Fakultät für Informatik
Technische Universität München

Analysis and Performance Engineering for Learning with Sparse Grids

Paul-Cristian Sârbu

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

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 03.01.2022 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 12.04.2022 angenommen.

Abstract

The past decade's explosion in the amount and diversity of available data has had a considerable impact on the field of machine learning. This increase in size, dimensionality, and variety of datasets came not only with new opportunities, but also with significant challenges, like the so-called "curse of dimensionality". Sparse grids are a numerical approach that can offer relief in this regard and have already been successfully used in various learning scenarios, however many aspects remain to be examined.

In this work, we explore and extend the use of spatially adaptive sparse grids for several specific learning tasks, examining the relationships between algorithm development, high-performance aspects, and user-oriented implementation. To this end, our contributions are multiple. In the area of density estimation, we introduce several new sparse grid-based algorithms which solve the problems of estimating differences, ratios, partial derivatives, and partial derivative ratios of probability densities, providing analysis of their theoretical properties and numerical accuracy. Our various tests show that these new methods can not only equal, but also surpass existing kernel-based approaches. In the area of classification and regression with sparse grids, we focus on high-performance aspects of code implementation, as well as their practical application. Specifically, we provide a code optimization study of a legacy regression/classification implementation for the recent Knight's Landing architecture, obtaining speed-ups versus older architectures comparable to those obtained in literature on similar computational kernels. The same regression code is then used on real-world financial data to improve the runtimes in our study of time series prediction using spatially adaptive sparse grids. We show there that modified linear basis functions can produce significantly better results than those obtained previously with the regular linear basis and the Combination Technique. Lastly, we bring contributions also to the area of sparse grid-based clustering, where we take a user-oriented approach to adapting, developing, and implementing two new methods, used for finding and analyzing deterministic, respectively uncertain, clusters in data. Tests on artificial and real datasets show that our approaches ease the ability of users to gain insight into the structure of datasets by providing compelling visual aid and relevant metrics for deterministic and statistical analysis.

All these results thus advance the field of spatially adaptive sparse grids, as applied to learning tasks, answering previously raised questions and introducing brand new avenues of future research.

Acknowledgements

First of all, I would like to address my deep gratitudes to my supervisor Hans-Joachim Bungartz for his invaluable and continuous support and advice throughout my dissertation work, for his openness and availability, and for fostering a great environment at his chair of Scientific Computing.

I would like to thank all past and present members of this chair for making it the inclusive, open, relaxed, and productive place to work I have known while being a part of it. There are many individual current and past colleagues to thank who have made this working experience so much more enjoyable by giving me advice, both professional and personal, by allowing me to bounce ideas off of, by equally helping me have fun when I needed to take my mind off of things and helping me to press on and push through when I would maybe not have been able to do it alone. I would like to mention here in particular Ionuț Farcaș, Michael Obersteiner, Steffen Seckler, and Severin Reiz.

I would like to express my gratitude to the whole Sparse Grids groups at Munich and Stuttgart for their time and availability in helping me delve into this topic and understand its various intricacies through many fruitful discussions. Especially here to mention are Kilian Röhner, Ionuț Farcaș, Michael Obersteiner, David Pfander, Valeriy Khakhutskyy, and Alfredo Parra.

I would be remiss not to mention the contributions from all the students who have collaborated with me during their student projects, whose work helped explore the various research directions which ended up shaping my thesis. I would like to also gratefully acknowledge the Leibniz Supercomputing Center for providing computing time and support on its Linux-Cluster for the high-performance simulations in this work.

Special gratitudes go to all the wonderful teachers and advisors I have had the good fortune of having throughout my studies in Romania and in Germany, who have made a lasting impact both on me as a person and on my academic pursuits and without whom I might not have taken the path of a PhD project. I would also like to thank the German Academic Exchange Service (DAAD) for their scholarship, which helped fund my stay in Germany during my Master's degree.

Last but very much not least, nothing really would have been possible without the support of dear friends and family. To my mother and brother, you have all my love and thanks for always being there for me, for your care, wisdom, and sacrifices. To my late father, who never got to see me follow in his footsteps by pursuing a PhD of my own, you have my eternal gratitude for your unconditional love and support throughout my life.

Contents

1 Introduction

A true boom in the amount, dimensionality, and sheer diversity of data available and required to learn from took place over the past decade. This information explosion has aided in the evolution of machine learning into an essential, vibrant, and complex field of study, with far-reaching and varied applications in many aspects of everyday life. However, this increase in both amount and complexity of the available datasets for learning tasks has brought to the front new challenges, one of the most prevalent being the so-called "curse of dimensionality" [6]. This term encompasses issues deriving from the exponential decrease in representation and prediction reliability encountered with the increase of data dimensionality, in the absence of an equally exponential increase in the amount of available data. Sparse grids, devised by Smolyak [76] and popularized later by Zenger [91], are a numerical method that tries to lessen the negative impact of this curse. In this thesis, we explore and expand the applicability of spatially adaptive sparse grids in various machine leaning scenarios.

While sparse grids were (re)introduced in the contemporary era in the context of solving differential equations, it soon became apparent that this numerical method had the potential for a far wider reach. We can trace back some of the earliest algorithms dealing with sparse grids in learning tasks specifically to the works of Garcke and Griebel. Their initial directions of study were geared on dimension-adaptive solutions to the regression/classification scenarios [28]. Further work followed suit, with focus on different basis functions [26] and an extension of these learning scenarios to the semi-supervised setting using the intrinsic geometric characteristics of the input datasets [27], results that Garcke coalesced in his PhD thesis [40]. He followed this with a better combination technique approach for solving the regression problem on sparse grids [22]. It has to be noted that the majority of these early solutions were designed with dimensionally adaptive sparse grids in mind, i.e., using the combination technique. Since then other interesting contributions have been made, for example an integrated preprocessing step of input data transformation to better fit the intrinsic axis-aligned grid point distribution of the sparse grids [10]. Solutions to the regression/classification problem have seen a big boost with the introduction of the spatially adaptive sparse grids, Pflüger proving that this approach can deliver high accuracies in these learning scenarios [64]. Algorithmically, this sparse grid approach to regression/classification has not changed significantly since, although its use-cases have been extended. Relevant to our work, Garcke et al. [9, 25] combined the delay embedding technique with sparse grid-based regression to successfully address the problem of time series prediction.

Another machine learning topic of interest in the community is density estimation. First algorithmic contributions in this direction came from Peherstorfer [58], who first introduced sparse grid density estimation by adapting a spline-smoothing approach [60]. This has opened further research avenues, as both density-based clustering [61] and density-based classification [59] could then be performed using sparse grids, however no concrete steps have since been taken into investigating further the application of sparse grids to estimating more complex quantities than a single density.

It goes without saying that performance considerations cannot be made independently of the algorithmic development. Pflüger gathered his contributions into a library called SG++[1], which has since expanded to contain many different modules pertaining to various applications. Generic node level optimizations and vectorizable code have been included from the start in order to obtain fast time-to-solution [64]. More hardware-aware approaches for the regression/classification problem have since been implemented by Heinecke [34], who used architecture-specific code (intrinsics) and manual vectorization techniques to optimize the vector-matrix multiplications required by the sparse grid solution for certain Intel processors. Although subsequently Pfander et al. [63] introduced a new subspace-based approach to obtain even better performance on CPUs and GPUs via an OpenCL implementation, the approach of Heinecke remains an interesting study case which so far was not fully explored in relation to newer CPU architectures.

In terms of an open-source, consolidated code base, SG++ is one of the largest (if not the largest) software libraries for sparse grids and its various applications that can be currently found. Since its inception, this toolbox has been constantly improved, expanded, and used as the basis of work on numerous theses and publications[2]. Most of its modules, ranging from basic sparse grid tasks like interpolation, to solving PDEs, quadrature, or optimization problems, are constructed and coded to be easily understood by future developers and to be easily used by even the less experienced users. In the machine learning module however, the data mining submodule stands out by its design as an easy to use, intuitive, and encompassing solution to various learning tasks using sparse grids. Röhner [69] took the density-based estimator and classifier of Peherstorfer, improved on both the algorithmic and performance aspects, added new functionality, and encapsulated everything in a customizable, extensible, and user-friendly framework. However, while sparse grid-based solutions to supervised learning tasks, such as density estimation and classification, which do not particularly require intrinsically user or expert knowledge in their assessment, can profit from this user-oriented nature of the data mining pipeline, the same cannot be said about supervised learning tasks, like clustering, where aiding a user in order to make informed decisions based on the results of the learning process can be critical.

[1]`https://sgpp.sparsegrids.org/`

[2]A non-exhaustive selection of such contributions can be found at: `https://sgpp.sparsegrids.org/publications/`

Our contributions in this thesis revolve around addressing exactly these three research directions mentioned. Firstly, we explore sparse grid-based solutions to the problem of estimating certain functions (e.g., differences, ratios, derivatives) of densities. We focus here on devising good quality, mathematically sound algorithmic solutions, getting our queues from existing kernel-based counterparts which have proven successful in various applications. Secondly, we investigate the capabilities of the legacy high-performance code of Heinecke to unlock the computational potential of one of the newest CPU architectures through code optimization, in order to obtain an improved time-to-solution for regression tasks. With the ability of such an optimized code to handle larger datasets, we also subsequently address the question of using spatially adaptive sparse grids for the problem of time series prediction in the context of currency exchange rates, where up to now only non-adaptive sparse grids using the combination technique were investigated. Lastly, we address the lack of a user-oriented sparse grid clustering software solution by integrating into the data mining pipeline and then extending the approach of Peherstorfer with new functionality. This results in two significant contributions: a practical hierarchical clustering algorithm and, to our knowledge, the first attempt at an uncertain clustering approach based on the standard sparse grid method.

Intrinsically, these three research directions lend themselves to be pursued each with a focus on either the algorithmic development aspects, the performance of the implemented numerical solution, or the user-oriented design and implementation decisions that create easy-to-use software. Throughout the thesis, while each of our contributions will be presented mostly from one of these three perspectives outlined, we will constantly be making references to the other two as needed, reinforcing the reality that these facets, while significant on their own, are (or at least should be) intertwined to various degrees in any good implementation of any numerical solution. Using different versions of the SG++ library as our basis for our implementations should already hint to the importance we place on this marriage of improved algorithms, good performance, and increased usability. Our hope is that this will also be one of the underlying take-aways of our work, besides our main, strictly scientific, contributions to the field of sparse grid-based learning.

The rest of the thesis follows a well-defined, natural structure. Chapter 2 lays out in the first part the theoretical foundations of spatially adaptive sparse grids, with which in the second part we present the main building blocks on which our work is constructed, i.e., the existing sparse grid algorithmic and numerical solutions to the different learning tasks. The following Chapters 3 to 5 will each be dedicated to one of our three research directions stated above and each with their own focus on different aspects of numerical software design and implementation. Finally, we will conclude our work and provide an outlook on future contributions in Chapter 6.

2 Foundations of Sparse Grids and Their Applications

As mentioned already in Chapter 1, the curse of dimensionality is noted as one of the main obstacles pertaining to the problem of handling high-dimensional functions numerically on a grid. Sparse grids are a method of delaying the onset of this "curse" (something which, even though highly desired, cannot unfortunately be completely avoided with current approaches). At its core, the idea of sparse grids is to create a suitable trade-off between the number of grid points required to approximate a function and the error such an approximation incurs, while keeping enough useful properties of the underlying data structure to allow efficient storing and evaluation.

The concept of such a method has quite a long history, although its first proper mathematical derivation is usually regarded to be the one given by Smolyak [76]. The method started its modern revitalization with the works of Zenger, who coined the term "sparse grids" [91]. Most early work on hierarchical basis functions for sparse grids and their applications was done through the contributions of Bungartz and Griebel [11]. There has since been no short supply of work done in extending the range of applications of sparse grids to diverse fields, such as interpolation [11, 74], quadrature [29], machine learning tasks [28, 64, 58, 69], optimization [84], PDEs [11, 91] or uncertainty quantification [21, 17]. In terms of software solutions, the most relevant for our contributions is the continuous development of the SG++ library, initially set up through the work of Pflüger [64] and continuously developed since, however we would be remiss not to mention the existence of other sparse grid toolboxes [5, 45, 77].

There is nowadays no short supply of resources when it comes to learning about sparse grids in general and spatially adaptive sparse grids in particular (e.g., [11, 64, 23], to name but a few). This chapter thus does not intend to reinvent the wheel in that regard, however, as we will treat different existing and new applications of sparse grids, some of which will require additional mathematical derivations, we consider important to begin with a uniform treatment of the underlying mathematical formulation. For the most part we employ a similar notation to the one in [83] and a perspective on hierarchical sparse grids similar to [69], which we found the most suitable for the purposes of our thesis.

The chapter is structured in two parts. In the first section the focus will be on introducing the notions common to all sparse grid-based learning methods, with application-related specifics being treated in the second part. In this way, the following chapters will be able to more clearly focus on our specific contributions.

2.1 Introduction into Spatially Adaptive Sparse Grids

2.1.1 Hierarchical Representation of Grid Points

The one-dimensional case. Without loss of generality, up to a linear transformation of coordinate, we consider our computational domain to be the compact unit interval $\Omega = [0, 1]$, which is a standard assumption made in the context of sparse grids. To discretize such a domain the straight-forward approach would be to generate equally-spaced grid points. We will restrict ourselves for now to zero-boundary functions, thus we can ignore grid points on the boundary $\partial\Omega$. (The boundaries will be treated in their own section.) For a given discretization level $l \in \mathbb{N}_0$ we can thus define $2^l - 1$ points given by

$$x_{l,i} = i \cdot 2^{-l}, \quad 1 \leq i < 2^l. \tag{2.1}$$

These grid points are uniquely determined by their level-index pair (l, i), and we can use them to, e.g., compute an interpolant of a function r passing through points $r(x_{l,i})$.

This same set of grid points can however be constructed also in a hierarchical manner. This stems from the precise position of grid points that we allow and the observation that specific grid points at lower levels appear as grid points at higher levels. We can therefore define the *hierarchical level-index* of a grid point (l, i) as the pair (l', i') that satisfies the conditions

$$\begin{cases} x_{l',i'} = x_{l,i}, \\ x_{l',i'} = x_{l'',i''} \Rightarrow l'' \geq l', \ \forall l'', i'' \in \mathbb{N}_0 \text{ subject to Eq. (2.1)}. \end{cases} \tag{2.2}$$

The uniqueness of such a pair can be quite easily proven: l' is the lowest level at which grid point $x_{l,i}$ can appear, and i' is its unique identifier at level l' based on Eq. (2.1). To distinguish between the two representations, we will call (l, i) the *nodal level-index pair*. Fig. 2.1 shows the relationship between the two representations for level 4 discretization grid points on the unit interval. Additionally, simple formulas that can compute the hierarchical level-index pair from the nodal ones, and vice versa, exist [83].

