is calculated of two hat basis

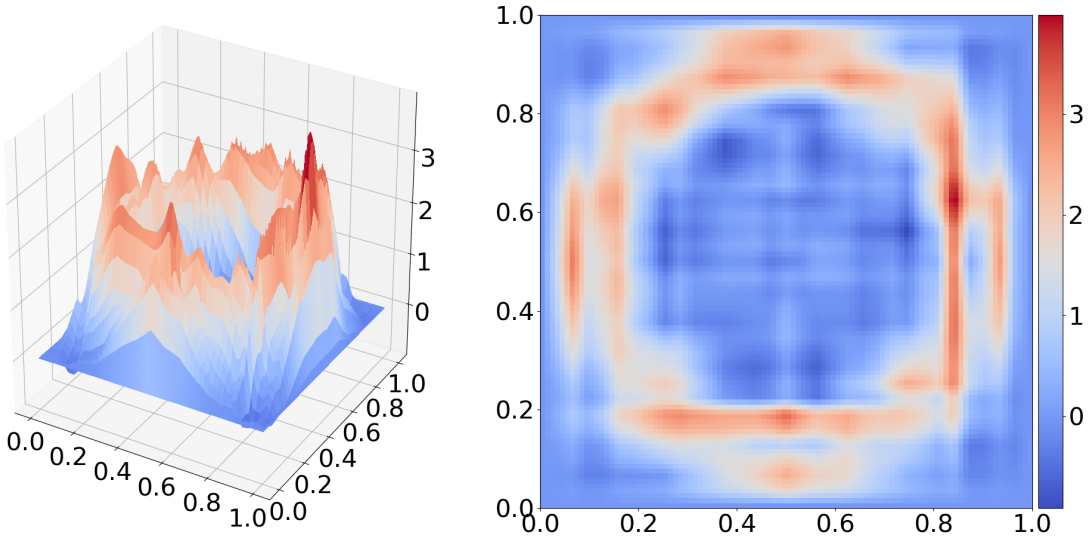


Figure 3.1.: The density estimation (left) and corresponding contour representation (right) for the circle data set for a regular sparse grid of level 5 and coefficient $\lambda = 0.001$.

functions. The support of these two hats can either completely or partially overlap or they do not overlap at all. In fact, many entries of R will be zero due to the non-overlapping support of the hat functions. We would have many more non-zero entries in the R matrix if we had used the hierarchical base, since there are many more overlapping functions. For the 1-dimensional case the L_2-inner product can be calculated directly:

$$\int_0^1 \varphi_{l,i}(x)\varphi_{l',i'}(x)dx = \begin{cases} \frac{2}{3}h_l & \text{full overlap,} \\ \frac{2}{12}h_l & \text{partial overlap,} \\ 0 & \text{no overlap.} \end{cases} \tag{3.2}$$

The explicit calculation of each of these three cases and the construction of the R-matrix is also shown in algorithm 1. In the implementation, only the upper right part of the matrix and a value on the diagonal must be calculated. This is possible because the matrix of size $n \times n$, where n is the number of grid points of the grid, is symmetrical and positive definite. We only need to calculate one of the values on the diagonal, because these values are all the same, because in these cases two identical hats overlap completely. When using mass lumping, only one diagonal value needs to be calculated. This speeds up the construction of the matrix considerably, but the accuracy decreases, as can be seen in chapter 4. The construction of the vector $\vec{b}$ of the linear system from Equation 2.35 is shown in algorithm 2.

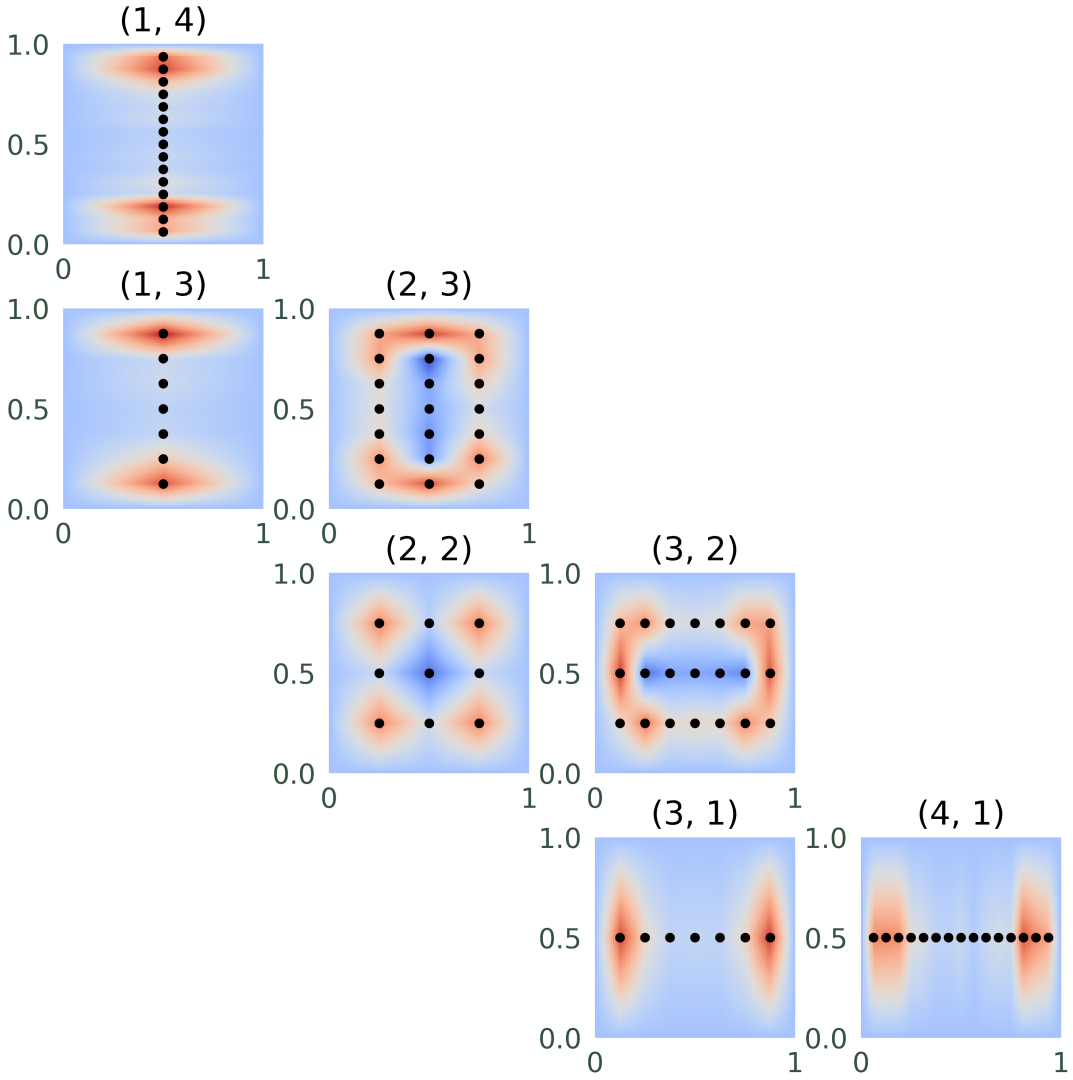


Figure 3.2.: The combination scheme of level 4 in two dimensions and $\lambda = 0.001$ with contour plot of the density of each component grid for the circle data set.