Fig. 2.1 shows that the set of grid points of nodal level l are nothing more than the reunion of all the sets of grid points of hierarchical levels $l' \leq l$. Moreover, in each hierarchical level l' the indices i' that can exist are odd integers, forming the set

$$I_{l'} = \left\{ 1 \leq i' < 2^{l'} \mid i' \equiv 1 \pmod 2 \right\}. \tag{2.3}$$

The multi-dimensional case. The generalization to the domain $\Omega = [0, 1]^d$ can be easily done by extending the definition of the nodal and hierarchical representations, respectively, to the d-dimensional case. We will however now discuss about a discretization (nodal) *level vector* $\boldsymbol{l} \in \mathbb{N}_0^d$ which produces grid points

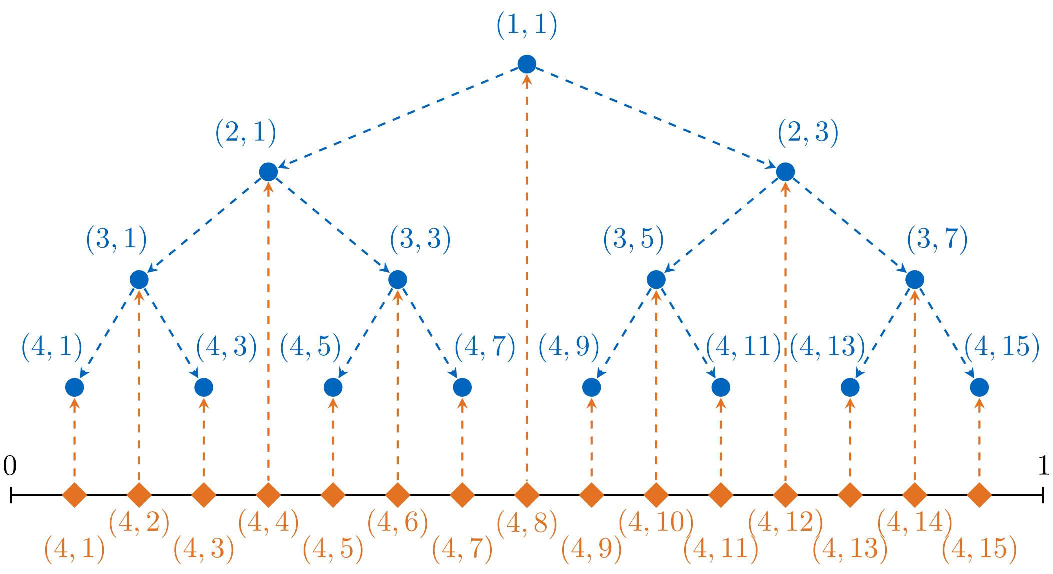


Figure 2.1: Example of one-dimensional discretization level 4 grid points inside the unit interval. The orange diamonds mark the nodal level-index pairs (l, i), and the blue circles mark the corresponding hierarchical level-index pairs (l', i').

$$\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}} = (x_{l_1,i_1}, \ldots, x_{l_d,i_d}), \quad 1 \leq i_k < 2^{l_k}, \quad k = 1, \ldots, d. \tag{2.4}$$

In order to obtain the corresponding hierarchical multi-index sets, we apply the Cartesian product on the univariate sets of each dimension:

$$\boldsymbol{I}_{\boldsymbol{l}'} = I_{l'_1} \times \cdots \times I_{l'_d}. \tag{2.5}$$

The same as for the one-dimensional case, the set of grid points of hierarchical level $\boldsymbol{l}'$ is given by the multi-level-index pairs $\{(\boldsymbol{l}', \boldsymbol{i}') \mid \boldsymbol{i}' \in \boldsymbol{I}_{\boldsymbol{l}'}\}$. Surprisingly, this Cartesian product of index sets is enough to maintain the fundamental property of the hierarchical representation: the set of grid points of nodal level vector $\boldsymbol{l}$ is given by the reunion of all sets of grid points with hierarchical level vectors $\boldsymbol{l}' \leq \boldsymbol{l}$, where

$$\boldsymbol{l}' \leq \boldsymbol{l} \Leftrightarrow l'_k \leq l_k, \; k = 1, \ldots, d. \tag{2.6}$$

Hierarchical relationships. There are some direct implications of this hierarchical representation. Given the tree-like structure of the univariate grid points in terms of levels, it is only natural that graph theory terminology was imported in the sparse grid vocabulary. Therefore, the one-dimensional level-index pairs (l^c, i^c) and (l^p, i^p) are in a *child-parent relationship* if

$$\begin{cases} l^p = l^c - 1, \\ i^p = 2 \cdot \left\lfloor \dfrac{i^c}{4} \right\rfloor + 1, \end{cases} \tag{2.7}$$

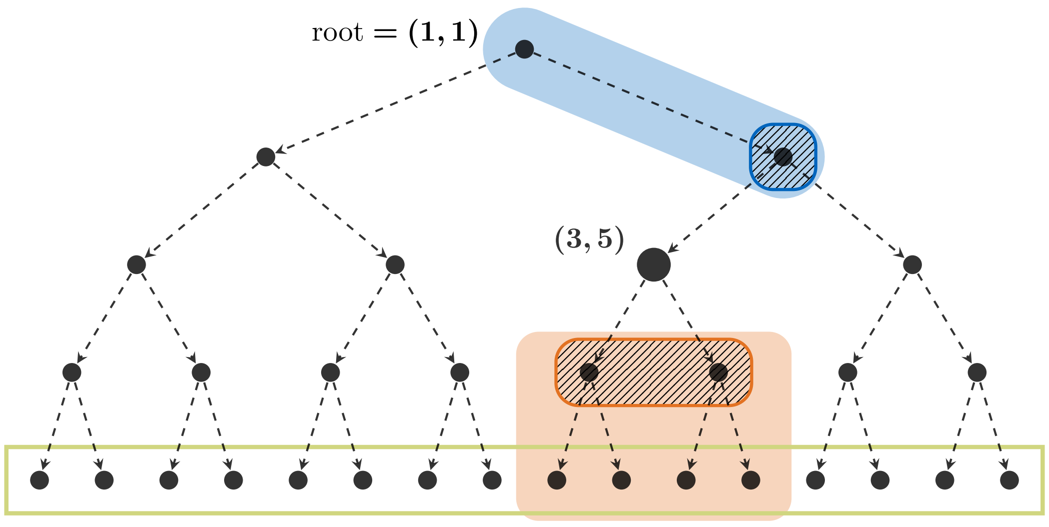


Figure 2.2: Example of relationships in the univariate grid point hierarchy of discretization level 5. With point $(3, 5)$ as reference, its ancestors and descendants are contained in the blue and orange areas, respectively, marking in each via hashed areas the direct parent and the two children, respectively. All grid points inside the green rectangle are leaves. To avoid clutter, we only show the hierarchical level-index pairs of the reference grid point and of the root $(1, 1)$.

or, alternatively,

$$\begin{cases} l^c = l^p + 1, \\ i^c = 2 \cdot i^p \pm 1. \end{cases} \tag{2.8}$$

The two formulations are given for completeness, as they are otherwise equivalent. Usually we are given one of the level-index pairs and are looking for the other, therefore one or the other formulation will be useful at any given time. Based on this relationship, grid point x_{l^p,i^p} is called the *hierarchical parent* of grid point x_{l^c,i^c}, and, conversely, x_{l^c,i^c} is the *hierarchical child* of grid point x_{l^p,i^p}.

Some interesting properties stem from these definitions. Firstly, $x_{1,1}$ is the only grid point that has no parent, so it can be considered the *root* of the hierarchy. There are grid points that have less than two children; these are called *leaves*. Secondly, in this one dimensional case each grid point has at most one parent (with the root having none) and at most 2 children (leaves have none; in an incomplete hierarchy grid points with only one child can exist). Lastly, the child-parent relationship can be applied recursively. Starting with a given grid point, going up the hierarchy from parent to parent up to the root we get the set of *hierarchical ancestors*. Similarly, starting at a given grid point and going down the hierarchy from child to child, considering always both children until we reach only leaves, we get the set of *hierarchical descendants*. Fig. 2.2 shows some of these univariate relationships on a level 5 discretization grid.

To extend these concepts to the d-dimensional case, all we have to do is define the child-parent condition for multi-level-index pairs $(\boldsymbol{l}, \boldsymbol{i})$, as all other terms are defined in relation to this concept. Therefore, the multi-level-index pairs $(\boldsymbol{l}^c, \boldsymbol{i}^c)$ and $(\boldsymbol{l}^p, \boldsymbol{i}^p)$ are in a child-parent relationship if the univariate level-index pairs (l_k^c, i_k^c) and (l_k^p, i_k^p) respectively are in a child-parent relationship as described by Eq. (2.7) or, alternatively, Eq. (2.8).

Using this definition, we can see that in the multivariate case grid point $x_{\mathbf{1}}$ is the root. Regarding the number of parents and children, respectively, of a generic grid point, they now increase proportional to d: any grid point except the root has d parents, one in each dimension, and any grid point except the leaves can have up to $2d$ children, two in each direction. While the notion of descendants of a grid point can still be easily understood and even visualized, the ancestors of a grid point now form a more complex set. This fact comes into play when one would want to implement a hierarchical traversal of all grid points, useful for evaluating on the grid or when discussing spatial refinement (the latter which will be introduced later, in Section 2.1.5).

2.1.2 Spaces, Subspaces, and Linear Basis Functions

The nodal space. Let us return to the one-dimensional case. There, for each grid point $x_{l,i}$ we can attach a basis function $\varphi_{l,i} : [0,1] \to \mathbb{R}$. As the name suggests, the minimum condition we require (for now) from these functions is for them to be linearly independent for a given nodal level l. This means that any linear combination of these basis functions will be unique in the *nodal space* given by

$$\mathcal{V}_l = \operatorname{span}\left\{\varphi_{l,i} \mid 1 \le i < 2^l\right\}. \tag{2.9}$$

For any d-dimensional domain $\Omega = [0,1]^d$ we can extend all these definitions by means of tensor products, which will keep the important linear independence property. For each multi-dimensional grid point $\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}}$, with $\boldsymbol{l} = (l_1, \ldots, l_d), \boldsymbol{i} = (i_1, \ldots, i_d)$, the corresponding basis functions will thus be

$$\varphi_{\boldsymbol{l},\boldsymbol{i}} : [0,1]^d \to \mathbb{R}, \quad \varphi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) = \prod_{k=1}^{d} \varphi_{l_k,i_k}(x_k), \tag{2.10}$$

which define the corresponding nodal space

$$\mathcal{V}_{\boldsymbol{l}} = \operatorname{span}\left\{\varphi_{\boldsymbol{l},\boldsymbol{i}} \mid \mathbf{1} \le \boldsymbol{i} < \mathbf{2}^{\boldsymbol{l}}\right\}, \tag{2.11}$$

where by $\mathbf{1} \le \boldsymbol{i} < \mathbf{2}^{\boldsymbol{l}}$ we mean $1 \le i_k < 2^{l_k},\ k = 1, \ldots, d$.

If we consider that for the case $d = 1$ we treat one-dimensional vectors just as scalars, we can give a unified formulation for the solution of an interpolation problem would look like in $\mathcal{V}_{\boldsymbol{l}}$ for arbitrary values of d. For any real valued function

$r : [0,1]^d \to \mathbb{R}$ and given discretization level vector l, we can find then a unique interpolant

$$r(\boldsymbol{x}) \approx r_{\boldsymbol{l}}(\boldsymbol{x}) = \sum_{\boldsymbol{1} \leq \boldsymbol{i} < \boldsymbol{2}^{\boldsymbol{l}}} \alpha_{\boldsymbol{i}} \varphi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}) \tag{2.12}$$

by solving the system of equations

$$\sum_{\boldsymbol{1} \leq \boldsymbol{i} < \boldsymbol{2}^{\boldsymbol{l}}} \alpha_{\boldsymbol{i}} \varphi_{\boldsymbol{l},\boldsymbol{i}}(\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}}) = r(\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}}) \tag{2.13}$$

for coefficients $\alpha_{\boldsymbol{i}}$.

Finally, a useful notation that we will use moving forward is $\mathcal{V}_l^d$, denoting the isotropic nodal space of level l for given or arbitrary dimension d. So, in this notation, $\mathcal{V}_l^1 \equiv \mathcal{V}_l$, and $\mathcal{V}_l^d \equiv \mathcal{V}_{\boldsymbol{l}}$, with $l_k = l,\ k = 1, \ldots, d$, for any $d > 1$.

Hierarchical subspaces and hierarchical splitting. We have already introduce a split of the set of grid points along hierarchical levels $\boldsymbol{l}'$ with multi-index sets $I_{\boldsymbol{l}'}$. We can extend this to basis functions. Therefore, using the unified notation for an arbitrary d-dimensional case, we can introduce the corresponding *hierarchical subspaces* $\mathcal{W}_{\boldsymbol{l}'}$ defined by

$$\mathcal{W}_{\boldsymbol{l}'} = \operatorname{span}\left\{\varphi_{\boldsymbol{l}',\boldsymbol{i}'} \mid \boldsymbol{i}' \in I_{\boldsymbol{l}'}\right\}, \tag{2.14}$$

containing the corresponding *hiearchical basis functions* $\varphi_{\boldsymbol{l}',\boldsymbol{i}'}$.

It is straight-forward that $\mathcal{W}_{\boldsymbol{l}'} \subseteq \mathcal{V}_{\boldsymbol{l}}$. However, that is quite a weak property and we would like to have also the stronger *hierarchical splitting*

$$\mathcal{V}_{\boldsymbol{l}} = \bigoplus_{\boldsymbol{l}' \leq \boldsymbol{l}} \mathcal{W}_{\boldsymbol{l}'}. \tag{2.15}$$

This condition would guarantee that the nodal space can be decomposed into the direct sum of all coarser or same level hierarchical subspaces, allowing us thus to build successively higher level nodal spaces by simply adding the corresponding missing intermediate subspaces. This property is key to the construction of sparse grids, which will be introduced shortly.

For isotropic spaces, Eq. (2.15) becomes

$$\mathcal{V}_{\boldsymbol{l}} \equiv \mathcal{V}_l = \bigoplus_{\|\boldsymbol{l}'\|_\infty \leq l} \mathcal{W}_{\boldsymbol{l}'}, \tag{2.16}$$

where by $\|\boldsymbol{l}'\|_\infty = \max_{1 \leq k \leq d} l'_k$ we denote the infinity, or maximum, norm.

In general, the hierarchical splitting condition is not easily attainable and it is highly dependable on the type of basis functions chosen.

The piecewise linear basis. In order to more easily understand the construction of sparse grids and their interpolants, we will introduce now our first type of basis. In the following we will restrict ourselves to presenting only the 1-dimensional case. This is done for two reasons: the multi-dimensional case builds upon the univariate formulation by means of Eq. (2.10), and [83] proved that the general hierarchical splitting property can be reduced to proving its one-dimensional counterpart, which is a simpler task.

This simple basis is formed using the *piecewise linear* (or *hat*) *functions*

$$\varphi^1_{l,i}(x) = \max\left(1 - \left|x \cdot 2^l - i\right|, 0\right), \tag{2.17}$$

whose supports are given by

$$\operatorname{supp}(\varphi^1_{l,i}) = [x_{l,i-1}, x_{l,i+1}]\,. \tag{2.18}$$

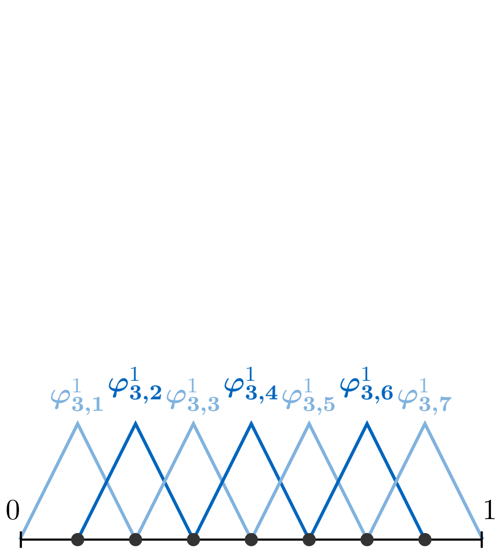
Here, the superscript "1" denotes the degree of the functions (in a polynomial sense), a notation which will link to the B-spline basis functions which will be introduced later in the chapter.

The piecewise linear are by far the most used basis functions, their attractiveness stemming from their simplicity, ease of evaluation, and narrow support. Being non-zero only in their corresponding intervals $\left(2^l(i-1), 2^l(i+1)\right)$, it is also straightforward to prove that they are indeed linearly independent, and moreover satisfy the hierarchical splitting property (Eq. (2.15)). The level 3 univariate hat functions are shown in Fig. 2.3.

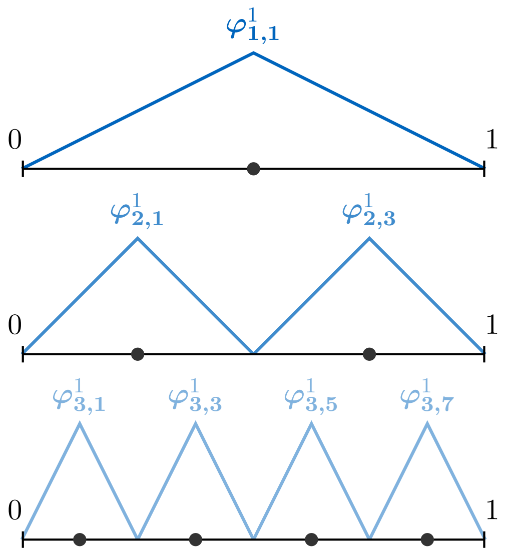
An important property of these functions is that the hierarchical splitting conditions applies to this basis even in the absence of any boundary treatment, a property that is not shared by general basis functions. Both the boundary issue and higher-degree bases will be treated later in the chapter.

2.1.3 Full Grids vs. Sparse Grids

Interpolation on the full grid. We have introduced the individual grid points, we have associated basis functions to them, and then grouped those functions into a nodal space that can be decomposed into hierarchical subspaces. With grid points and basis functions being in a on-to-one correspondence, it is only natural to also better define the sets and subsets of grid points in this nodal-hierarchical dichotomy. Therefore, the set $\left\{\boldsymbol{x}_{\boldsymbol{l},\boldsymbol{i}} \mid \boldsymbol{1} \leq \boldsymbol{i} < \boldsymbol{2}^{\boldsymbol{l}}\right\}$ of all grid points of level $\boldsymbol{l}$ form a *full grid*, while the subsets $\{\boldsymbol{x}_{\boldsymbol{l}',\boldsymbol{i}'} \mid \boldsymbol{i}' \in I_{\boldsymbol{l}'}\}$ of grid points corresponding to each subspace $\mathcal{W}_{\boldsymbol{l}'}$ form *subspace grids*. (We have imported here a useful terminology introduced in [69]. There full grids are called instead "component grids".) For full grids, in the case where the same discretization level l is being used in all dimensions, so for isotropic full grids, we can eliminate the vector notation and denote those grids by the scalar level value (e.g., 3-dimensional full grid of level $l = 2$, instead of level vector $\boldsymbol{l} = (2, 2, 2)$).