The $\vec{b}$-vector is constructed by summing all evaluations of the standard hat basis function at every data point x_j of the data set of size M and scaling it by $\frac{1}{M}$:

$$b_i = \frac{1}{M} \sum_{j=1}^{M} \varphi_i(x_j) \,. \tag{3.3}$$

This process is accelerated in the implementation seen in algorithm 2 by evaluating only those hat functions in whose support the data point x_j lies. To obtain these specific hat functions, another function seen in algorithm 3 is used.
The idea is that given a data point $\vec{x}$ and the d-dimensional mesh size $h_{\vec{l}} := (h_{l_1}, ..., h_{l_d})$ we can calculate all hat functions in whose support the data point lies. To do this, we iterate through each dimension and create a list of indices of the lower and upper index of the hat functions for this data point. We obtain the lower and upper indexes by dividing the data point coordinate in the specific dimension by the mesh size in that dimension and using the floor function for the lower index and the ceiling function for the upper index. We then need to filter out the indices that are on the boundary of the grid. Then we build the cartesian product with itself of the constructed list of indices, to get all indices of the hat functions in whose support the data point $\vec{x}$ lies. In the end we only have to filter out the duplicates, which can be done efficiently, for example, by using a set when calculating the cartesian product.

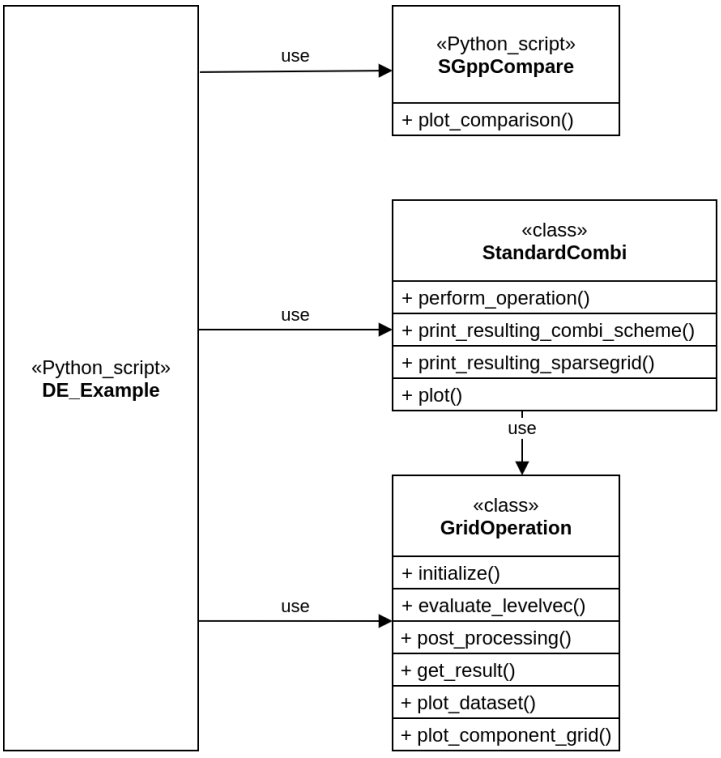


Figure 3.3.: Diagram of density estimation execution.

input : A level vector $lvec$ of size dim
output: The R-matrix for the component grid specified by the level vector

```
1  grid_size ← get_num_points(lvec);
2  index_list ← get_indexlist(lvec);
3  diag_val ← ∏_{k=0}^{dim} 2^{-(lvec[k]-1)} · 3^{-1};
4  if masslumping = false then
5      for i ← 0 to grid_size −1 do
6          for j ← i + 1 to grid_size do
7              result ← 1 for k ← 0 to dim do
8                  hat_i ← index_list[i][k];
9                  hat_j ← index_list[j][k];
10                 // hat_i and hat_j overlap fully
11                 if hat_i = hat_j then
12                     result ← result · (2^{-(lvec[k]-1)} · 3^{-1});
13                 else // hat_i and hat_j do not overlap
14                     left_index ← max((hat_i − 1) · 2^{lvec[k]-1)}, (hat_j − 1) · 2^{lvec[k]-1)});
15                     right_index ← min((hat_i + 1) · 2^{lvec[k]-1)}, (hat_j + 1) · 2^{lvec[k]-1)});
16                     if left_index ≥ right_index then
17                         result ← 0;
18                         break;
19                     else // hat_i and hat_j overlap partly
20                         result ← result · (2^{-(lvec[k]-1)} · 12^{-1});
21                     end
22                 end
23             end
24             if result ≠ 0 then
25                 R[i, j] ← result;
26                 R[j, i] ← result;
27             end
28         end
29     end
30 end
31 return R
```

Algorithm 1: Construction of the R-matrix of the linear system $(R + \lambda I)\vec{\alpha} = \vec{b}$.

input : A level vector *lvec* of size *dim* and a data set *data* of size M.
output: The $\vec{b}$-vector for the component grid specified by the level vector

```
1  grid_size ← get_num_points(lvec);
2  index_list ← get_indexlist(lvec);
3  for i ← 0 to M do
4  |   // Only evaluate at the basis functions in whose support the current
   |      data point lies
5  |   hat_functions ← get_hat_functions(lvec,data[i]);
6  |   for j ← 0 to len(hat_functions) do
7  |   |   b[index(hat_functions [j])] ← b[index(hat_functions [j])] +
   |   |    evaluate_hat_function(hat_functions [j],lvec,data[i]);
8  |   end
9  end
10 // Scale the b-vector by 1/M
11 b ← b · M^-1;
12 return b
```

Algorithm 2: Construction of the $\vec{b}$-vector of the linear system $(R + \lambda I)\vec{\alpha} = \vec{b}$.

input : A level vector $lvec$ of size dim and a data point x of size dim.
output: All the hat functions in whose support the data point x of size dim lies.

```
1  if point_not_zero(x) then
2      numb_points ← get_num_points_per_dim(lvec);
3      meshsize ← foreach element l of lvec do 2^{-l};
4      indices ← [];
5      for i ← 0 to dim do
6          lower ← ⌊ x[i] / meshsize[i] ⌋;
7          upper ← ⌈ x[i] / meshsize[i] ⌉;
8          if
            (lower > 0 ∧ lower ≤ numb_points[i]) ∧ (upper > 0 ∧ upper ≤ numb_points[i])
            then
9              indices[i] ← (lower, upper);
10         else if lower < 1 ∨ lower > numb_points[i] then
11             indices[i] ← (upper, );
12         else if upper < 1 ∨ upper > numb_points[i] then
13             indices[i] ← (lower, );
14         remove_duplicates(indices);
15         return indices × indices;
16     end
17 else
18     return [];
19 end
```

Algorithm 3: Returns all the hat functions in whose support the data point x of size dim lies.

4. Evaluation

In this chapter, the implementation is evaluated using various data sets. Two data sets were generated using scikit-learn and one using NumPy. The generated data sets are the circle data set, the two moon data set, for which the results can be found in the Apendix, and a multivariate Gaussian distribution. The other data set contains data gathered in the real world, from eruptions of the old faithful geyser in the Yellowstone National Park.
The solution of the density estimation for the full grid is compared with the sparse grid toolbox SG++, the implemented combination technique with and without mass lumping and with the best component grid of the combination scheme. It is compared with respect to the l_1, l_2 and l_∞ norm of the difference between the values at all grid points of the full grid and the values of the other methods at these points. Additionally, different combinations for the minimum and maximum level of the combination scheme are compared in terms of the number of grid points used as well as the error norms in relation to the full grid solution.
For the multidimensional Gaussian distribution, the output of the probability density function is used as a reference, since the use of the density estimation as the reference is not feasible with a 5-dimensional full grid. These values are then also compared with the density function estimates of SG++ and the implemented sparse grid combination technique with and without mass lumping. The comparison between different combinations of minimum and maximum values for the combination scheme is omitted here, since we then have to deal with combinations with a high minimum value with anisotropic full grids with a high number of grid points in each dimension that come close to the full grid, which means that we are faced with the curse of dimensionality. This chapter first explains the Sparse Grid Toolbox SG++ and its data mining pipeline, which was used for the evaluation, followed by the evaluation of the different data sets.

4.1. The General Sparse Grid Toolbox SG++

The General Sparse Grid Toolbox is a open-source toolbox that offers a variety of methods for spatially adaptive sparse grids and the combination technique. It was created by Dirk Pflüger as part of his dissertation [1] and is written in C++. In this thesis the module for data-driven tasks is used as one way to compare the estimation of a density function in the SG++ and sparseSpACE framework. The module provides an easy way to use the implemented machine learning methods by implementing a data mining pipeline.

4.1.1. The Data Mining Pipeline

The data mining pipeline provides an easy and uncomplicated way of using the implemented algorithms of the data-driven module and requires only a minimal setup in C++. JSON configuration files are used to configure the data mining pipeline.
The main attributes of the JSON configuration file is a data source, a scorer and a model fitter. The data source attribute contains a JSON dictionary that specifies the input data, manages which columns of the data should be considered, or what size each batch should have. The scorer attribute specifies the metric that should be used during training. The scorer then evaluates the model in relation to the metric specified in this attribute. In addition, the fitter configuration specifies the data mining task to be performed and the way it is to be carried out. Currently SG++ supports least square regression, density estimation and classification. Furthermore, this JSON attribute can include a grid configuration dictionary that specifies parameters such as grid type and grid level, a density estimation configuration that specifies parameters such as whether the conjugate gradient method or matrix decomposition should be used and, if so, what type of matrix decomposition, as well as a regularization configuration in which the regularization parameter λ can be specified.
The data mining pipeline offers many more configuration options than mentioned, which will be ignored in this paper because they are not relevant for our use of the density estimation method. One such JSON configuration file can be found in the appendix A.1. The JSON file is then used when creating a `SparseGridMiner` by passing it to a `DensityEstimationMinerFactory`. This miner factory builds the miner. Then all that has to be done is calling `miner.learn()`. We get the trained model by calling `miner.getModel()`. The model can then be evaluated at one or more data points with the method `evaluate()`.

4.2. The Circles Data Set

The circles data set was generated using scikit-learn. It contains 500 two-dimensional data points. Please refer to the appendix A.2 for a code snippet of the generation of this data set. Figure 4.1 shows the visualization of the circle data set. This data set was selected to illustrate that because we use piecewise linear basis functions, we get sharp edges and corners as a result of the density estimation. This can be mitigated, for example, by using B-Splines as basis functions [1]. The tests were performed with different values of λ. For λ values smaller than 0.01 we were overfitting to the data so the errors increased. Since $\lambda = 0.01$ had the smallest errors in the tests, we use this lambda for our evaluations. Results for other λ values can be found in the Apendix.

From Table 4.1 we can see that the errors of SG++ with respect to the full grid solution are usually the largest. This could be due to the fact that SG++ evaluates directly on the sparse grid based on the hierarchical basis and not with the combination technique. In addition, SG++ performs a number of optimizations by default. Overall, the standard combination technique achieves the best results in comparison to the full grid solution. When mass lumping is used, we get an improvement in speed, but also have an error about 50% higher for all three norms when compared to the combination technique without mass lumping.

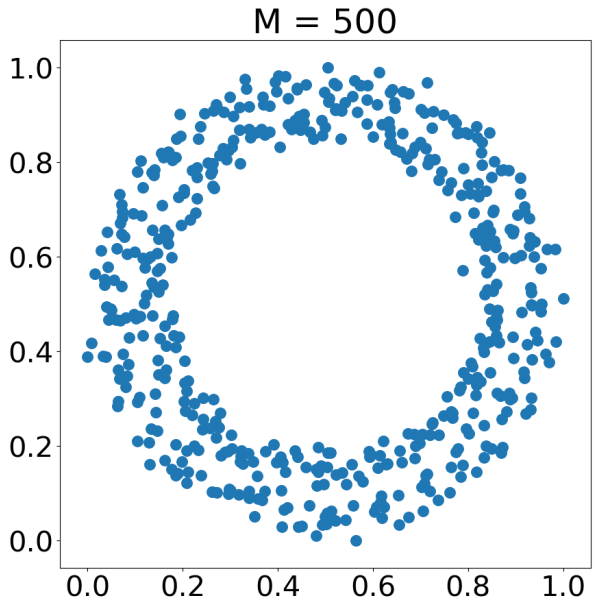


Figure 4.1.: Visualization of the circles data set generated with scikit-learn, containing $M = 500$ data points.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	46.06	7.82	2.19
	4	203.45	15.97	2.46
	5	914.44	36.81	3.28
Combination technique	3	16.18	3.08	1.11
	4	78.41	6.31	1.09
	5	223.13	9.09	0.99
Mass lumping	3	33.77	5.28	1.28
	4	152.66	11.52	1.70
	5	349.68	13.59	1.05
Best combination component	3	(3,1): 30.12	(1,3): 5.20	(2,2): 1.28
	4	(4,1): 101.76	(4,1): 8.11	(3,2): 1.53
	5	(5,1): 325.19	(5,1): 14.39	(4,2): 1.47

Table 4.1.: Comparison for the circle data set with $\lambda = 0.01$.