(a) Nodal hat functions



(b) Hierarchical hat functions in level-wise decomposition

Figure 2.3: The 1-dimensional nodal and hierarchical piecewise linear basis functions up to level 3. Both sets of functions span the same space. Due to their linear nature, these basis functions satisfy the hierarchical splitting condition on the whole unit interval even without boundary grid points.

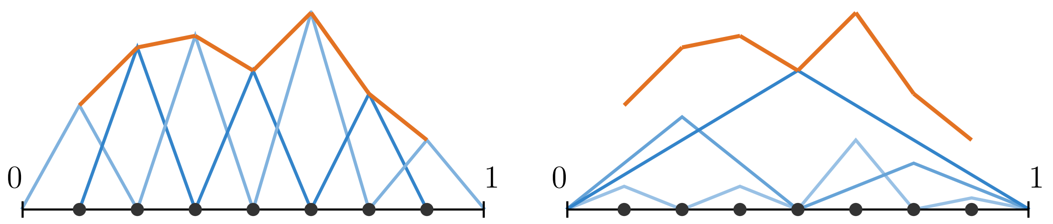
For bases that satisfy hierarchical splitting, it is clear that the solution of the interpolation problem in $\mathcal{V}_{\boldsymbol{l}}$ from Eq. (2.12) does not change whether we consider the nodal representation or the hierarchical one on a full grid. Therefore it holds:

$$r_{\boldsymbol{l}}(\boldsymbol{x}) = \sum_{\boldsymbol{l}' \leq \boldsymbol{l}} \sum_{\boldsymbol{i}' \in I_{\boldsymbol{l}'}} \alpha_{\boldsymbol{l}',\boldsymbol{i}'} \varphi_{\boldsymbol{l}',\boldsymbol{i}'}(\boldsymbol{x}). \tag{2.19}$$

In this formula, the coefficients $\alpha_{\boldsymbol{l}',\boldsymbol{i}'}$ are called the *(hierarchical) surpluses*. The process of finding these hierarchical coefficients that satisfy the interpolation conditions is called *hierarchization*, and, depending on the basis functions used, can have different efficient implementations [64, 83]. The topic of hierarchization algorithms will however not be covered further in this thesis. Nonetheless, a visualization of the difference between nodal and hierarchical representations is given in Fig. 2.4 on a simple one-dimensional interpolation scenario using the already introduced hat functions.

Always using the full grid for our numerical solutions quickly becomes unusable due to the curse of dimensionality. For the isotropic full grid of level l in d dimensions, the interpolation error with hat functions on the full grid of a smooth enough function (i.e., in order two mixed Sobolev space H^2_{mix}, so with bounded weak mixed derivatives up to order two) can be shown to be quadratic in the mesh size, i.e.,

$$\|r(\boldsymbol{x}) - r_{\boldsymbol{l}}(\boldsymbol{x})\|_{L_2} \in \mathcal{O}(2^{-2l}), \tag{2.20}$$



(a) 1D interpolation on the level 3 nodal hat functions **(b)** 1D interpolation on the level 3 hierarchical hat functions

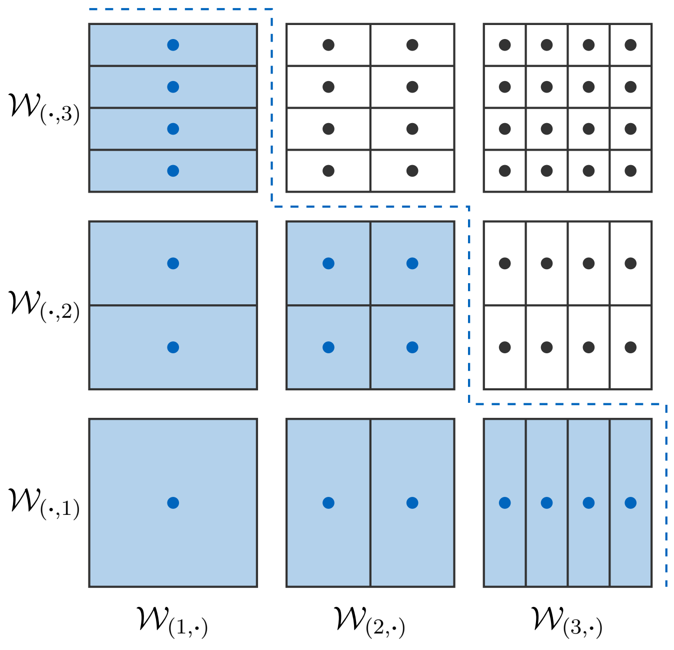
Figure 2.4: Example of a one-dimensional interpolation in both the nodal and hierarchical piecewise linear basis of level 3. In both representations the basis functions φ are shown scaled by their corresponding surpluses α (nodal and hierarchical, respectively). While the nodal surpluses are equal to the values of the interpolated function (in orange) at the respective grid points, the corresponding hierarchical surpluses require to be computed taking into account also the ancestral hierarchical surpluses (see, e.g., [83] for more details).

however at the cost of $\left(2^l - 1\right)^d \in \mathcal{O}(2^{ld})$ grid points [11, 28].

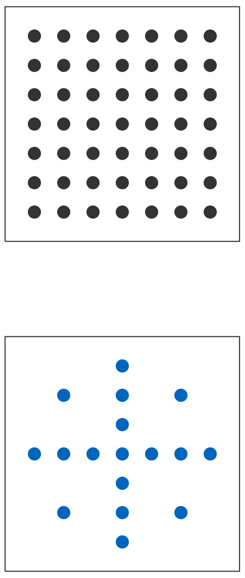
Building the sparse grid. This is where the hierarchical splitting again comes into play: by choosing subspaces with lower contributions to the total approximation error, we can eliminate a significant number of grid points with only a minimal impact on the error. The optimization problem of finding such a subset of subspaces is of course influenced by the norm we choose for the error. In our case, where we use the L_2 norm, and under the same assumption of the isotropic case of level l, the solution to such an optimization problem turns out to be the subset of hierarchical grids of levels $\|\boldsymbol{l}'\|_1 \leq l + d - 1$, where $\|\boldsymbol{l}'\|_1 = \sum_{k=1}^{d} l'_k$ is the 1-norm of vector $\boldsymbol{l}'$. We call such a grid the *regular sparse grid* of level l.

In comparison to the corresponding isotropic full grid of same level from before, on the sparse grid the error worsens only slightly to $\mathcal{O}(2^{-2l} \cdot l^{d-1})$ (in the same space of H^2_{mix}), while the size of the grid (i.e., the number of grid points) reduces to just $\sum_{j=0}^{l-1} 2^j \binom{d-1+j}{d-1} \in \mathcal{O}(2^l \cdot l^{d-1})$ [11, 28]. Fig. 2.5 shows the construction of a 2D sparse grid and the reduction in required grid points it provides.

The combination technique. There exists a second way of interpolating a function on the same set of grid points as the ones of the regular sparse grid of level l. Instead of direct computation, the *combination technique* proposes to use multiple solutions on coarser, anisotropic, full grids, building the sparse grid solution as their linear combination. This is done with three practical observations in mind. Firstly, as already stated, grid points on lower levels appear in higher ones as well, so combining their contributions in grids of different discretization levels requires no interpolations. Secondly, many numerical methods use full grids in order to solve a variety of problems, therefore there is a simple way of directly coupling those


$\mathcal{W}_{(\cdot,3)}$
$\mathcal{W}_{(\cdot,2)}$
$\mathcal{W}_{(\cdot,1)}$
$\mathcal{W}_{(1,\cdot)}$
$\mathcal{W}_{(2,\cdot)}$
$\mathcal{W}_{(3,\cdot)}$

(a) Hierarchical subspaces $\mathcal{W}_{\boldsymbol{l}'}$ of nodal space $\mathcal{V}_{\boldsymbol{l}}$ for $\boldsymbol{l} = (3,3)$. Combining all hierarchical subspaces produces a full grid. Highlighted subspaces (in blue) combine to form a regular sparse grid. Each subspace shows the grid points it contains, as well as their minimal supports (which coincide with the supports of the nodal basis functions at the grid points).



(b) The two-dimensional level 3 full grid (*top*) and corresponding regular sparse grid, respectively.

Figure 2.5: The two-dimensional level 3 subspaces and the construction of the corresponding level 3 regular sparse grid.

full grids into the combination technique. Lastly, as we combine (independent) solutions on multiple grids, this computation can easily be parallelized.

Now that we know what this approach proposes, we can give its mathematical formulation. The counterpart of the interpolation solution Eq. (2.12) in the combination technique for the isotropic case of level l is

$$r(\boldsymbol{x}) \approx r_{\boldsymbol{l}}(\boldsymbol{x})^{combi} = \sum_{k=0}^{d-1} \left[(-1)^k \binom{d-1}{k} \sum_{\|\hat{\boldsymbol{l}}\|_1 = l+(d-1)-k} r_{\hat{\boldsymbol{l}}}(\boldsymbol{x}) \right], \tag{2.21}$$

where by $r_{\hat{\boldsymbol{l}}}$ we denoted the interpolation solution on the full grid of level $\hat{\boldsymbol{l}}$. Fig. 2.6 exemplifies this combination formula for a 2D level 3 sparse grid.

Although it is clear the same grid points are being used, it can be proven, while not directly evident, that the combination technique solution and the regular sparse grid solution to the interpolation problem coincide (for the same grid level l) [64]. However, in learning tasks, as the ones treated in this thesis and which do not use

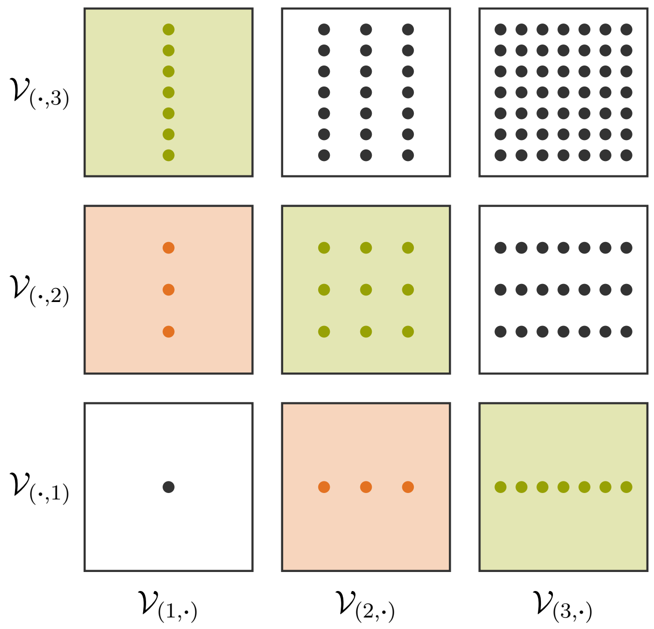

$\mathcal{V}_{(\cdot,3)}$
$\mathcal{V}_{(\cdot,2)}$
$\mathcal{V}_{(\cdot,1)}$
$\mathcal{V}_{(1,\cdot)}$
$\mathcal{V}_{(2,\cdot)}$
$\mathcal{V}_{(3,\cdot)}$

Figure 2.6: The two-dimensional level 3 combination technique. As opposed to Fig. 2.5a, here the components are not hierarchical subspaces but rather nodal ones (i.e., full grids). However, the same grid points are being used. With orange we have marked the components contributing with a negative sign in the formula of Eq. (2.21), and with green those contributing with a positive sign.

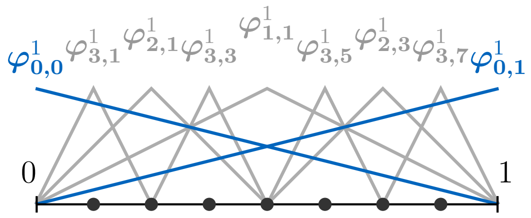
interpolation, the two approaches diverge slightly numerically, each having their own characteristics in different application scenarios [22].

Further aspects related to the combination technique, such as dimensionally adaptive sparse grids, will not be covered in this thesis. For our purposes, this introduction to the combination technique will be sufficient moving forward, but interested parties can find more in, e.g., [28, 23, 69, 83].

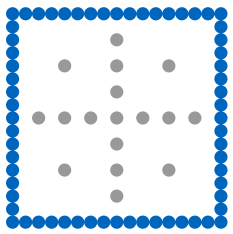
2.1.4 Boundary Grid Points, Modified Bases and Higher-Degree Bases

The boundary issue. We have postponed until now the discussion regarding the boundary treatment. That is because this topic is quite extensive on its own, linking also into the issue of hierarchical splitting and many options exist [64, 83]).

In order to approximate functions that are not zero on the boundary of the domain Ω the straight-forward approach is to add grid points that live on that boundary $\partial\Omega$. The simplest option in our univariate hierarchical representation would be to add these points in their own level $l' = 0$, i.e., grid points at indices $i = 0$ and $i = 2^l$ for a given discretization level l. In the generic d-dimensional case this translates to multiple subspace grids of levels $\boldsymbol{l}'$ for which $\exists k : l'_k = 0$.



(a) Univariate level 3 hierarchical piecewise linear basis with level 0 boundary functions $\varphi_{0,0}$ and $\varphi_{0,1}$.



(b) Example of a 2D regular sparse grid with level 0 boundary points (marked in blue).

Figure 2.7: A solution to the boundary issue on sparse grids using hierarchical bases. Adding level 0 functions is the simplest way to allow non-zero boundary values. This however leads to a drastic increase in total grid points. For example, in two dimensions for discretization level l we have $2^{2(l-1)}+1$ inner points and 2^{l+2} boundary points. For more details on how the size of grids changes with the increase in dimensions and discretization level, as well as other options, like adding boundary points on levels other than 0 (see, e.g., [83]).

As can be seen from Fig. 2.7, this method unfortunately has the unwanted side-effect that it increases the number of total grid points significantly. An option to reduce this effect is to add these boundary points in one or more higher levels, which reduces the amount of boundary points, at the cost of loosing efficiency in the computation of hierarchical surpluses. For brevity, we will not address those variants in this thesis, but for more details see, e.g., [83].

Modified hat functions. Another option to avoid the boundary issue was introduced by Pflüger [64], who proposes instead of augmenting the grid with boundary points to modify the basis functions by means of non-zero extensions towards the boundary $\partial\Omega$.

As the multivariate case will again be constructed via the usual tensor product, we are only interested in the univariate case. The idea of Pflüger for applying this process to hat functions was to have in each level l the inner indices $i=1$ and $i=2^l-1$ linearly extrapolate outwards. For all other indices, as their influence is zero on the boundary, the basis remains unchanged. This results in a slightly more complex definition for these *modified hat functions*, but which can still be implemented and evaluated in practice quite efficiently:

$$\varphi^{1,mod}_{l,i}(x) = \begin{cases} 1, & l=1, \quad i=1, \\ \max\left(2 - x\cdot 2^l\right), & l\geq 2, \quad i=1, \\ \varphi^{1,mod}_{l,i}(1-x), & l\geq 2, \quad i=2^l-1, \\ \varphi^1_{l,i}(x), & \text{otherwise.} \end{cases} \tag{2.22}$$

The basis is exemplified in Fig. 2.8.

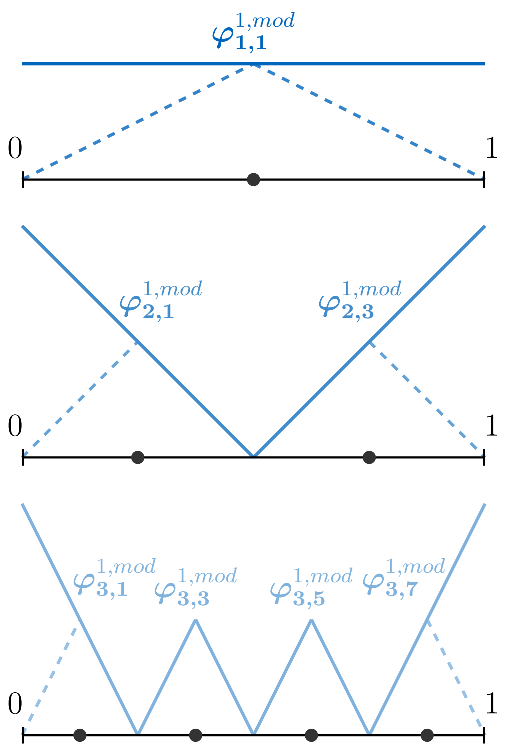


Figure 2.8: The level 3 modified hat functions. We denote by dashed curves the portions of the original (unmodified) basis functions which do not overlap with the modified ones.