It is interesting to note that in most cases the best component grid of the combination scheme for each level outperforms the mass lumping approach. Furthermore, the component grids that minimize the l_1-norm and the l_2-norm of the difference to the full grid solution for $\lambda = 0.01$ are mostly those where one of the level vector components is one. These anisotropic full grids have a very fine mesh size in one dimension and a mesh size in the other dimension that spans the entire domain. The smallest maximum absolute difference is achieved with grids that have a more balanced number of points in both dimensions.

Table 4.2 shows a comparison of regular sparse grids with varying combinations of minimum and maximum levels of the combination technique. The combinations are compared to the solution of the density estimation for the full grid of their respective maximum level. If we invest more points by choosing a higher minimum level for the combination scheme, we obviously reduce the difference to the full grid since we are getting increasingly closer to the full grid. The N column contains the number of grid points of the regular sparse grid resulting from the combination scheme where a total of C points were used. For example, by using a $(2,5)$ sparse grid instead of a $(1,5)$ sparse grid, we roughly double the grid points used in the combination scheme, which also results in a sparse grid that has about twice as many grid points. This reduces the three errors by about 20% for this case.

max-level	combi	C	N	l_1-norm	l_2-norm	l_∞-norm
3	(1,3)	29	17	16.18	3.08	1.11
	(2,3)	51	33	8.71	1.99	0.79
4	(1,4)	95	49	78.42	6.31	1.09
	(2,4)	181	97	41.28	3.57	0.72
	(3,4)	259	161	18.46	1.69	0.40
5	(1,5)	273	129	223.13	9.09	0.99
	(2,5)	535	257	179.79	7.20	0.80
	(3,5)	869	449	128.95	5.72	0.63
	(4,5)	1155	705	49.72	2.36	0.28

Table 4.2.: Comparison of different combinations of the minimum and maximum level for the combination scheme for the circle data set with $\lambda = 0.01$.

Figure 4.2 shows the logarithmic-linear plot of the error norms decreasing by increasing the number of grid points.

4.3. The Old Faithful Data Set

The Old Faithful data set[1] is a well known data set on geyser eruptions, where each line represents an observed eruption of the Old Faithful Geyser in Yellowstone National Park. It contains 272 observations with 2 variables (dimensions). The data set visualization can be seen in Figure 4.3.

[1] `https://www.stat.cmu.edu/~larry/all-of-statistics/=data/faithful.dat`

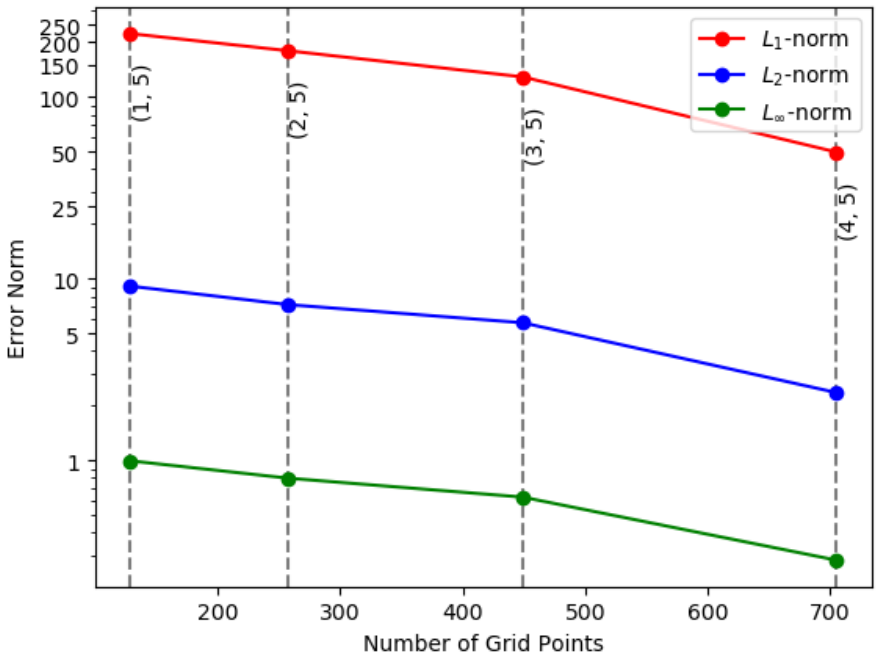


Figure 4.2.: Comparison of the error norms with the number of grid points used for the density estimation for the circle data set with $\lambda = 0.01$.

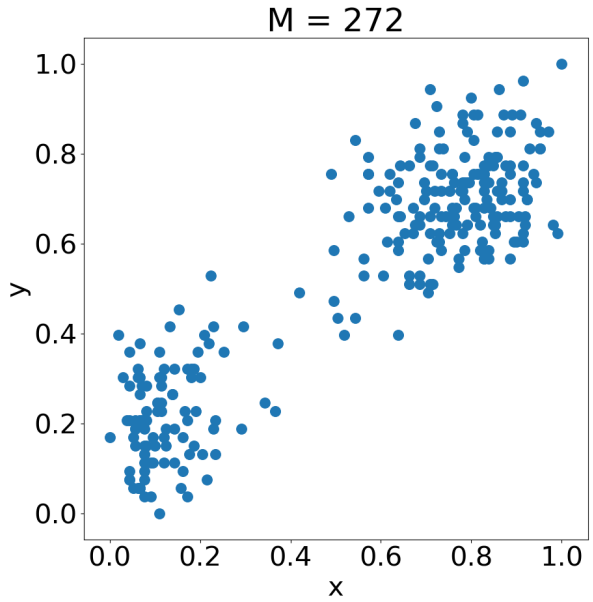


Figure 4.3.: Visualization of the old faithful geyser data set, containing $M = 272$ data points.