Two remarks need to be made. Firstly, the modified hat functions provide only an educated guess regarding the values of the true values of the interpolated function at the boundary, under assumptions that either the function does not change too much close to the boundary, or that the accuracy close to the boundary is of no concern. Secondly, in many applications, as we will see in this thesis, we can transform our datasets such that they are fully inside the domain $(0,1)^d$ and have no boundary influence, thus in those scenarios we can simply ignore the boundaries altogether and use the regular hat functions that go to zero at the boundary by default.

Higher-degree bases – not-a-knot B-splines. While hat functions are often enough to obtain a continuous grid interpolant, sometimes we require additional properties (e.g., continuous differentiation) from our numerical representation. *B-splines* offer a good compromise between additional computational effort for basis evaluation, minimal function support, and sufficient degree of smoothness.

Valentin [83] worked extensively on finding viable sparse grid bases using B-splines, which is not a simple task due to the fact that the hierarchical splitting condition and the hierarchization process are non-trivial for any kind of generic (i.e., non-linear) basis functions. In the following we will restrict ourselves to presenting the basics of construction not-a-knot B-splines and their modified counterparts, with proofs and intermediary results left for interested parties to be found in the cited reference.

The building block to our basis function is the *non-uniform B-spline.* As usual, we restrict ourselves to presenting the 1-dimensional case, as the d-dimensional case can be obtained via the already known tensor product approach. For a given degree $n \in \mathbb{N}_0$ and number of splines $m \in \mathbb{N}_0$, let $\boldsymbol{\xi} = (\xi_0, \dots, \xi_{m+n})$ be an ordered increasing set of real numbers called the *knot sequence.* Then, for $k = 0, \dots, m-1$, we can define the non-uniform B-splines by the recurrence

$$b^n_{k,\boldsymbol{\xi}}(x) = \begin{cases} \dfrac{x-\xi_k}{\xi_{k+n}-\xi_k} b^{n-1}_{k,\boldsymbol{\xi}}(x) + \dfrac{\xi_{k+n+1}-x}{\xi_{k+n+1}-\xi_{k+1}} b^{n-1}_{k+1,\boldsymbol{\xi}}(x), & n \geq 1, \\ \chi_{[\xi_k,\xi_{k+1})}(x), & n = 0, \end{cases} \tag{2.23}$$

these m splines forming a basis (which denotes the “B” in the name “B-splines”).

The *not-a-knot B-splines* are a type of non-uniform spline where the knot sequence is a carefully chosen subset of the set of grid points for a given discretization level l. This restricts us to using only odd degree B-splines (i.e., $n = 1, 3, 5, \dots$), however, when taking into account also boundary grid points, guarantees the hierarchical splitting condition on the whole unit domain. This *not-a-knot sequence*, for level l and odd degree n is given by

$$\begin{aligned} \boldsymbol{\xi}^{n,nak}_l &= \left(\xi^{n,nak}_{l,0}, \dots, \xi^{n,nak}_{l,m+n}\right), \ m = 2^l + 1, \\ \xi^{n,nak}_{l,k} &= \begin{cases} x_{l,k-n}, & k = 0, \dots, n, \\ x_{l,k-(n+1)/2}, & k = n+1, \dots, 2^l, \\ x_{l,k-1}, & k = 2^l+1. \dots, 2^l+n+1. \end{cases} \end{aligned} \tag{2.24}$$

Using this knot sequence we can define the *not-a-knot B-spline basis* as

$$\varphi^{n,nak}_{l,i} = \begin{cases} L_{l,i}, & l < \lceil \log_2(n+1) \rceil, \\ b^n_{i,\boldsymbol{\xi}^{n,nak}_l}, & l \geq \lceil \log_2(n+1) \rceil, \end{cases} \quad i = 0, \dots, 2^l, \tag{2.25}$$

where

$$L_{l,i} : [0,1] \to \mathbb{R}, \ L_{l,i}(x) = \prod_{\substack{i' = 0, \dots, 2^l \\ i' \neq i}} \frac{x - x_{l,i'}}{x_{l,i} - x_{l,i'}} \tag{2.26}$$

are Lagrange polynomials.

Two remarks have to be made regarding this basis. Firstly, one need to note that the basis is defined for grids that contain boundary points, as this is a necessary condition for the hierarchical splitting to hold, as was proven by Valentin [83]. Secondly, for degree $n = 1$ not only does the not-a-knot sequence coincide with the full set of grid points, but the not-a-knot B-splines coincide with the hat functions,

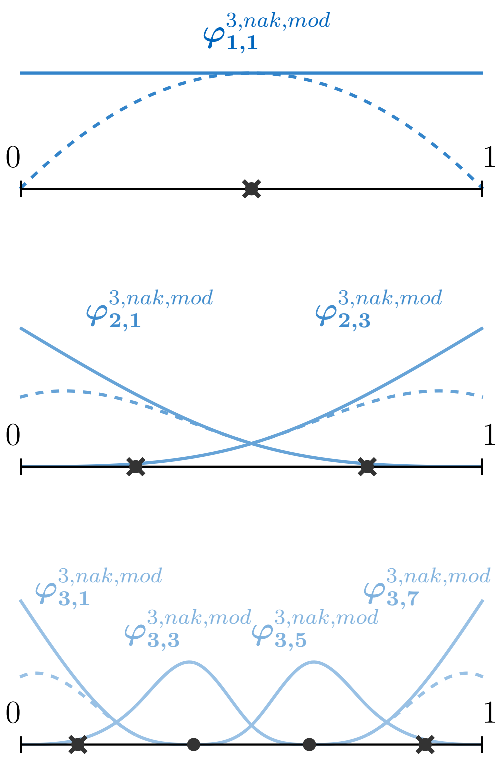


Figure 2.9: The level 3 modified not-a-knot B-spline functions of degree 3. We denote by dashed curves the portions of the original (unmodified) basis functions which do not overlap with the modified ones.

i.e., $\varphi_{l,i}^{1} \equiv \varphi_{l,i}^{1,nak}$, which justifies the superscript notation we introduced in that section.

For practical reasons, and for the purpose of our thesis, we will not use this basis due to the increasing number of boundary grid points it entails. We will instead use the *modified not-a-knot B-splines*, which are constructed by a similar procedure as modified hat functions and are defined for a given level l and for inner indices $i = 1, \ldots, 2^l - 1$ by

$$\varphi_{l,i}^{n,nak,mod}(x) = \begin{cases} 1, & l = 1, \quad i = 1, \\ \varphi_{l,1}^{n,nak}(x) - \dfrac{\frac{\mathrm{d}^2}{\mathrm{d}x^2}\varphi_{l,1}^{n,nak}(0)}{\frac{\mathrm{d}^2}{\mathrm{d}x^2}\varphi_{l,0}^{n,nak}(0)}\varphi_{l,0}^{n,nak}(x), & l \geq 2, \quad i = 1, \\ \varphi_{l,1}^{n,nak,mod}(1-x), & l \geq 2, \quad i = 2^l - 1, \\ \varphi_{l,i}^{n,nak}(x), & \text{otherwise.} \end{cases} \tag{2.27}$$

The basis is exemplified in Fig. 2.9.

2.1.5 Spatial Adaptivity

The issue with using only regular sparse grids to approximate all kind of functions is that we can only increase accuracy by increasing the level, which adds new grid points in specific places in the domain, however in a spread-out fashion. Real world functions do not exhibit in general smoothness in the whole domain, therefore ideally we would like to invest grid points only in areas where these smoothness conditions are not met, or where the target function is suspected of having interesting characteristics that we want to be reflected in our grid-based approximation. This is the main idea behind spatial adaptivity: the (re)distribution of grid points in places of interest inside the computational domain in order to reduce some approximation error for the problem we are trying to solve. For this purpose, spatial adaptivity requires the use of the hierarchical representation of grid points.

This is however not the only method of refinement used with sparse grids. The combination technique naturally lends itself to a dimension-wise adaptivity strategy instead, where we add or remove grid points at the subspace level, instead of the more localized approach of spatial adaptivity. This comes with both pros and cons, as on the one hand it can more easily increase or decrease the resolution in certain dimensions of interest, but on the other hand it can lead to a more steep increase in used grid points, especially when the phenomena we want to capture with the sparse grid in a certain direction does not span the whole domain, or when the directions of interest are not axis-aligned. Some of these drawbacks were addressed by Obersteiner [55] who implemented a spatially adaptive combination technique using dimension-wise refinement, with significant improvement to the regular adaptivity strategy for interpolation and quadrature tasks.

We could say that selecting a certain level sparse grid is already a sort of *a priori* adaptivity step. There have also been ideas put forward that propose the use of pre-refined (or pre-coarsened, depending how they are constructed) sparse grids for certain applications (e.g., [62]). However, in general and also in this thesis, when discussing sparse grid adaptivity we refer to the *a posteriori* approach, where we use error estimates of the current grid approximation to add and/or remove grid points in order to improve the numerical solution of our problem. In the following we will explore in more detail the process of spatial adaptivity applied to hierarchical sparse grids.

Grid refinement. The main procedure mentioned the most when referring to grid adaptivity is *refinement*, i.e., the process of adding new grid points. For spatially adaptive sparse grids we need to take into account the nature of the underlying data structure. It is clear that only leaf grid points are able to be refined, as they still have hierarchical children to be added. While it is possible to add only specific children (as, e.g., the refinement strategy for classification in [69]), for our purposes we focus on the original, and most widespread, approach of always adding all $2d$ hierarchical children of any refined grid point. It also has to be noted that in the multivariate case it is not guaranteed when adding a new grid point that all its

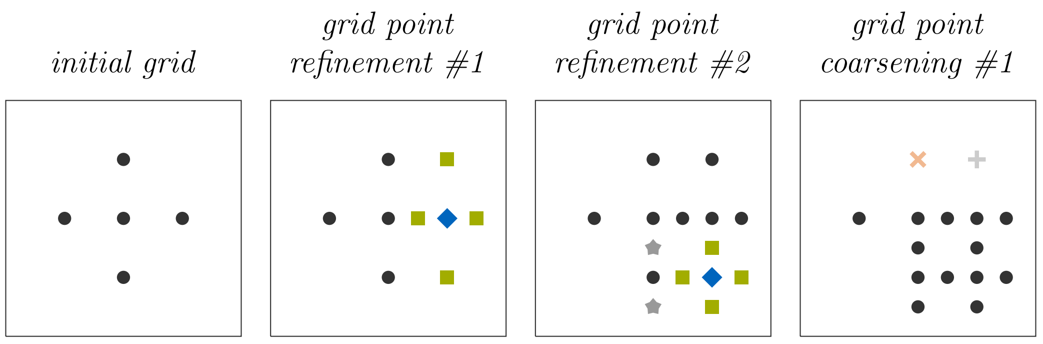


Figure 2.10: Examples of a spatial adaptivity step on a two-dimensional sparse grid. Here, we perform two grid point refinements followed by one grid point coarsening, starting from a level 2 regular sparse grid. Refining a point (◆) adds to the grid all its children (■), and those children's hierarchical parents that are missing (⬟). Coarsening a point (✕) removes it from the grid, together with all its children (+).

hierarchical ancestors actually exist, therefore any such missing ancestor has to also be added to the grid in order to fulfill the requirements of algorithms that require efficient traversals of the grid hierarchy[64].

The spatial *refinement step* we use is as follows: based on a so-called *refinement criterion*, we score ***leaf*** grid points and rank them to reflect their importance or need for refinement. We decide on how many of them to consider, and only choose that number of top ranked points. For each of them we add all missing hierarchical children, for each child making sure all of its ancestors exist in the grid.

The grid points allowed to be refined are called *refinable*. This can be changed on a problem-specific basis, however in this thesis we consider the maximal refinable set, i.e., the set of all leaf points.

Grid coarsening. Especially when working with a limited number (or budget) of grid points, a second type of adaptive step can be required: coarsening, i.e., the process of removing existing points from the sparse grid. Grid coarsening is comparatively more seldom used in the context of sparse grids, even though it does improve the ability to tailor the computational grid to the underlying data or target function we work with.

The spatial *coarsening step* goes as follows: based on a so-called *coarsening criterion*, we score ***all*** grid points and rank them to reflect their need for removal. We decide on how many of them to consider, and only choose that number of top ranked points. For each of them we delete it along with all its hierarchical children.

The grid points allowed to be coarsened are called *coarsenable*. Some applications could possibly add an additional restriction on which points can be coarsened. In this thesis however we only consider the maximal coarsenable set, i.e., the set of all grid points.

The adaptive step. One step of grid adaptivity can contain a refinement step, or both a coarsening and a refinement step (see Fig. 2.10 for an example). Performing only coarsening is never done, to our knowledge. Mixing the types of adaptive steps on the course of a sparse grid run (which normally might encompass multiple adaptivity steps) is also never performed as far as we know, meaning that one has to decide whether to include coarsening at the start of a sparse grid-based simulation.

Whether it is better in general to perform a grid coarsening before or after the corresponding refinement step is an open question. By coarsening after refining, on the one hand we can trim out unwanted children from the maximum of $2d$ introduced by each refined point. On the other hand, as we apply the point-wise operations multiple times in each refinement/coarsening step, poorly chosen adaptivity criteria can lead to the removal of too many new points, some as soon as they are introduced by the refinement, leading to a waste of computational effort. Either way, we recommend that in general the number of coarsened points to be kept well below that of refined ones, and that coarsening be performed before the corresponding refinement in each adaptivity step.

Adaptivity criteria. A multitude of adaptivity criteria for sparse grids have been proposed. While it is in principle possible to use a different criterion for coarsening than for refinement, that is not done in practice as far as we know. As scores are supposed to measure some error contribution, a practical option to reduce some of the computational effort, which we also employ, is to use a single ranking approach, where we score all grid points once per adaptivity step and choose from top ranks points for refinement and from bottom ranks points for coarsening (or vice versa, depending on whether the scores are meant to reflect a direct or inverse relation to the error). Of course, in both cases we restrict our choices of ranked points to those which are refinable and coarsenable, respectively. Throughout this thesis we will use exclusively this single scoring method, with high ranks always marking grid points needed to be refined.

The first criterion we will use in this thesis is also the simplest. This *surplus-based* adaptivity scorer ranks grid points by the absolute value of their corresponding hierarchical surplus:

$$\text{score}^s(x_{\boldsymbol{l},\boldsymbol{i}}) = |\alpha_{\boldsymbol{l}',\boldsymbol{i}'}|. \tag{2.28}$$

It has shown that refining using this criterion reduces the overall L_2 interpolation error [11], making a great default choice for a variety of applications [64, 34, 58].

The simple surplus-based criterion was noted to be prone to overfit when used for some learning scenarios [64, 69]. Therefore the *surplus-volume* criterion tries to alleviate this issue by including the support of each grid point in the score calculation:

$$\text{score}^{sVol}(x_{\boldsymbol{l},\boldsymbol{i}}) = |\alpha_{\boldsymbol{l}',\boldsymbol{i}'}| \cdot |\text{supp}(\varphi_{\boldsymbol{l}',\boldsymbol{i}'})|, \tag{2.29}$$

where $\mathrm{supp}(\varphi_{\boldsymbol{l}',\boldsymbol{i}'}) = \prod_{k=1}^{d} \mathrm{supp}(\varphi_{l'_k,i'_k})$ is the multivariate support obtained via the Cartesian product of the component univariate supports.

The last criterion we will discuss is the one introduced by Peherstorfer specifically for the problem of density estimation using sparse grids [58]. The *surplus-value* score is more data-driven, being a combination of the absolute magnitude of the corresponding hierarchical surpluses and the value of the approximation at the respective grid point:

$$\mathrm{score}^{sVal}(x_{\boldsymbol{l},\boldsymbol{i}}) = |\alpha_{\boldsymbol{l}',\boldsymbol{i}'}| \cdot r_{\boldsymbol{l}}(x_{\boldsymbol{l},\boldsymbol{i}}). \tag{2.30}$$

For the problem of more complex density estimators, which will be covered in Chapter 3, we adjust this criterion to take into account the fact that our target functions are not expected to be probability density functions, therefore negative values can occur. Therefore, in those scenarios, we will work with a *surplus-absolute-value* score

$$\mathrm{score}^{sAbsVal}(x_{\boldsymbol{l},\boldsymbol{i}}) = |\alpha_{\boldsymbol{l}',\boldsymbol{i}'}| \cdot |r_{\boldsymbol{l}}(x_{\boldsymbol{l},\boldsymbol{i}})|. \tag{2.31}$$

2.1.6 Additional remarks

Moving forward, we will consider more often than not the case of a generic spatially adaptive sparse grid, defined only by the number of grid points it contains (or the size of the grid) N and, possibly, the number of dimensions (or dimensionality) d. In this case, the hierarchical basis functions span a now level-independent space $\mathcal{V}_N$ and any function $r \in \mathcal{V}_N$ living on the grid can to be written as

$$r(\boldsymbol{x}) = \sum_{k=1}^{N} \alpha_k \varphi_k(\boldsymbol{x}), \tag{2.32}$$

where index k follows a non-specific traversal of all grid points, and α_k are the hierarchical surpluses corresponding to the hierarchical basis functions φ_k.