This data set was selected to show that the implementation can handle real world data besides generated data. For this data set the best results were also achieved by $\lambda = 0.01$. For λ values smaller than this value, the errors increased because we have fitted the data too much. Results for other λ values can be found in the Apendix.
Table 4.3 shows the result of the comparison with the full grid for the old faithful data set for $\lambda = 0.01$. Here it is interesting to note that some component grids of the combination scheme actually have a lower error than the combination technique for this level. If we look at the visualization of the data set in Figure 4.3, we see that we have a group of points in the upper right corner and another group of points in the lower left corner with some outliers in between. Because for the sparse grid we have more grid points along the axis, on which there are fewer data points for this data set, than is the case for the component grids, the error for the density calculated with the sparse grid combination technique increases. As a result, for example, the component grid for the level vector $(2, 2)$ has a smaller total error than the sparse grid combination technique for the level 3 case, and the anisotropic full grid component $(1, 5)$ of the level 5 combination scheme yields better results for all three norms compared to the level 5 sparse grid resulting from the combination scheme.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	66.09	13.70	6.68
	4	253.47	26.23	5.86
	5	971.57	54.41	7.36
Combination technique	3	30.69	5.94	2.49
	4	92.26	9.67	2.68
	5	298.03	14.84	3.43
Mass lumping	3	40.88	7.99	2.73
	4	151.41	15.69	3.61
	5	511.14	22.12	3.92
Best combination component	3	(2,2): 28.64	(2,2): 6.56	(2,2): 2.92
	4	(2,3): 104.36	(1,4): 10.34	(1,4): 2.62
	5	(1,5): 283.19	(1,5): 13.30	(1,5): 1.86

Table 4.3.: Comparison for the old faithful geyser data set with $\lambda = 0.01$.

Table 4.4 again shows of the trade off between investing more points and higher accuracy. The effect of investing more points by increasing the minimum level of the combination scheme on the l_1, l_2 and l_∞ norm errors decreases as the component grid approaches the full grid.

Figure 4.4 shows the logarithmic-linear representation of the error norms corresponding to Table 4.4, which decrease with a decreasing number of grid points.

max-level	combi	C	N	l_1-norm	l_2-norm	l_∞-norm
3	(1,3)	29	17	30.69	5.94	2.49
	(2,3)	51	33	9.11	2.09	1.07
4	(1,4)	95	49	92.26	9.67	2.68
	(2,4)	181	97	36.89	4.51	2.13
	(3,4)	259	161	17.73	2.49	1.19
5	(1,5)	273	129	298.03	14.84	3.43
	(2,5)	535	257	194.27	10.60	1.89
	(3,5)	869	449	127.59	7.94	1.39
	(4,5)	1155	705	49.33	3.18	0.60

Table 4.4.: Comparison of different combinations of the minimum and maximum level for the combination scheme for the old faithful geyser data set with $\lambda = 0.01$.

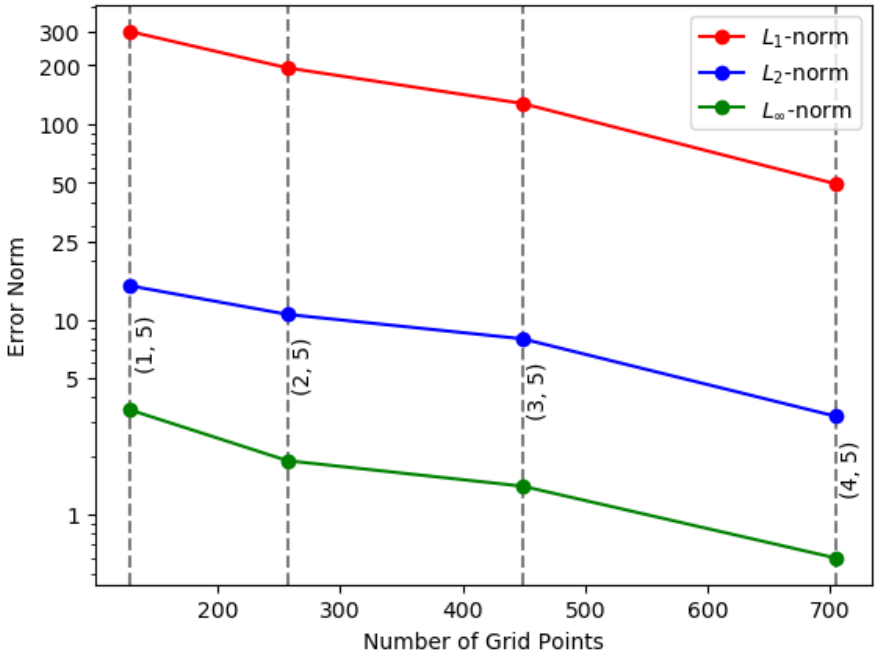


Figure 4.4.: Comparison of the error norms with the number of grid points used for the density estimation for the old faithful geyser data set with $\lambda = 0.01$.

4.4. A Multivariate Gaussian Distribution

The multivariate Gaussian distribution was generated using NumPy. A 3-dimensional and 5-dimensional data set was generated where 10000 and 100000 random data samples where drawn from the probability density function

$$f(x) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right), \tag{4.1}$$

where μ is the mean, Σ the covariance matrix and k is the dimension. A code snipped of the data set generation with the used parameters can be found in the appendix A.7. Since full grids for the dimensions ≥ 3 are not really feasible, in this section we take the output of the density function in Equation 4.4, which was sampled at all grid points of the sparse grid, as the reference. The regularization coefficient is set to 0.01. Table 4.5 shows the results for the comparison of a 3-dimensional Gaussian distribution with 10000 data points.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	64.51	12.66	4.09
	4	197.89	21.44	3.96
	5	558.87	35.17	4.04
Combination technique	3	59.85	11.29	3.51
	4	423.21	48.02	11.11
	5	1519.02	106.69	16.15
Mass lumping	3	47.85	9.29	3.02
	4	604.85	65.91	13.86
	5	2064.22	143.91	21.28

Table 4.5.: Comparison for a multivariate Gaussian distribution of dimension 3 with $\lambda = 0.01$.

Table 4.6 shows the results for the comparison of a 5-dimensional Gaussian distribution with 100000 data points. The errors are generally higher than in the previous 2-dimensional data sets, as we are working with more data points and more grid points due to the larger dimension. The SG++ toolbox achieves the best results in estimating the probability density function. We see that we need to increase the level of the sparse grid for this increased dimension and amount of data points in order to increase the number of grid points and the mesh size so that we can estimate the probability density function more accurately. What is interesting here is that the errors that occur in the combination technique of the sparse grid of level 4 with and without mass lumping are much higher than those of the sparse grid of level 5, which was never the case in the previous results. This means that the individual differences at the grid points must be much higher for the level 4 sparse grid, since the differences should decrease with a higher number of grid points. Furthermore, the mass lumping approach surprisingly achieves lower errors for level 5 than the combination technique.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	172.57	26.98	7.47
	4	799.81	57.81	9.26
	5	3138.47	112.82	9.63
Combination technique	3	1206.95	159.79	42.92
	4	3747.47	278.44	52.16
	5	3498.66	181.28	24.15
Mass lumping	3	1370.37	178.11	45.45
	4	3989.12	298.32	55.91
	5	2682.55	152.50	22.28

Table 4.6.: Comparison for a multivariate Gaussian distribution of dimension 5 with $\lambda = 0.01$.

5. Conclusion and Future Work

Density estimation with the combination technique was implemented and successfully integrated into the sparseSpACE framework. A new grid operation class was implemented that performs the density estimation on a sequence of small anisotropic full grids in the conventional nodal basis and combines them linearly. The evaluation showed that in order to accurately estimate density functions of higher dimensions, we have to invest more points by increasing the level. For some specific cases a component grid achieved better results in respect to a certain error norm than the sparse grid. The use of mass lumping in the density estimation process still achieves relatively good results compared to the standard combination method.
There are a number of things where the implemented density estimation method has a lot of potential for improvement. A later version utilizing spatially adaptive refinement or the dimensional adaptive combination technique can further increase the accuracy and reduce the number of grid points used. In addition, future implementations could use Offline/Online splitting to significantly speed up the calculation of the linear system in Equation 2.35 by precalculating the left side of the equation for a large number of grids, so that in the online phase only the right side needs to be solved. This could make the implementation more practical for higher dimensions. A further potential improvement is the parallelization of the calculation of the individual component grid solutions.

List of Figures

2.1. Basis functions $\varphi_{l,i}$ and grid points $x_{l,i}$ of level 3 (left) and a function $u \in V_3$ as a weighted sum of hat functions in the nodal basis. 5
2.2. Hat function $\varphi_{(2,1),(1,1)}$.[3] 6
2.3. Hat basis functions $\varphi_{l,i}$ centered on the respective grid points $x_{l,i}$ in the hierarchical basis for $l \leq 3$ (left) and in the nodal basis (right). 7
2.4. An interpolant $u(x) \in V_3$ (left) and its corresponding weighted basis functions (right) in the hierarchical basis. 8
2.5. Basis hat functions in two dimensions of the subspaces $W_{\vec{l}}$ for $l_j \leq 3$. [1] 9
2.6. The two-dimensional sparse grid space $V_n^{(1)}$ and its subspaces $W_{\vec{l}}$ up to level $n = 3$ (left) and the resulting sparse grid space (right).[1] 11
2.7. Example of spatial adaptive refinement process.[4] 11
2.8. An example full grid of level 3 without points on the boundary on the left, a regular sparse grid of leven 3 in the middle and a spatial adaptive sparse grid on the right. 11
2.9. Combination technique in two dimensions with points on the boundary. By adding the blue subspaces and subtracting the red ones, we obtain a level 4 sparse grid on the left and dimensionally adaptive sparse grid on the right. [1] 13
2.10. Example of the standard combination technique for the funnychess data set. . 13
2.11. Example of the classification process based on density estimation for the two moon data set. [5] 15
2.12. Example of the clustering process based on density estimation for the two moon data set. [5] 16

3.1. The density estimation (left) and corresponding contour representation (right) for the circle data set for a regular sparse grid of level 5 and coefficient $\lambda = 0.001$. 20
3.2. The combination scheme of level 4 in two dimensions and $\lambda = 0.001$ with contour plot of the density of each component grid for the circle data set. . . . 21
3.3. Diagram of density estimation execution. 22

4.1. Visualization of the circles data set generated with scikit-learn, containing $M = 500$ data points. 28
4.2. Comparison of the error norms with the number of grid points used for the density estimation for the circle data set with $\lambda = 0.01$. 30
4.3. Visualization of the old faithful geyser data set, containing $M = 272$ data points. 30
4.4. Comparison of the error norms with the number of grid points used for the density estimation for the old faithful geyser data set with $\lambda = 0.01$. 32

A.1. Example data mining pipeline configuration. 40
A.2. Generation of the circle data set using scikit-learn. 41
A.3. The density estimation (left) and corresponding contour representation (right) for the old faithful data set for a regular sparse grid of level 4 and coefficient $\lambda = 0.01$. 41
A.4. Visualization of the two moons data set generated with scikit-learn, containing $M = 500$ data points. 43
A.5. Generation of the two moons data set using scikit-learn. 43
A.6. Comparison of the error norms with the number of grid points used for the density estimation for the two moons data set with $\lambda = 0.01$. 44
A.7. Generation of the multivariate Gaussian data set using NumPy. 45
A.8. Example of a 2-dimensional Gaussian distribution being sampled at the grid points of a sparse grid of level 6. 45

List of Tables

4.1. Comparison for the circle data set with $\lambda = 0.01$. 28
4.2. Comparison of different combinations of the minimum and maximum level for the combination scheme for the circle data set with $\lambda = 0.01$. 29
4.3. Comparison for the old faithful geyser data set with $\lambda = 0.01$. 31
4.4. Comparison of different combinations of the minimum and maximum level for the combination scheme for the old faithful geyser data set with $\lambda = 0.01$. . . . 32
4.5. Comparison for a multivariate Gaussian distribution of dimension 3 with $\lambda = 0.01$. 33
4.6. Comparison for a multivariate Gaussian distribution of dimension 5 with $\lambda = 0.01$. 34

A.1. Comparison for the circle data set with $\lambda = 0.001$. 41
A.2. Comparison for the old faithful data set with $\lambda = 0.001$. 42
A.3. Comparison for the two moons data set with $\lambda = 0.0$. 42
A.4. Comparison for the two moons data set with $\lambda = 0.01$. 43
A.5. Comparison of different combinations of the minimum and maximum level for the combination scheme for the two moon data set with $\lambda = 0.01$. 44
A.6. Comparison for a 2-dimensional Gaussian distribution 2 with $\lambda = 0.01$. . . . 46

Bibliography

[1] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. München: Verlag Dr. Hut, Aug. 2010. ISBN: 978-3-86853-555-6. URL: http://www5.in.tum.de/pub/pflueger10spatially.pdf.

[2] J. Valentin. "B-Splines for Sparse Grids: Algorithms and Application to Higher-Dimensional Optimization". In: *arXiv:1910.05379 [cs, math]* (2019). arXiv: 1910.05379. DOI: 10.18419/opus-10504. URL: http://arxiv.org/abs/1910.05379 (visited on 02/27/2020).

[3] J. Garcke. "Sparse Grids in a Nutshell". In: *Sparse Grids and Applications*. Ed. by J. Garcke and M. Griebel. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 57–80. ISBN: 978-3-642-31703-3.