2.2 Learning with Sparse Grids - Current State

Having introduced spatially adaptive sparse grids, we can now introduce some of the application areas in which we have brought contributions with our thesis. These refer to problems in the field of machine learning, where some of the more consistent increase in contributions from the sparse grid community took place over the past couple decades. In this section we will introduce the main existing approaches that our work is based on, allowing us later to do a better-informed and more objective assessment of the place our thesis takes in the larger context. We will focus here primarily on the relevant notations and the more theoretical aspects upon which we will build in the following chapters.

As mentioned already in the introductory Chapter 1, our work in this thesis brings contributions which have come together by addressing them each with a specific focus. Therefore, in the following we will present the learning tasks solved using sparse grids also each with a spotlight on one of the three aspects we want to highlight: for the density estimation task there will be a more algorithmic focus, for the regression and classification task we will delve specifically more on the high-performance computing aspects, and for the clustering task we will focus on the usability aspects and the interaction of software implementations and their users.

It has to be stated that, while the methods we will be covering, not just in this section but throughout the thesis, will be presented as viewed primarily from one of three perspectives, usually these three facets are inexorably linked, and an interplay of them cannot be understated or dismissed. It is clear that any good new algorithm has to keep also a performance focus of its subsequent implementation as early on in its development as possible. Conversely, as certain algorithms reach their limits in terms of the possible performance they can deliver, innovative new approaches come naturally as a response in order to allow for a faster time-to-solution. On the software side of things, pieces of code seldom find satisfying or complete usefulness independent of larger contexts (i.e., numerical libraries). Especially when wanting to increase the reach and applicability of algorithms, maximizing the usability aspects of these pieces of software has to also be taken into account.

Having established the overarching frame of view in which we will address these sparse grid-based methods, let us then begin our rundown of these building blocks which form the foundation of our contributions in this thesis.

2.2.1 Density Estimation using Sparse Grids

Problem statement. The problem of density estimation can be stated generically as:

> Given a set of independent and identically distributed samples in $\Omega \subseteq \mathbb{R}^d$, $\mathcal{S}_p = \{\boldsymbol{x}_i^p\}_{i=1}^{M_p}$ with density $p(\boldsymbol{x})$, estimate the value of $p(\boldsymbol{x})$.

Short history of the methods. While not the earliest machine learning tasks to receive a sparse grid treatment, the problem of *sparse grid density estimation* (SGDE) is one that is particularly important among sparse grid algorithms not only due to the fact that its solution has nice theoretical and numerical properties in of itself, but also because it provided the means to both reach previously unattainable machine learning tasks (i.e., clustering [61]) and improve on already existing solutions (i.e., classification [59]).

The algorithm introduced by Peherstorfer [60] to solve the density estimation task is based on the derivations of Hegland et al. [33], who used a spline smoothing approach to find the desired estimate as the solution of a variational equation. Peherstorfer restricted then the search space to that of sparse grid-based functions. For the purposes of this thesis, we will reframe the mathematical derivation of the algorithm more into the key of a straight-forward squared loss optimization problem, similar to a constrained *kernel density estimation* (KDE) (introduced in its modern form by Rosenblatt [67] and Parzen [57]) with a *mean integrated squared error* (MISE) optimality criterion. This will more closely resemble thus the derivations we will obtain for our new density estimators in Chapter 3.

In the following we will introduce mathematically the algorithm of Peherstorfer, present some of the numerical properties of the method, and then shortly introduce the existing implementation that we will use as a template for our own new density estimators to be introduced in Chapter 3.

The sparse grid algorithm. We begin the mathematical derivation of SGDE by searching for a function $r(\boldsymbol{x})$ in a suitable function space V that minimizes the regularized squared loss given by

$$J(r) = \int_\Omega [r(\boldsymbol{x}) - p(\boldsymbol{x})]^2 \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda r\|_{L^2}^2. \tag{2.33}$$

The first term of the loss is meant to guarantee that our target function is close to the exact value. The second term $\|\Lambda r\|_{L^2}^2$ imposes a smoothness constraint, similar to those found in kernel-based methods, in order to avoid possible overfitting problems; the operator Λ is also not imposed. The relationship between these two effects (accuracy and smoothness) is controlled via the regularization parameter λ.

In order to find our target $r(\boldsymbol{x})$ we employ techniques of calculus of variations, such that our minimization problem is replaced by the corresponding weak form

$$\int_\Omega s(\boldsymbol{x})\, r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int_\Omega s(\boldsymbol{x})\, p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0, \, \forall s \in V. \tag{2.34}$$

The exact density $p(\boldsymbol{x})$ is, of course, unknown, therefore we need to use some sample-based approximation. While one may consider more complex options, in practice the usual empirical estimator

$$p(x) \approx \frac{1}{M_p} \sum_{i=1}^{M_p} \delta_{\boldsymbol{x}_i^p} \tag{2.35}$$

is enough, where $\delta_{\boldsymbol{x}_i^p}$ is the Dirac delta function centered at $\boldsymbol{x}_i^p$. This approximation via an empirical estimator can also be seen as an initial overfitted guess at the target density from a spline smoothing perspective, especially if we look at the origin of the method that Peherstorfer applied to sparse grids. Introducing Eq. (2.35) in Eq. (2.34), we can move the r-independent term on the right-hand side and obtain

$$\int_\Omega s(\boldsymbol{x})\, r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \frac{1}{M_p} \sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p), \, \forall s \in V. \tag{2.36}$$

Until now we have worked in an arbitrary domain Ω and with functions of an arbitrary space V. At this point we decide to restrict our solution to the domain of sparse grids, therefore we impose $\Omega = [0,1]^d$ and $V = \mathcal{V}_N$. Moreover, we use Galerkin projection in order to transform our variational problem into a linear system of equations, meaning we consider

$$r(\boldsymbol{x}) := \sum_{k=1}^{N} \alpha_k \varphi_k(\boldsymbol{x}), \quad s(\boldsymbol{x}) := \varphi_l(\boldsymbol{x}),\, l = 1, \ldots, N, \tag{2.37}$$

which results in

$$\begin{aligned}\sum_{k=1}^{N} \alpha_k \int_\Omega \varphi_k(\boldsymbol{x})\, \varphi_l(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} + \lambda \sum_{k=1}^{N} \alpha_k \int_\Omega \Lambda\varphi_k(\boldsymbol{x})\, \Lambda\varphi_l(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \\ = \frac{1}{M_p} \sum_{i=1}^{M_p} \varphi_l(\boldsymbol{x}_i^p),\ l = 1, \ldots, N.\end{aligned} \tag{2.38}$$

It is usual to write this equation in its equivalent matrix-vector form

$$(\boldsymbol{R} + \lambda \boldsymbol{C})\, \boldsymbol{\alpha} = \boldsymbol{b}^p, \tag{2.39}$$

where $R_{k,l} = \langle \varphi_k, \varphi_l \rangle_{L^2}$, $C_{k,l} = \langle \Lambda\varphi_k, \Lambda\varphi_l \rangle_{L^2}$, $\boldsymbol{b}^p = \frac{1}{M_p} \boldsymbol{B}^p \boldsymbol{e}^p$, with $\boldsymbol{B}^p_{l,i} = \varphi_l(\boldsymbol{x}_i^p)$ and $\boldsymbol{e}^p$ is the all-ones vector of length M_p.

This representation of the right-hand side as a multiplication of a matrix $\boldsymbol{B}$ of evaluations of all grid basis functions at all input data points and of a vector of ones $\boldsymbol{e}$ is used to more closely depict the way the matrix-vector operations are actually implemented in the code of the SG++ library.

The choice of regularization. The choices of different regularization operators Λ has been quite extensively studied [64]. As pointed out also in [58] and [83], the hierarchical surpluses α_k themselves are linked to the second derivative of the target function interpolated on the sparse grid, so for many applications a simple yet useful choice in practice is

$$\|\Lambda r\|_{L^2}^2 = \sum_{k=1}^{N} |\alpha_k|^2, \tag{2.40}$$

which results in using the identity matrix $C = I$ in Eq. (2.39). This avoids the dimensionality-dependent computations of using for example the Laplacian $\Lambda = \nabla$, while obtaining similar regularization properties at virtually no added computational cost. This behavior was initially observed in classification and regression by Pflüger [64], but Peherstorfer then saw similar benefits when applied to density estimation [60]. Therefore, for the purposes of our thesis, we will use only the simple identity regularization as well.

Properties of the linear system. The main reason this sparse grid-based density estimator is a very powerful approach is because it breaks the dependency of the resulting linear system to be solved on the number of input data samples, which is usually a significant drawback in the more well-known kernel-based approaches. Indeed, as can be seen from Eq. (2.39), the system matrix R grows only with the number of grid points. While for full grids this might not even be an advantage at all, when employing sparse grids normally the number of grid points can be even orders of magnitude less than the number of input data points. The only place the input dataset is used in this formulation is in the right-hand side term, but also there, with the reduced number of basis functions to iterate through, one expects fewer computations to be done overall than an equivalent kernel-based approach. This is of course just strictly in terms of maximum computations to be performed, excluding all the possibilities of speed-ups due to efficient implementations and parallelization opportunities that both approaches can provide.

As the system matrix does not change with extra input datasets, one can precompute these matrices and store them efficiently using algebraic method, namely matrix decomposition techniques. Peherstorfer observed already the potential of this approach by proposing the use of lower-upper (LU) and eigenvalue decompositions of R in what is called an offline/online splitting: a more expensive offline step to learn the decomposition for a given grid, then repeated online applications of the decomposed matrix to learn from consecutive batches of data, in a similar approach to classical techniques in stream learning.

The data mining pipeline of SG++. To properly conclude this rundown on the existing work related to the SGDE method we could not omit the implementation aspects. The piece of software of interest to us in the context of density-based learning is the data mining pipeline of Röhner[69], who created it as a submodule of the larger SG++ sparse grid library (originally by Pflüger [64]) and upon which we will build our own implementation of the new density estimators in Chapter 3.

In this code, Röhner extended significantly the offline/online splitting introduced by Peherstorfer. Firstly, he expanded the choices of possible decomposition, such that now the default recommended options are in fact some of the more recent additions, namely the orthogonal and Cholesky decompositions. Secondly, the pipeline supports spatial adaptivity in most system matrix decompositions, such that changes in the grid (i.e., refinement and coarsening steps) as well as in the regularization parameter λ will result in efficient updates of the existing decomposition of R, not in a full recalculation of the system matrix. Added performance was introduced by Röhner by offering ScaLAPACK[1]-based variants to all these decompositions, which parallelizes the matrix-vector operations done when solving the linear system in Eq. (2.39).

One additional important aspect of the data mining pipeline is its batch learning process [69]. What this entails is the processing of subsets of the input dataset

[1]`http://performance.netlib.org/scalapack/`

(called *batches*) and the incremental build of the final solution to the density estimation problem from the results on each successive batch. This allows not only an affordable way to deal with very large datasets for which the effort can now be split and even parallelized, but also allows for the implementation of a concept drift approach, where batches have a time component, thus older data can be discounted at a different rate than newer data. In the case of SGDE, one important remark has to be made: splitting the computation on different batches does allow us a simple way to recover the exact solution we would obtain when processing the full dataset. This is due to the fact that the whole left-hand side of the linear system in Eq. (2.39) is independent of the input dataset. In the case of our new estimators in Chapter 3, we will see that the recovery of the exact full dataset solution from batches can sometimes be more problematic, as it will be addressed in the implementation details of each of those methods.

While some more details regarding the structure and other functionalities of the data mining pipeline will be given in Chapter 5, we cannot understate the benefits and importance of working with a well-crafted piece of software, that has what we will call later in the thesis a high degree of usability. This aspect is extremely relevant, as our own density estimators which we will introduce in Chapter 3 were integrated into this existing framework, and thus could benefit directly from a lot of preexisting functionalities. For example, the opportunities for parallelizing the batch processing and ScaLAPACK-based speed-ups of matrix-vector operations (where applicable), as well as hyper-parameter optimization and visualization of results, come with very little added cost due to the modular fashion in which the pipeline is constructed. The former two functionalities in particular will be expanded upon for each new density estimator, as their implementation involved more effort and was more algorithm dependent. The latter two functionalities are mostly independent of the algorithms, so they will not be detailed further (although more information for interested parties can be found in [69]). We thus chose deliberately to only focus in this thesis on discussing those functionalities of the pipeline that required more significant work on our part in the process of integrating our new algorithms and which differ thus from those already existing which apply also to the regular density estimation algorithm.

2.2.2 Classification and Regression with Sparse Grids

Problem statements. The problem of regression for machine learning can be formulated as:

> Given a set of training data points $\mathcal{S} = \{\boldsymbol{x}_i\}_{i=1}^M$ in $\Omega \in \mathbb{R}^d$ and a corresponding set of scalar targets $\{y_i\}_{i=1}^M \in \mathbb{R}$, best approximate the function f that satisfies $f(\boldsymbol{x}_i) = y_i$, $\forall i = 1, \ldots, M$.

With this formulation, the task of classification can be viewed to a great extent as nothing more than a particular case of regression, where the targets are restricted to non-negative integers representing the possible class labels: $\{y_i\}_{i=1}^M \in$

$\{1, 2, \ldots, k\}$, $k \in \mathbb{N}_0$. Especially in the context of the standard sparse grid approach to solving the two tasks (which we will introduce shortly), there is little to no difference in terms of solving these tasks. This is why in various sections of this thesis we might refer, when needed, to a combined "regression/classification" task, instead of specifically one or the other of the problems.

Short history of the methods. We can trace back some of the earliest algorithms dealing with sparse grids in learning tasks to the works of Garcke and Griebel. Their initial directions of study were on dimension-adaptive solutions to the regression/classification scenarios [28]. Further work followed suit, with focus on different basis functions [26] and an extension of these learning scenarios to the semi-supervised setting using the intrinsic geometric characteristics of the input datasets [27], results that Garcke coalesced in his PhD thesis [40]. He followed this with a better combination technique approach for solving the regression problem on sparse grids [22]. It has to be noted that the majority of these early solutions were designed with dimensionally adaptive sparse grids in mind, i.e., using the combination technique. Since then other interesting additions to this approach have been made, for example an integrated preprocessing step of input data transformation to better fit the intrinsic axis-aligned grid point distribution of the sparse grids [10]. Solutions to the regression/classification problem have seen a big boost with the introduction of the spatially adaptive sparse grids, Pflüger proving that this approach can deliver high accuracies in these learning scenarios [64]. Algorithmically, this sparse grid approach to regression/classification has not changed significantly since, although its applications have been extended. Relevant to our work, Garcke et al. [9, 25] combined the delay embedding approach with sparse grid-based regression to solve the problem of time series prediction successfully, the details of which will be expanded upon in Section 4.2.

In the following we will give a short theoretical introduction to the method, which is necessary to the understanding of the optimization efforts which will be described in Chapter 4.

The sparse grid algorithm. In order to solve the regression/classification problem, we start by searching for a function in a suitable space $f \in V$ that minimizes the regularized squared error

$$J(f) = \frac{1}{M} \sum_{i=1}^{M} \left[y_i - f(\boldsymbol{x}_i)\right]^2 + \lambda \|\Lambda f\|_{L^2}^2, \tag{2.41}$$

where the first term contains the cumulated error in approximation at the data points and the second term combats the tendency for overfitting by imposing a smoothness constraint.

Restricting the search for a solution to the sparse grid space $\mathcal{V} = \mathcal{V}_N$, i.e., $f(\boldsymbol{x}) := \sum_{j=1}^{N} \alpha_j \varphi_j(\boldsymbol{x})$, we obtain:

$$J(f_N) = \frac{1}{M}\sum_{i=1}^{M}\left[y_i - \sum_{j=1}^{N}\alpha_j\varphi_j(\boldsymbol{x}_i)\right]^2 + \lambda\|\Lambda f\|_{L^2}^2. \tag{2.42}$$

We require then for this error to be minimized with respect to all α_k, $k = 1, \ldots, N$, i.e., we impose $\frac{\partial J(f_N)}{\partial \alpha_k} = 0$, resulting, after maneuvering the terms, in the linear system:

$$\begin{aligned}\sum_{j=1}^{N}\alpha_j\,\frac{1}{M}\sum_{i=1}^{M}\varphi_j(\boldsymbol{x}_i)\,\varphi_k(\boldsymbol{x}_i) + \lambda\sum_{j=1}^{N}\alpha_j\int_{\Omega}\Lambda\varphi_j(\boldsymbol{x})\,\Lambda\varphi_k(\boldsymbol{x})\;\mathrm{d}\boldsymbol{x}\\ = \frac{1}{M}\sum_{i=1}^{M}\varphi_k(\boldsymbol{x}_i)\,y_i,\;\; k = 1, \ldots, N,\end{aligned} \tag{2.43}$$

or, in matrix-vector form:

$$\left(\frac{1}{M}\boldsymbol{B}\boldsymbol{B}^T + \lambda\boldsymbol{C}\right)\boldsymbol{\alpha} = \frac{1}{M}\boldsymbol{B}\boldsymbol{y}, \tag{2.44}$$

where, similarly to the notation introduced by Eq. (2.39), $C_{j,k} = \langle\Lambda\varphi_j, \Lambda\varphi_k\rangle_{L^2}$ and $\boldsymbol{B}_{k,i} = \varphi_k(\boldsymbol{x}_i)$.