[4] B. Peherstorfer, D. Pflüge, and H.-J. Bungartz. "Density Estimation with Adaptive Sparse Grids for Large Data Sets". en. In: *Proceedings of the 2014 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Apr. 2014, pp. 443–451. ISBN: 978-1-61197-344-0. DOI: 10.1137/1.9781611973440.51. URL: https://epubs.siam.org/doi/10.1137/1.9781611973440.51 (visited on 11/16/2019).

[5] B. Peherstorfer. "Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques". Dissertation. Department of Informatics, Technische Universität München, Oct. 2013.

[6] B. Peherstorfer, F. Franzelin, D. Pflüger, and H.-J. Bungartz. "Classification with Probability Density Estimation on Sparse Grids". In: *Sparse Grids and Applications - Munich 2012*. Ed. by J. Garcke and D. Pflüger. Cham: Springer International Publishing, 2014, pp. 255–270. ISBN: 978-3-319-04537-5.

[7] B. Peherstorfer, D. Pflüger, and H.-J. Bungartz. "Clustering Based on Density Estimation with Sparse Grids". In: *KI 2012: Advances in Artificial Intelligence*. Ed. by B. Glimm and A. Krüger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 131–142. ISBN: 978-3-642-33347-7.

[8] H.-J. B. Michael Obersteiner. "A Spatially Adaptive Sparse Grid Combination Technique for Numerical Quadrature". en. In: *Sparse Grids and Applications - Munich 2018*. Springer Verlag, Jan. 2019.

A. Apendix

A.1. Configurations and Test Results

A.1.1. SG++ configuration

```
{
    "dataSource": {
        "filePath": "Datasets/twomoons.arff",
        "hasTargets": false,
        "readinColumns": [0,1]
    },
    "scorer": {
        "metric": "mse"
    },
    "fitter": {
        "type": "densityEstimation",
        "gridConfig": {
            "gridType": "linear",
            "level": 4
        },
        "regularizationConfig": {
            "lambda": 0
        },
        "densityEstimationConfig": {
            "densityEstimationType": "decomposition",
            "matrixDecompositionType": "orthoadapt"
        }
    }
}
```

Figure A.1.: An example for a JSON data mining pipeline configuration file.

A.1.2. The Circle Data Set

```
from sklearn import datasets
data = datasets.make_circles(500, noise=0.1)
```

Figure A.2.: Generation of the circle data set using scikit-learn.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	71.16	12.19	3.56
	4	261.40	21.35	3.77
	5	1073.55	45.33	4.91
Combination technique	3	25.72	4.81	1.69
	4	134.67	11.49	2.53
	5	570.64	26.01	3.31
Mass lumping	3	45.05	8.04	2.22
	4	276.43	22.52	4.07
	5	1266.58	52.24	4.64
Best combination component	3	(1,3): 48.37	(1,3): 9.10	(2,2): 2.76
	4	(3,2): 179.02	(2,3): 15.13	(1,4): 2.87
	5	(3,3): 623.73	(3,3): 28.30	(3,3): 2.99

Table A.1.: Comparison for the circle data set with $\lambda = 0.001$.

A.1.3. The Old Faithful Data Set

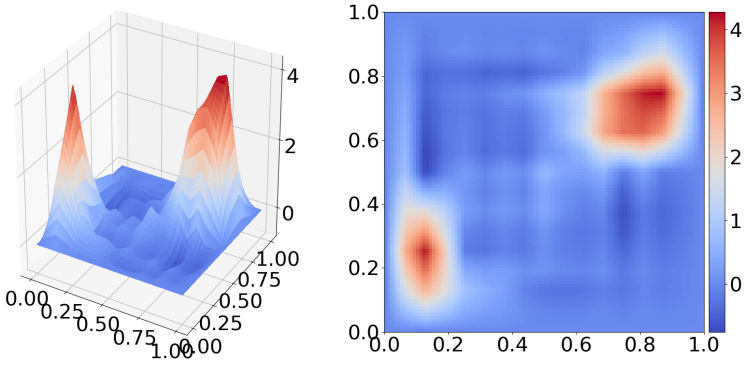


Figure A.3.: The density estimation (left) and corresponding contour representation (right) for the old faithful data set for a regular sparse grid of level 4 and coefficient $\lambda = 0.01$.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	96.63	20.87	8.83
	4	391.74	40.88	10.03
	5	1437.15	74.83	10.22
Combination technique	3	51.40	10.79	5.46
	4	146.44	16.57	6.97
	5	647.67	34.86	8.86
Mass lumping	3	68.29	13.42	5.45
	4	285.02	29.81	6.93
	5	1246.76	72.92	11.83
Best combination component	3	(2,2): 56.92	(2,2): 11.69	(2,2): 6.05
	4	(3,2): 148.35	(3,2): 18.34	(3,2): 7.33
	5	(3,3): 605.81	(4,2): 37.86	(3,3): 6.91

Table A.2.: Comparison for the old faithful data set with $\lambda = 0.001$.

A.1.4. The Two Moons Data Set

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	98.87	17.44	6.91
	4	380.96	34.28	7.78
	5	2094.89	97.38	20.09
Combination technique	3	60.42	11.29	4.42
	4	252.15	23.87	5.51
	5	1403.85	72.37	16.09
Mass lumping	3	76.63	13.99	5.58
	4	375.08	30.99	6.18
	5	1897.60	86.61	15.28
Best combination component	3	(3,1): 70.51	(3,1): 13.25	(1,3): 4.03
	4	(3,2): 297.02	(3,2): 27.80	(1,4): 6.67
	5	(3,3): 1455.83	(3,3): 77.08	(4,2): 17.05

Table A.3.: Comparison for the two moons data set with $\lambda = 0.0$.

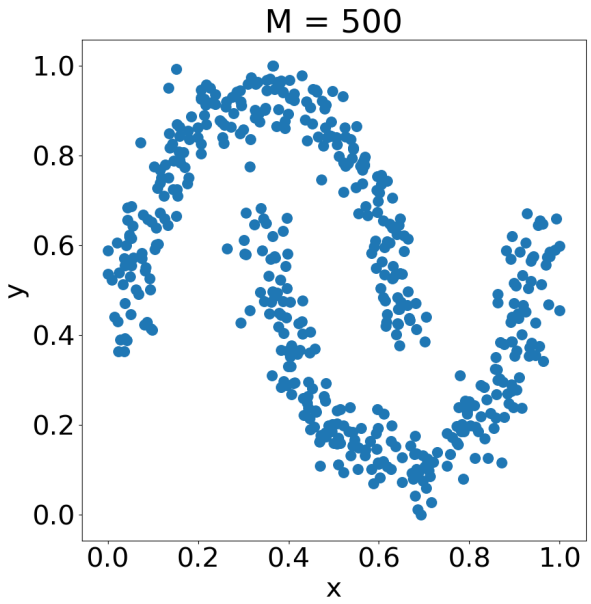


Figure A.4.: Visualization of the two moons data set generated with scikit-learn, containing $M = 500$ data points.

```
from sklearn import datasets
data = datasets.make_moons(500, noise=0.1)
```

Figure A.5.: Generation of the two moons data set using scikit-learn.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	48.18	8.36	2.69
	4	201.68	16.30	2.75
	5	920.76	37.48	3.10
Combination technique	3	25.09	4.69	1.84
	4	87.36	7.39	1.38
	5	296.99	12.05	1.41
Mass lumping	3	34.38	6.28	2.51
	4	126.08	10.49	1.74
	5	461.57	18.04	1.66
Best combination component	3	(3,1): 32.25	(1,3): 5.64	(1,3): 1.76
	4	(1,4): 109.28	(1,4): 8.92	(4,1): 1.58
	5	(3,3): 1455.83	(3,3): 77.08	(4,2): 17.05

Table A.4.: Comparison for the two moons data set with $\lambda = 0.01$.

max-level	combi	C	N	l_1-norm	l_2-norm	l_∞-norm
3	(1,3)	29	17	25.09	4.69	1.84
	(2,3)	51	33	12.44	2.67	1.27
4	(1,4)	95	49	87.36	7.39	1.38
	(2,4)	181	97	50.12	4.43	1.01
	(3,4)	259	161	21.51	1.96	0.43
5	(1,5)	273	129	296.99	12.05	1.41
	(2,5)	535	257	189.65	8.03	0.86
	(3,5)	869	449	130.41	5.79	0.63
	(4,5)	1155	705	49.57	2.45	0.32

Table A.5.: Comparison of different combinations of the minimum and maximum level for the combination scheme for the two moon data set with $\lambda = 0.01$.

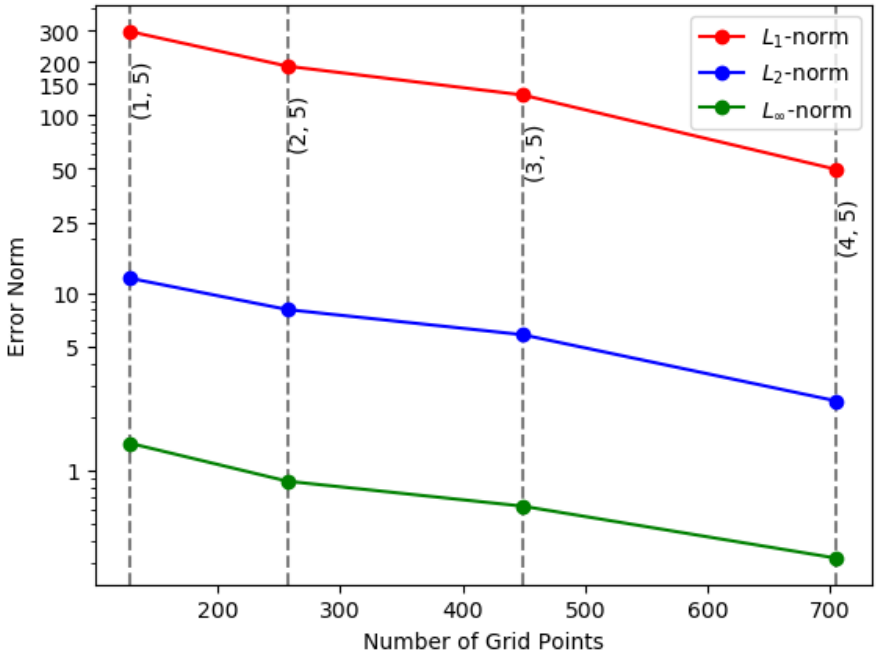


Figure A.6.: Comparison of the error norms with the number of grid points used for the density estimation for the two moons data set with $\lambda = 0.01$.

A.1.5. A Multivariate Gaussian Distribution

```
import numpy as np
dim = 5
size = 100000
mean = np.array([0.5] * dim)
sigma = np.array([0.25]*dim)
cov = np.diag(sigma**2)
data = np.random.multivariate_normal(mean, cov, size)
```

Figure A.7.: Generation of the multivariate Gaussian data set using NumPy.

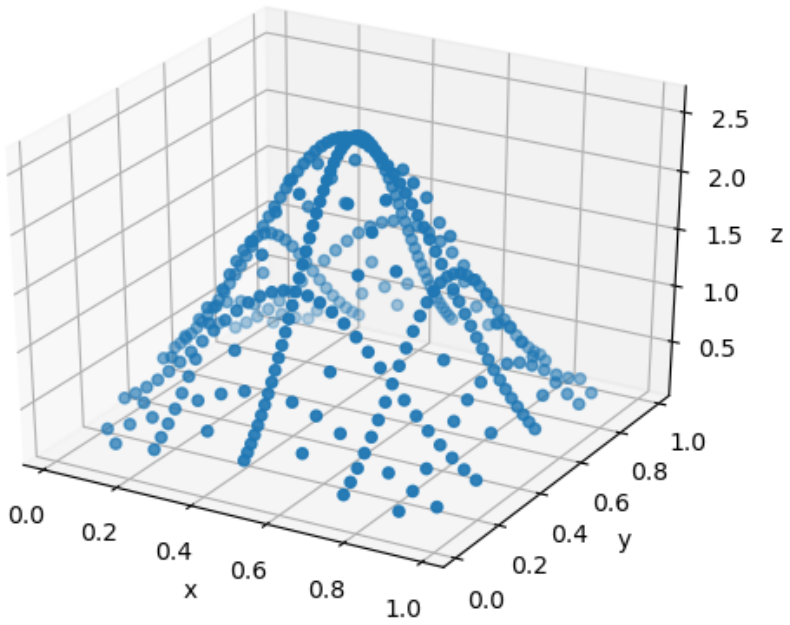


Figure A.8.: Example of a 2-dimensional Gaussian distribution being sampled at the grid points of a sparse grid of level 6.

	Level	l_1-norm	l_2-norm	l_∞-norm
SG++	3	11.00	3.33	1.45
	4	34.95	6.19	1.89
	5	101.15	10.89	2.27
Combination technique	3	16.55	4.65	2.47
	4	37.42	5.82	1.56
	5	170.95	16.97	3.18
Mass lumping	3	25.78	9.27	5.34
	4	45.01	7.05	2.19
	5	233.04	22.84	3.84

Table A.6.: Comparison for a 2-dimensional Gaussian distribution 2 with $\lambda = 0.01$.