Implications from a numerical and practical perspective of this resulting linear system will be covered in more details while discussing the code optimization process in Chapter 4.

2.2.3 Clustering using Sparse Grids

Problem statement. We need, as we have done before, to first define the problem we are trying to solve. The well-known task of clustering can be stated simply as:

> Given a set of data points $\mathcal{S} = \{\boldsymbol{x}_i\}_{i=1}^{M}$ in $\Omega \in \mathbb{R}^d$, split it into meaningful subsets (called *clusters*).

While everyone can agree that the description of the task is simple, the main reason why solving the problem of clustering is so equally sought after and hard to achieve is the fact that there is no unique definition of what the term "meaningful" actually represents, as it is something that changes not only in term of the context and type of input data, but also in terms of the algorithmic approaches taken.

The sparse grid algorithm. The idea introduced by Peherstorfer [61] for clustering using sparse grids is centered on a simple density-based description of clusters: on the one hand, data points that lie in high density regions are assumed to be more closely related to each other than to those that lie in low density regions, and on the other hand, data points lying in similarly high density regions that are completely separated by low density regions are supposed to be unrelated. This can be easily explained with the sketch in Fig. 2.11.

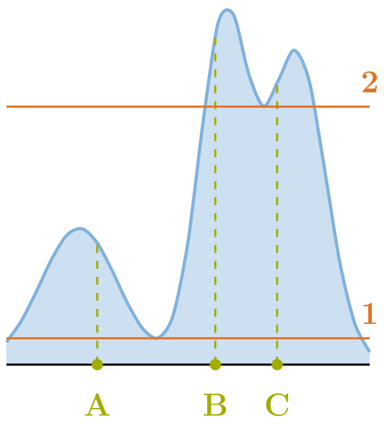


Figure 2.11: Sketch of an estimated density and the relationships between data points in terms of density levels. Points **A** and **B** (and, similarly, **A** and **C**) are well separated by a region of low density (line **1**), so it is very likely that they belong in different clusters. Points **B** and **C** however are less separated, as the minimal density level between them is relatively very high (line **2**), therefore it is less likely that they are in different clusters. Clearly this understanding of similarities in terms of densities has some limitations, especially in terms of defining more clearly how to identify these density levels that separate clusters. Such aspects are addressed by our contributions in this thesis, which will be expanded upon in Chapter 5.

The sparse grid approach is based on two components. Firstly, we require to approximate the densities at each data point. This is naturally done by performing SGDE on the input dataset. Secondly, we need a method to relate the data points to each other in a way that allows the handling of these density-based relations in order to define the different clusters. The method chosen by Peherstorfer is that of a similarity graph. The basic steps of Peherstorfer's algorithm can be summarized as:

Step 1: Estimate the density distribution of the input dataset $\mathcal{S}$.

Step 2: Construct a similarity graph G for the data points in $\mathcal{S}$.

Step 3: Use the estimated densities to split G into subsets.

Step 4: Assign labels to each obtained subset.

We will now go more into the details of each of these steps. The first one simply requires us to perform the SGDE algorithm, which we have already described in Chapter 3, therefore we won't go into any more details about it here.

The second step is done with a nearest neighbor algorithm using the Euclidean distance as a similarity metric. Peherstorfer opted for the naive approach of computing all M^2 pairwise distances, arguing that the computational costs can be offset by the simple parallelization options such an approach offers. (This and other performance aspects will be addressed shortly.) Using these distances a n-nearest neighbor graph is constructed, such that vertices represent data point and edges are linking each vertex to its closest (by distance) n neighbors. The value of n is a parameter of the clustering algorithm that has to be decided upon in advance.

The third step of the algorithm is where we merge the graph representation with the estimated density. The algorithm proposes to eliminate from the nearest neighbor graph all vertices (i.e., the data points) with a low density value together with their edges, as well as any additional edge that bridges two high density vertices over a low density region. In order to split the similarity graph into meaningful subsets by this method, we have to first define what we consider a low and high density region, respectively, for this density-based clustering. Peherstorfer introduced another parameter, a density threshold ϵ, such that density values below ϵ will be denoted as "low", and those above it as "high". While for vertices it is easy to decide in which category they are by simply evaluating the sparse grid density estimation at that data point, for edges the approach taken was to evaluate the density at the geometric midpoint of the edge. After this vertices and edges are eliminated, we are left with a graph $\hat{G} \subseteq G$ containing a certain number of, e.g., k connected components which we need to find.

The forth step is now simple: we identify these k connected components of the remaining graph and assign to each a unique label $1, ..., k$, representing a cluster. Each data point, or graph vertex, in $\hat{G}$ will thus be assigned the label of the component containing it. The remaining data points in $G \setminus \hat{G}$, eliminated in step 3, are considered noise for the purposes of our thesis.

We need to make a couple small remarks on the algorithm described so far. Firstly, in the original description by Peherstorfer, steps 1 and 2 are in reverse order. This is however actually of no consequence, as their respective operations are completely independent, meaning not only that the order does not matter, but also that they could be performed in parallel. We have chosen this particular ordering for the purposes of our thesis as it more clearly reflects our implementation (which will be addressed later). Secondly, the algorithm description we have opted for in our thesis contains only 4 steps, while the original in [61] contained 5, with the additional (last) step described as a classifier trainer on existing clusters in order to assign also to the eliminated low density data points one of the obtained labels $1, ..., k$. This is not an omission on our part, rather a conscious decision. Even Peherstorfer actually considered this extra step as optional, with low density points being just as well be classified as noise, an approach we have also taken in our implementation in Chapter 5, leaving the study of the best methods to implement such a fifth step as a future possible contribution.

Performance aspects. As already mentioned, one reason the naive nearest neighbor approach was used by Peherstorfer is because it lends itself to be easily parallelized. The main existing contribution in this regard, i.e., with a heavy HPC focus, was done by Pfander et al. [62], who have implemented a full OpenCL-based pipeline for the density-based clustering with sparse grids. For the density estimation component they straight-forward parallelized the computation of the right-hand side, and for the system matrix they opted not to decompose it, instead optimizing directly the matrix-vector operations of the conjugate gradient solver of the linear system. The graph creation, as mentioned, was easily parallelized across the data points. The graph pruning, i.e., removing the low density vertices and edges, can be done in two steps, each of them parallelizable across the data points. Lastly, the operation of determining the connected components was not distributed, but was still optimized for shared memory. Pfander et al. proved in their work that a highly optimized implementation of this approach allows the handling of datasets of up to 10^8 elements, viable on compute clusters.

This implementation however lacked the ability to adapt the sparse grid, which normally would lead to a significant decrease in performance due to the fact that the linear system needs to be solved again after each refinement and/or coarsening step. They assessed though that some of these performance losses could be offset by the use of a less naive, sub-quadratic, yet still parallelizable, approach for the nearest neighbor algorithm. Both hash- and tree-based methods are known to perform sub-quadratically and be parallelizable, an aspect which motivated one of our own implementation choices in Section 5.1.

All these are in line with our overall approach to our contributions to clustering methods in Chapter 5, where, although we have focused mainly on the usability criteria, we also took the algorithmic and high-performance aspects into account in both design and implementation.

3 Contributions with an Algorithmic Focus

While the modern study of probability and statistics is old (tracing back to the works of Pascal and Fermat in the 17th century), interest in the task of estimating (probability) density functions is comparably recent. First attempts at the mathematical formalism of the problem and initial numerical solutions have been provided in the works of Rosenblatt and Parzen in the 1950s and 1960s, who are usually also credited with the creation of the well-known kernel density estimation (KDE) method (see, e.g., [41] for a rundown of early contributions in the field). Since then most effort has been placed in improving and expanding these kind of kernel-based estimators, which do offer some nice properties, as they are non-parametric, however grow with the number of data points.

As mentioned in Chapter 2, Peherstorfer [60] was the one who introduced density estimation to the sparse grids community, and his contribution has been the basis for a significant amount of related work in the field and specifically in the SG++ library, most of it centered around the data mining pipeline of Röhner [69]. Contributions so far have only focused on improvements in the implementation of the method, or on expanding the range of applications, especially in the direction of density-based classification.

Our work in this chapter was motivated by, to the best of our knowledge, the lack of any study into more complex density estimators using sparse grids. In this category we include the solutions to the problems of estimating the difference and ratio of two probability functions, as well as the estimation of the derivative and derivative-ratio of a probability function. The same as with the original density estimation problem, these new estimators could become the basis of future work in various applications.

The chapter is structured as follows. First we will shortly describe and motivate the need for new density estimators. Then, for each such estimator we will present their purpose, the existing kernel-based counterparts, we will introduce mathematical derivations for our sparse grid-based methods, discuss properties and implementation considerations, and in the end present comparative results against kernel-based variants on various datasets.

3.1 Introduction to Complex Density Estimators

By a *complex estimator* we will refer to the solution of estimating directly a function (other than the identity) of one or more probability densities from corresponding independent samples of those probability distributions. Probably the most well-known places where complex estimators can be used are in the computation of divergences in statistics, which come under the form of integrals over functions of probability densities. Having not just accurate, but also easily integrable numerical approximations of those functions would allow for simpler computation of such quantities. Applications of such estimators are varied, kernel-based approaches having been used (besides the already mentioned divergence estimation [72]) for tasks such as time series segmentation [48], covariate shift adaptation[79], probabilistic classification [81], mode and ridge seeking [71] and feature selection [72], to name a few.

As hinted already, kernel-based approaches have been known for quite some time (for example, early work on direct density ratio estimation are already two decades old, as stated in [80]). However, since the introduction of sparse grid density estimation no attempts have been made (to the best of our knowledge) to assess the suitability of sparse grids in solving similar tasks. Our work comes thus to introduce such sparse grid variants for complex estimators that have found success in kernel-based implementations and assess their numerical properties, opening the possibility of future uses in various applications previously unattainable by existing sparse grid methods.

Before delving into these new algorithms, we want to address one final motivating aspect for our work. With an existing sparse grid-based algorithm for approximating quite well a probability density from samples, one could naturally pose the question of whether we actually need any direct procedures for estimating functions of densities. We bring two arguments to the affirmative. The first one is generic and is similar to the concept of transductive learning introduced by Vapnik in the late 1990s [81]: it is usually beneficial to avoid solving more general intermediate problems. Especially in the case of working with functions of more than one density (like the difference or ratio), estimating the result directly is less work than estimating the individual densities first. The second argument is more practical: increasing accuracy in individual estimated densities would not necessarily result in a similar increase of accuracy in a function that operates on those densities, as errors can easily get amplified. Therefore, in order to reduce the errors in the actual target of our estimation, it can be better to use a direct approach. This holds especially true when employing spatially adaptive sparse grids, where refinement and coarsening criteria would be much more complicated to devise for each individual density estimation than for the direct complex estimator.

For each proposed algorithm the corresponding section will focus first on the problem formulation, the mathematical derivation, the algebraic properties of the obtained linear system, and implementation and performance details. The numerical results subsection will then address the goals of the testing procedure, before describing how the tests are carried out in terms of the implemented pipeline, quality measures, and parameters used for both our sparse grid-based methods and the kernel-based counterparts we compare against. Our tests have been carried out on various single and mixed data distributions of different sizes and dimensionality in order to assess all relevant practical properties of our methods. We will then conclude each section with a short summary of the important aspects our tests revealed and a discussion on the usefulness and applicability of each method moving forward.

3.2 Density Difference Estimation

The problem statement. The first problem we want to address using sparse grids is *density difference estimation* (DDE). Mathematically, we can formulate this task as:

> Given two sets of independent and identically distributed samples in $\Omega \subseteq \mathbb{R}^d$, $\mathcal{S}_p = \{\boldsymbol{x}_i^p\}_{i=1}^{M_p}$ with density $p(\boldsymbol{x})$, respectively $\mathcal{S}_q = \{\boldsymbol{x}_j^q\}_{j=1}^{M_q}$ with density $q(\boldsymbol{x})$, estimate the difference $p(\boldsymbol{x}) - q(\boldsymbol{x})$.

Existing approaches. To our knowledge, the method proposed by Sugiyama et al. [78], called *least squares density difference* (LSDD), and with which we will compare later our sparse grid approach, is indeed the first attempt at a direct solve of the density difference estimation problem. As stated in their work, previously only studies on the suitability of differences of independent kernel density estimations (KDEs) have been investigated. A second variant, called *constrained LSDD* (CLSDD), showed a slightly improved accuracy, however at a higher computational cost [53]. Our literature review has seen neither a significant usage of this constrained variant in various applications of kernel-based density difference estimation, nor a freely available software implementation as in the case of LSDD, and as such we have restricted ourselves to compare our approach to the original version.

3.2.1 Sparse Grid Density Difference Estimation

Mathematical derivation. The derivation of the linear system of equations to be solved in our *sparse grid density difference estimation* (SGDDE) algorithm follows quite closely the derivation in the case of regular density estimation (Eqs. (2.33) to (2.39)), with the obvious difference that the target function is now given by $p(\boldsymbol{x}) - q(\boldsymbol{x})$.

Thus, we search for a function $r(\boldsymbol{x}) \in V$ in a suitable space that minimizes the regularized squared loss

$$J(r) = \int_\Omega \{r(\boldsymbol{x}) - [p(\boldsymbol{x}) - q(\boldsymbol{x})]\}^2 \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda r\|_{L^2}^2. \tag{3.1}$$

The corresponding variational equation will then be given by

$$\int_\Omega s(\boldsymbol{x})\, r(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} - \int_\Omega s(\boldsymbol{x})\, [p(\boldsymbol{x}) - q(\boldsymbol{x})]\ \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = 0,\ \forall s \in V. \tag{3.2}$$

Replacing the expectations in the second integral with the empirical estimators

$$p(x) \approx \frac{1}{M_p} \sum_{i=1}^{M_p} \delta_{\boldsymbol{x}_i^p}, \quad q(x) \approx \frac{1}{M_q} \sum_{j=1}^{M_q} \delta_{\boldsymbol{x}_j^q}, \tag{3.3}$$

and moving the second term to the right-hand side, we get

$$\int_\Omega s(\boldsymbol{x})\, r(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = \frac{1}{M_p} \sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p) \ - \ \frac{1}{M_q} \sum_{j=1}^{M_q} s(\boldsymbol{x}_j^q),\ \forall s \in V. \tag{3.4}$$

Using the sparse grid Ritz-Galerkin projection, we can restrict our search to the sparse grid space $V = \mathcal{V}_N$ with N grid points, i.e., $r(\boldsymbol{x}) := \sum_{k=1}^N \alpha_k \varphi_k(\boldsymbol{x})$ and $s(\boldsymbol{x}) := \varphi_l(\boldsymbol{x})$, obtaining

$$\begin{aligned} \sum_{k=1}^N \alpha_k \int_\Omega \varphi_k(\boldsymbol{x})\, \varphi_l(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} + \lambda \sum_{k=1}^N \alpha_k \int_\Omega \Lambda\varphi_k(\boldsymbol{x})\, \Lambda\varphi_l(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \\ = \frac{1}{M_p} \sum_{i=1}^{M_p} \varphi_l(\boldsymbol{x}_i^p) \ - \ \frac{1}{M_q} \sum_{j=1}^{M_q} \varphi_l(\boldsymbol{x}_j^q),\ l = 1, \dots, N. \end{aligned} \tag{3.5}$$

In matrix-vector form, the resulting linear system to solve for surpluses $\boldsymbol{\alpha}$ is thus

$$(\boldsymbol{R} + \lambda \boldsymbol{C})\, \boldsymbol{\alpha} = \boldsymbol{b}^p - \boldsymbol{b}^q, \tag{3.6}$$

with $R_{k,l} = \langle \varphi_k, \varphi_l \rangle_{L^2}$, $C_{k,l} = \langle \Lambda\varphi_k, \Lambda\varphi_l \rangle_{L^2}$, $\boldsymbol{b}^p = \frac{1}{M_p} \boldsymbol{B}^p \boldsymbol{e}^p$, and $\boldsymbol{b}^q = \frac{1}{M_q} \boldsymbol{B}^q \boldsymbol{e}^q$. As in Eq. (2.39), $\boldsymbol{B}^p_{l,i} = \varphi_l(\boldsymbol{x}_i^p)$, and correspondingly $\boldsymbol{B}^q_{l,j} = \varphi_l(\boldsymbol{x}_j^q)$, with $\boldsymbol{e}^p$ and $\boldsymbol{e}^q$ all-ones vectors of lengths M_p and M_q, respectively.

Properties of the linear system. Comparing the resulting linear systems, there is little differentiation between the usual density estimation and SGDDE. The only changes are those affecting the right-hand side, which now is computed based on two different input datasets, doubling thus the amount of operations usually performed on computing the term. However, as we work on independent datasets, the operations can in principle be performed in parallel up to the actual difference operation on the resulting vectors with data points contributions. Regarding the system matrix of SGDDE, it remains the same as in the case of SGDE, therefore it inherits all the benefits already described in Section 2.2.1.

Accuracy considerations. We are able to provide for this method a proof of consistency, in the sense that we can show that

$$\Pr\left(\lim_{M_p,\, M_q,\, N\,\to\,\infty} \|r-(p-q)\|_{L^2}^2 = 0\right) = 1, \tag{3.7}$$

with $r \in \mathcal{V}_N \subset H^2_{mix}$ being the solution of the sparse grid linear system in Eq. (3.6). This consistency assertion simply states that as we approach infinitely many samples from both input distributions on a sparse grid with a number of grid points approaching infinity our numerical solution approaches the exact density difference. Note that due to the fact that we operate with probability density functions we require the formulation of consistency to also be probabilistic, which however does not detract from its implications. Furthermore, in order for the sparse grid error bounds (as presented in Chapter 2) to hold we have to make the mild assumption that our target density difference function $p-q$ is also in H^2_{mix}. While this assumption is necessary theoretically, in practice sparse grids can deliver good results also for functions that violate this condition (see, e.g., [64]).

The proof of the consistency assertion in Eq. (3.7) follows the one described in [60]. If $\tilde{r} \in \mathcal{V}_N$ is the sparse grid interpolant of the target density difference $p-q$, we can write the error as

$$\|r-(p-q)\|_{L^2}^2 = \langle r-(p-q), r-\tilde{r}\rangle_{L^2} + \langle r-(p-q), \tilde{r}-(p-q)\rangle_{L^2}\,. \tag{3.8}$$

We will treat these two summands independently. For the first term of Eq. (3.8), as we consider the behavior at the limits $M_p \to \infty$ and $M_q \to \infty$, we can assume no overfitting takes place and thus no regularization is needed, i.e., $\lambda = 0$. Furthermore, as both r and $\tilde{r}$ are functions represented on sparse grids, their difference will also be representable in the same sparse grid space. Therefore, if we denote $s := r - \tilde{r} \in \mathcal{V}_N$ and we use Eq. (3.4), we can write:

$$\begin{aligned}\langle r-(p-q), r-\tilde{r}\rangle_{L^2} &= \langle r-(p-q), s\rangle_{L^2}\\ &= \langle r, s\rangle_{L^2} - \big(\langle p, s\rangle_{L^2} - \langle q, s\rangle_{L^2}\big)\\ &= \frac{1}{M_p}\sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p) - \frac{1}{M_q}\sum_{j=1}^{M_q} s(\boldsymbol{x}_j^q) - \big(\mathbb{E}_p(s(\boldsymbol{x})) - \mathbb{E}_q(s(\boldsymbol{x}))\big),\end{aligned} \tag{3.9}$$

where by $\mathbb{E}_f(X)$ we denote the expected value of X with respect to a density function f.

For the second term of Eq. (3.8) we can provide the upper bound

$$\begin{aligned}|\langle r-(p-q), \tilde{r}-(p-q)\rangle_{L^2}| &\le \|r-(p-q)\|_{L^2} \cdot \|\tilde{r}-(p-q)\|_{L^2}\\ &\le \frac{1}{4}\|r-(p-q)\|_{L^2}^2 + \|\tilde{r}-(p-q)\|_{L^2}^2\\ &\le \frac{1}{4}\|r-(p-q)\|_{L^2}^2 + \mathcal{O}\left(2^{-2l}\cdot l^{d-1}\right).\end{aligned} \tag{3.10}$$

In the first step we have applied directly the Cauchy–Bunyakovsky-Schwarz inequality. In the second step we have used Young's inequality for products via a small mathematical artifice, in the form of

$$a \cdot b = \frac{a}{\sqrt{2}} \cdot b\sqrt{2} \leq \frac{1}{2}\left(\frac{a}{\sqrt{2}}\right)^2 + \frac{1}{2}\left(b\sqrt{2}\right)^2 = \frac{1}{4}a^2 + b^2. \tag{3.11}$$

The third and last step in Eq. (3.10) uses the sparse grid interpolation error bound for piecewise linear basis functions under the already mentioned assumption of $p - q \in H^2_{mix}$. Here we assume the sparse grid is regular and of level l.

Combining Eqs. (3.9) and (3.10) into Eq. (3.8), we obtain after shuffling terms:

$$\begin{aligned} \|r - (p-q)\|^2_{L^2} \leq\ & \frac{4}{3}\,\mathcal{O}\left(2^{-2l} \cdot l^{d-1}\right) + \\ & \frac{4}{3}\left(\frac{1}{M_p}\sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p) - \frac{1}{M_q}\sum_{j=1}^{M_q} s(\boldsymbol{x}_j^q) - \left(\mathbb{E}_p(s(\boldsymbol{x})) - \mathbb{E}_q(s(\boldsymbol{x}))\right)\right). \end{aligned} \tag{3.12}$$

Under the limit $N \to \infty$, which translates to $l \to \infty$, the sparse grid error term converges to zero, while the second term probabilistically disappears due to the strong law of large numbers for $M_p, M_q \to \infty$ as the sample averages converge to their respective density's expected values. Therefore, as asserted, the sparse grid method is consistent: $\Pr\left(\|r - (p-q)\|^2_{L^2} = 0\right) = 1$.

Implementation details. The data mining pipeline as it was designed before our contributions worked on the assumption that a single input dataset would be enough. However, some of our new algorithms, SGDDE as an example, require the usage of more than one input dataset. Besides of course the extra code required to allow the processing of these new input parameters, one important aspect we wanted to keep form the regular SGDE implementation was the incremental batch learning of the input datasets.

The critical part of the whole batch processing concept is the combination step, i.e., the calculation that performs the incremental update of the right-hand side to account for the contribution of the current batch. Using a notation similar to [69], we can consider that datasets $\mathcal{S}_p$ and $\mathcal{S}_q$ are being processed simultaneously in batches of sizes $\tilde{M}_p^{(k)}$ and $\tilde{M}_q^{(k)}$, respectively, resulting in the batch right-hand sides $\tilde{\boldsymbol{b}}^{p^{(k)}}$ and $\tilde{\boldsymbol{b}}^{q^{(k)}}$, respectively (computed as in Eq. (2.39)). If we denote $M_p^{(k)} = \sum_{i=1}^{k} \tilde{M}_p^{(i)}$ and $M_q^{(k)} = \sum_{i=1}^{k} \tilde{M}_q^{(i)}$, the update rule for the right-hand side of the SGDDE linear system after k batches can be written as:

$$\boldsymbol{b}^{p(k)} - \boldsymbol{b}^{q(k)} = \left(\frac{M_p^{(k-1)}}{M_p^{(k)}}\boldsymbol{b}^{p(k-1)} - \frac{M_q^{(k-1)}}{M_q^{(k)}}\boldsymbol{b}^{q(k-1)}\right) + \left(\frac{\tilde{M}_p^{(k)}}{M_p^{(k)}}\tilde{\boldsymbol{b}}^{p(k)} - \frac{\tilde{M}_q^{(k)}}{M_q^{(k)}}\tilde{\boldsymbol{b}}^{q(k)}\right). \tag{3.13}$$

This is of course just the difference between the SGDE update rule as applied on $\mathcal{S}_p$ and the one on $\mathcal{S}_q$. This combination step holds only under an important condition: the grid has to remain the same during this whole process. The reason the grid could change is that adaptivity steps can take place between batches, not only at the end of a learning epoch (i.e., after going through the whole input datasets). This is yet another functionality stemming from this batch processing concept of the data mining pipeline, which allows in fact for the resulting sparse grid approximation to vary spatially, meaning that at different grid points only certain amounts of past batches can contribute to the estimation.

One would assume that also splitting the two datasets $\mathcal{S}_p$ and $\mathcal{S}_q$ into equal number of batches would need to be enforced additionally in order to properly apply batching. We consider this to be more of a conceptual condition for the combination step rather than a necessarily mathematical one, as there is no requirement in Eq. (3.13) for either $\tilde{M}_p^{(k)}$ or $\tilde{M}_q^{(k)}$ to be non-zero for any batch number k. However, it is hard to retrieve a proper understanding of what processing such a lob-sided batch would represent practically. With also good indications that not only the absolute (batch) sizes, but also the relationship between the (batch) sizes of the two input datasets affect the quality of the estimation (as shown in [78] and also in our experimental results to be presented), maintaining a proportionally stable batch size relationship between for the two input datasets is recommended, especially when grid adaptivity is being used.

The treatment of concept drift (similarly to the SGDE method described in [69]) can also be performed here. While not implemented in our code, the corresponding required modifications on Eq. (3.13) are presented in Appendix A.1.

Performance considerations. SGDDE resembles closely the general form of the SGDE linear system, therefore it can benefit from all the increased performance opportunities offered by the data mining pipeline when it comes to parallel processing of the matrix-vector operations. As such, in particular the ScaLAPACK implementations of all the system matrix decomposition strategies existing for SGDE can be used with minimal additional coding by the SGDDE implementation, offering considerable speed-ups (see [69] for details).

3.2.2 Numerical results

Goals. As SGDDE is a brand new method, the tests we have performed were done to assess the main qualitative properties of the estimator, as well as any possible weaknesses or caveats one should look out for and which might not be obvious from the mathematical model. Towards this purpose we have ran a wide array of tests with varying parameters (input datasets size, data dimensionality, grid level, regularization strength, adaptivity settings) in controlled scenarios, i.e., using samples from known data distributions, which is a well-known approach in this context. Additionally, we wanted to also assess the place our method takes in the current field of available approaches to solving the density difference estimation problem, and as such we have ran all tests also on an equivalent kernel-based method (i.e., LSDD) with which we could compare our SGDDE result.

Quality criteria. By using samples drawn from known probability density functions, i.e., with analytical solutions, we have a direct means of exactly evaluating the accuracy of our method, as well as as being able to directly compare errors between methods (in our case, SGDDE and LSDD). In order to obtain a comprehensive view of the performance of the implementations in terms of accuracy, we employed three measures:

- The *root mean squared error* (RMSE), given by

$$\mathrm{RMSE} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}\left(r(\boldsymbol{x}_i) - \tilde{r}(\boldsymbol{x}_i)\right)^2}, \tag{3.14}$$

  which is probably the simplest and most often used measure. A small RMSE shows that on average the errors are well-balanced across the points at which we evaluate our estimator. This error is the value we actually use to differentiate between test cases and choose the best results.

- The *mean absolute error* (MAE), given by

$$\mathrm{MAE} = \frac{1}{M}\sum_{i=1}^{M}\left|r(\boldsymbol{x}_i) - \tilde{r}(\boldsymbol{x}_i)\right|, \tag{3.15}$$

  as opposed to RMSE, gives a more uniform treatment of individual error terms due to their equal weighting. As such, it provides a better insight of how well the approximation at the evaluation points is overall.

- The *maximum absolute error* (MaxAE), given by

$$\mathrm{MaxAE} = \max_{i=1,\ldots,M}\left|r(\boldsymbol{x}_i) - \tilde{r}(\boldsymbol{x}_i)\right|, \tag{3.16}$$

  is generally used to detect outliers, in the sense of individual high error terms. Between approximations with equivalent MAE values, the one with the smallest MaxAE is preferable, as it signals a smaller spread of possible error magnitudes across the evaluation domain.

The evaluation points $\boldsymbol{x}_i$ chosen depend on the dimensionality of the input datasets. For one- and two-dimensional tests we can do an exhaustive equidistant evaluation grid of size 100, respectively 100×100, covering the whole computational domain. For higher-dimensional tests, where this approach would fast become computationally unfeasible, we consider as evaluation points the union set $\mathcal{S}_p \bigcup \mathcal{S}_q$ of both input datasets.

The testing pipeline. Our tests were performed using the Python[1] programming language [68], which was motivated by multiple factors. Firstly, the language is especially suitable at easily handling both the pre- and post-processing steps for our tests, which involve both input-output (IO) and numerical operations, like setting up the input configuration files for the SG++ runs, handling multi-dimensional and complex data structures, computing quantity of interest, and plotting and saving to file the final results. Secondly, no suitable implementation of the LSDD method, which we wanted to compare against, was available in SG++'s native implementation language (C++), but a Python variant did exist. This will be the case with all other kernel-based estimators mentioned in this chapter. With SG++ providing code wrappers for Python, a cohesive testing pipeline could be set up incorporating all aspects we required. Finally, Python is a powerful language in no small part due to its packages, which are in effect specialized mini-libraries. Of particular help were the packages of the SciPy ecosystem[2], especially `scipy`, which provided the functionalities needed to handle the probability data distributions used to sample our input datasets.

The testing pipeline works as follows: we first generate input datasets by sampling from certain single and mixed probability distributions (see Appendix B for more details on the different sample distributions used in our thesis) and we generate the appropriate configuration files needed for the SG++ runs. We then run both the respective SG++ and the kernel-based methods to obtain the density difference estimations. Lastly, we compute the quality criteria, we create plots (where suitable, i.e., for one- and two-dimensional cases), and save all results to file for further post-processing. This testing pipeline was implemented to allow the control of all important parameters of the sparse grid- and kernel-based methods.

Testing parameters. As our tests will compare our method to the kernel-based approach of LSDD, we need to address the parameters these implementations require. For LSDD we use an existing Python code[3] that implements the method as described in [78]. We use 5-fold cross-validation to select the best parameters from a grid space of 9 values for the kernel width $\sigma_K = \{10^{-3}, 10^{-2.6875}, \ldots, 10^{-0.5}\}$ and 5 values for the regularization parameter $\lambda_K = \{10^{-4}, 10^{-3}, \ldots, 10^{0}\}$. Here we use the subscript K to differentiate the parameters of the kernel-based approach to those of our sparse grid method. This search space was chosen such that it fits with the targeted computational domain of the sparse grids, i.e., the unit hypercube, while

[1]We refer specifically to Python 3, however, for brevity, we decided to omit the distinction. While we are aware that there still might be software that operates with Python 2, even though it reached its end-of-life on April 20, 2020, SG++ is not one of those pieces of software, therefore its code wrappers work only with Python 3 and any backward-compatibility in this regard is broken.

[2]`https://www.scipy.org/citing.html`

[3]Python code written by M.C. du Plessis, available at `http://www.ms.k.u-tokyo.ac.jp/sugi/software.html#LSDD`

offering a good trade-off between the added computational effort of cross-validation and a large enough search space in order to obtain credible and fair results.

In order to have a reasonable trade-off between performance and time-to-solution, we use the strategy from [78] to also limit the size of the system matrix of the LSDD linear system by fixing the maximum number of parameters (i.e., data points where to place the Gaussian kernels) to a fixed value b, meaning that the system matrix will be of fixed size $b \times b$ instead of $M \times M$, where M is, usually, taken as M_q. By comparison, the linear system to be solved in SGDDE has a system matrix of size $N \times N$, where N is the grid size. For one- and two-dimensional tests we use $b = 100$, while for higher-dimensional tests where we use larger dataset sizes and also the number of grid points increases we consider $b = \left\lfloor \frac{M}{50} \right\rfloor$.

For our sparse grid method we perform a separate grid search, combining different regularization parameter values λ, various grid discretization levels, grid adaptivity settings or dataset batching strategies. The actual search space for each test scenario will be mentioned explicitly when presenting the results in the following, and unless otherwise specified we will only be presenting the best parameter combination results in each case. For our tests we use only the linear basis functions, with no boundary points, due to the fact that we choose data distributions that have negligible values outside of the unit domain required by sparse grids. Additionally, and valid for all testing of our complex estimators, we use a simple sample rejection strategy to make sure we do not have any input data points outside of the interval, e.g., $[0.05, 0.95]$ (unless otherwise specified) in each dimension, such that we reduce even more any negative influences we might have due to data points too close to the boundary. These requirements for the input datasets are not actually restrictive due to the fact that, on the one hand, the difference of two density functions is well-behaved, in the sense that the tails of the difference density has a similar decay behavior as the input densities and, on the other hand, any input dataset can be easily transformed to fit in the required suitable sparse grid domain. However, for fairness of the testing procedure, we always choose parameters for the data distributions that fit directly, as best possible, in our desired computational domain.

1D results. The one-dimensional scenarios we first tested were

$$\mathcal{N}_1 := \begin{cases} \mathcal{S}_p \in \mathcal{N}\left(\mu = 0.5 + \epsilon,\ \Sigma = 0.01\right),\ \epsilon \in \{-0.2, -0.1, 0, 0.1, 0.2\}, \\ \mathcal{S}_q \in \mathcal{N}\left(\mu = 0.5,\ \Sigma = 0.01\right) \end{cases}$$

and

$$\mathcal{SN}_1 := \begin{cases} \mathcal{S}_p \in \mathcal{SN}\left(\xi = 0.5 + \epsilon,\ \Omega = 0.01,\ \alpha = -5\right),\ \epsilon \in \{-0.2, -0.1, 0, 0.1, 0.2\}, \\ \mathcal{S}_q \in \mathcal{SN}\left(\xi = 0.5,\ \Omega = 0.01,\ \alpha = -5\right) \end{cases},$$

i.e., we kept the $\mathcal{S}_q$ dataset fixed and tested 5 different shifted $\mathcal{S}_p$ distributions. The normal distribution is the standard default option in such cases. We however wanted to investigate also more realistic scenarios, which is why we also tested a skewed distribution. These tests act as a first validation of the sparse grid approach, as well as the simplest comparison scenario with the kernel-based approach. The sampling sizes we used were $M_p = M_q = \{200, 500, 750, 1000\}$. These small sizes were chosen specifically due to the fact that these are the main cases where the kernel method has seen most practical use. While it is known that in general sparse grid methods work best for larger datasets, we wanted to specifically test towards the lower bounds of usability for our new method in the hopes that it can achieve similar accuracies as the kernel-based approach. For the sparse grid runs we performed a grid search using 4 regularization parameters $\lambda = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and grid levels $l = \{3, 4, 5\}$.

The best results obtained in the two test scenarios in terms of minimum RMSE per test case are shown in Fig. 3.1 and Fig. 3.2, respectively. We observe that even for small datasets we obtained good accuracies, on par or even better than those obtained using the kernel-based approach. We got for LSDD similar RMSE values as those in literature for same-sized datasets and distributions (e.g., those in [78]), which validates also our choice of search space for the LSDD parameters. Of course, a more dense search space could bring some accuracy improvements, however we wanted to balance out as much as possible the computational effort compared to the sparse grid approach.

2D results. To assess the performance changes brought by the interaction of separate spatial dimensions, our next tests focused on 2D scenarios, whose parameters are tabulated in Table 3.1. The wide range of scenarios (short- and long-tailed, skewed and non-skewed, single and mixed distributions) are meant to truly push both methods in terms of approximation prowess. For the sparse grid runs we have again investigated the same range of grid levels and regularization parameter values, for 7 dataset sizes between 200 and 5000 (with $M_p = M_q$).

Results for minimum RMSE per test case are shown in Fig. 3.3. We observe that our sparse grid approach has good results also in the two-dimensional case, although at slightly lower accuracies than LSDD. This is not surprising, considering that the grid level is still relatively low, no adaptivity is being used, and the number of data points is also relatively low. For data distributions with strong axes alignment, like in our $m\mathcal{ST}_2$ test case, favorable to sparse grids, SGDDE performed even better than the Gaussian kernels of LSDD, which suffer from having to use the same parameters for each of the b chosen kernels. For a more complete view of the obtained results we also show in Fig. 3.4 a visual side-by-side comparison of the methods in two of these test cases for the largest dataset size and the best obtained parameters.

Higher-dimensional results. Lastly, the final tests were meant to compare the performance of the two density difference methods for larger datasets and in higher dimensions. Specifically, we ran tests on datasets of sizes $M_p = M_q = \{5000,$

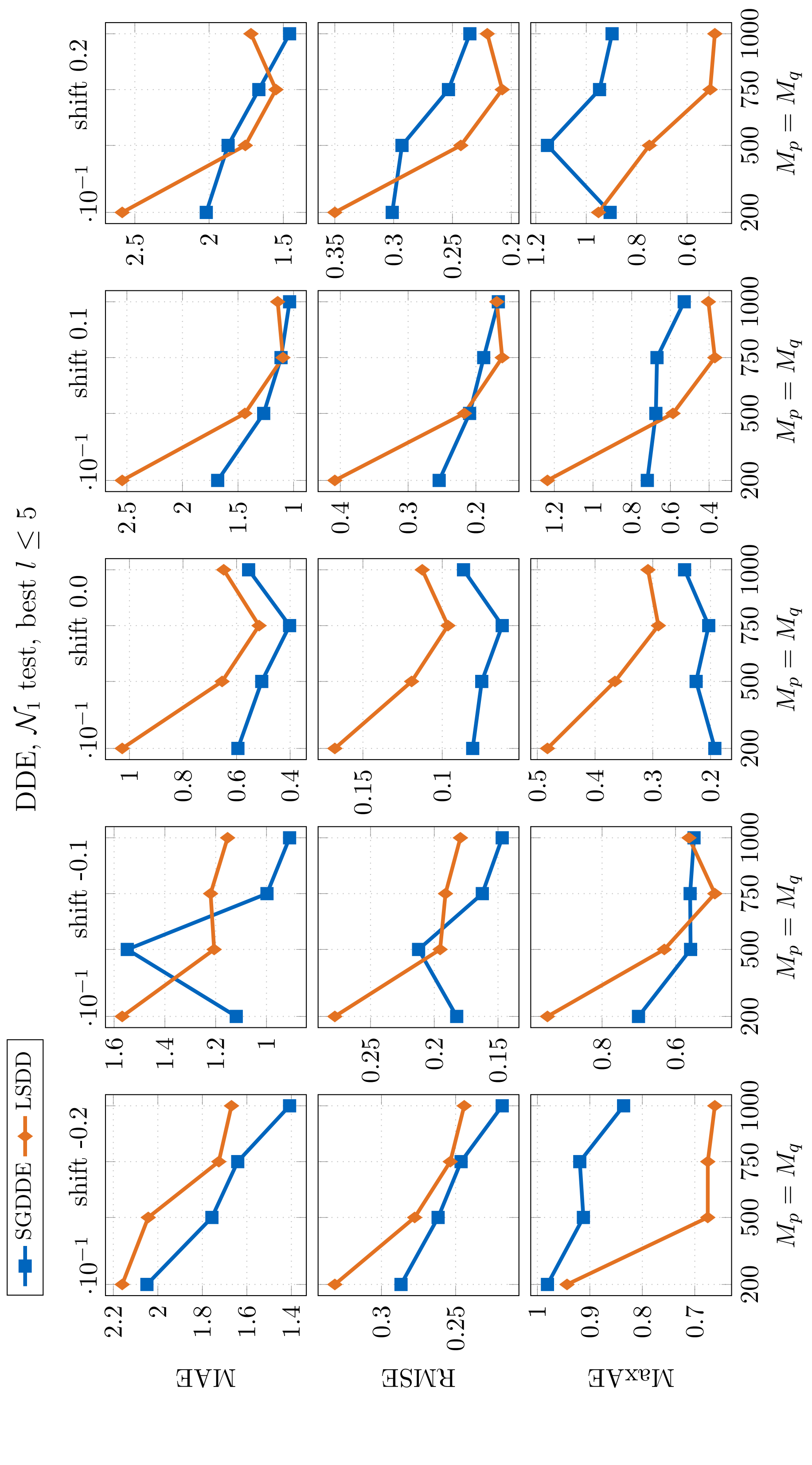


Figure 3.1: 1D DDE results for normal-distributed input datasets. We test for various shifts in the positioning of the $\mathcal{S}_p$ distribution, while keeping $\mathcal{S}_q$ fixed. While in general sparse grid methods are not very reliable for small datasets, we observe in general similar or better results for our SGDDE method compared to LSDD. Of course, if one requires a smooth approximation of the density difference, the kernel-based approach is qualitatively better.

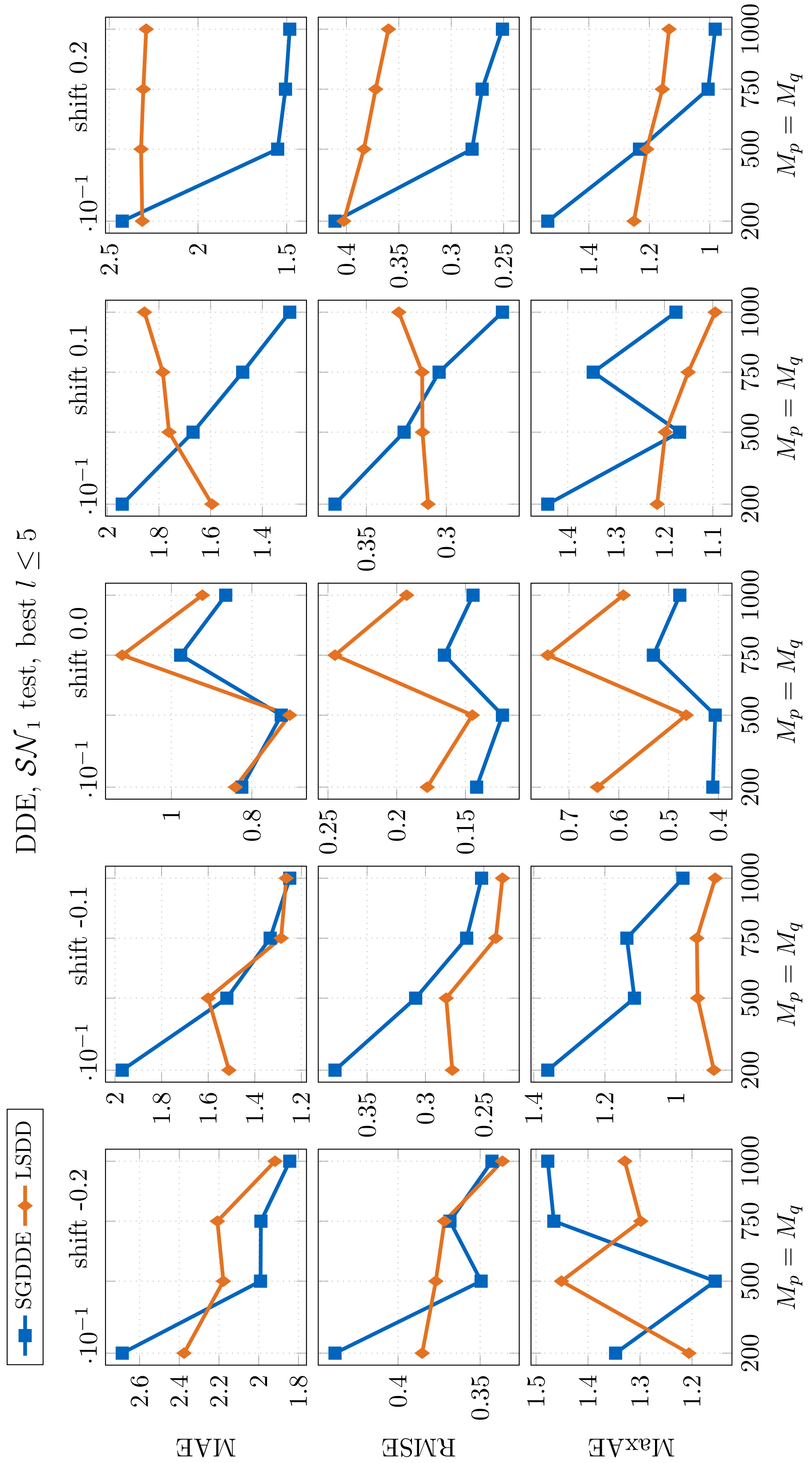


Figure 3.2: 1D DDE results for skew-normal-distributed input datasets. We test for various shifts in the positioning of the $\mathcal{S}_p$ distribution, while keeping $\mathcal{S}_q$ fixed. Similarly to the normal-distributed data, we observe in general similar or better results for our SGDDE method compared to LSDD, which handles small amounts of skewness well. However, again, if one requires a smooth approximation of the density difference, the kernel-based approach is qualitatively better.

Table 3.1: Parameters of the 2D DDE test scenarios.

test name	distrib. type(s)	distribution parameters
$\mathcal{N}_2$	$\mathcal{N}$	P: $\boldsymbol{\mu} = [0.4,\ 0.5]^T$, $\boldsymbol{\Sigma} = [[0.005,\ 0.01];\ [0.01,\ 0.04]]$ Q: $\boldsymbol{\mu} = [0.5,\ 0.5]^T$, $\boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.01]]$
$\mathcal{SN}_2$	$\mathcal{SN}$	P: $\begin{cases} \boldsymbol{\xi} = [0.5,\ 0.5]^T,\ \boldsymbol{\Omega} = [[0.02,\ 0.02];\ [0.02,\ 0.08]], \\ \boldsymbol{\alpha} = [5,\ -2]^T \end{cases}$ Q: $\begin{cases} \boldsymbol{\xi} = [0.4,\ 0.5]^T,\ \boldsymbol{\Omega} = [[0.06,\ 0];\ [0,\ 0.02]], \\ \boldsymbol{\alpha} = [2,\ -2]^T \end{cases}$
$\mathcal{T}_2$	$\mathcal{T}$	P: $\begin{cases} \boldsymbol{\mu} = [0.4,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.005,\ 0.01];\ [0.01,\ 0.04]], \\ \nu = 10 \end{cases}$ Q: $\begin{cases} \boldsymbol{\mu} = [0.5,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.01]], \\ \nu = 5 \end{cases}$
$\mathcal{ST}_2$	$\mathcal{ST}$	P: $\begin{cases} \boldsymbol{\xi} = [0.5,\ 0.5]^T,\ \boldsymbol{\Omega} = [[0.015,\ 0.015];\ [0.015,\ 0.045]], \\ \boldsymbol{\alpha} = [3,\ -2]^T,\ \nu = 10 \end{cases}$ Q: $\begin{cases} \boldsymbol{\xi} = [0.4,\ 0.5]^T,\ \boldsymbol{\Omega} = [[0.03,\ 0];\ [0,\ 0.015]], \\ \boldsymbol{\alpha} = [2,\ -2]^T,\ \nu = 5 \end{cases}$
$m\mathcal{N}_2$	P: $0.35 \cdot \mathcal{N}$ $+\,0.2 \cdot \mathcal{N}$ $+\,0.45 \cdot \mathcal{N}$ Q: $0.4 \cdot \mathcal{N}$ $+\,0.3 \cdot \mathcal{N}$ $+\,0.3 \cdot \mathcal{N}$	P: $\begin{cases} \boldsymbol{\mu} = [0.25,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.0045,\ 0.0075];\ [0.0075,\ 0.02]] \\ \boldsymbol{\mu} = [0.5,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.0045,\ 0.0075];\ [0.0075,\ 0.02]] \\ \boldsymbol{\mu} = [0.65,\ 0.4]^T,\ \boldsymbol{\Sigma} = [[0.0045,\ 0.0075];\ [0.0075,\ 0.02]] \end{cases}$ Q: $\begin{cases} \boldsymbol{\mu} = [0.5,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.01]] \\ \boldsymbol{\mu} = [0.5,\ 0.75]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.001]] \\ \boldsymbol{\mu} = [0.5,\ 0.25]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.001]] \end{cases}$
$m\mathcal{ST}_2$	P: $0.5 \cdot \mathcal{ST}$ $+\,0.5 \cdot \mathcal{ST}$ Q: $0.5 \cdot \mathcal{ST}$ $+\,0.5 \cdot \mathcal{ST}$	P: $\begin{cases} \boldsymbol{\xi} = [0.45,\ 0.65]^T,\ \boldsymbol{\Omega} = [[0.025,\ 0];\ [0,\ 0.005]] \\ \quad \boldsymbol{\alpha} = [0,\ 3]^T,\ \nu = 1 \\ \boldsymbol{\xi} = [0.65,\ 0.45]^T,\ \boldsymbol{\Omega} = [[0.005,\ 0];\ [0,\ 0.025]] \\ \quad \boldsymbol{\alpha} = [3,\ 0]^T,\ \nu = 1 \end{cases}$ Q: $\begin{cases} \boldsymbol{\xi} = [0.45,\ 0.55]^T,\ \boldsymbol{\Omega} = [[0.025,\ 0];\ [0,\ 0.005]] \\ \quad \boldsymbol{\alpha} = [0,\ -3]^T,\ \nu = 1 \\ \boldsymbol{\xi} = [0.55,\ 0.45]^T,\ \boldsymbol{\Omega} = [[0.005,\ 0];\ [0,\ 0.025]] \\ \quad \boldsymbol{\alpha} = [-3,\ 0]^T,\ \nu = 1 \end{cases}$

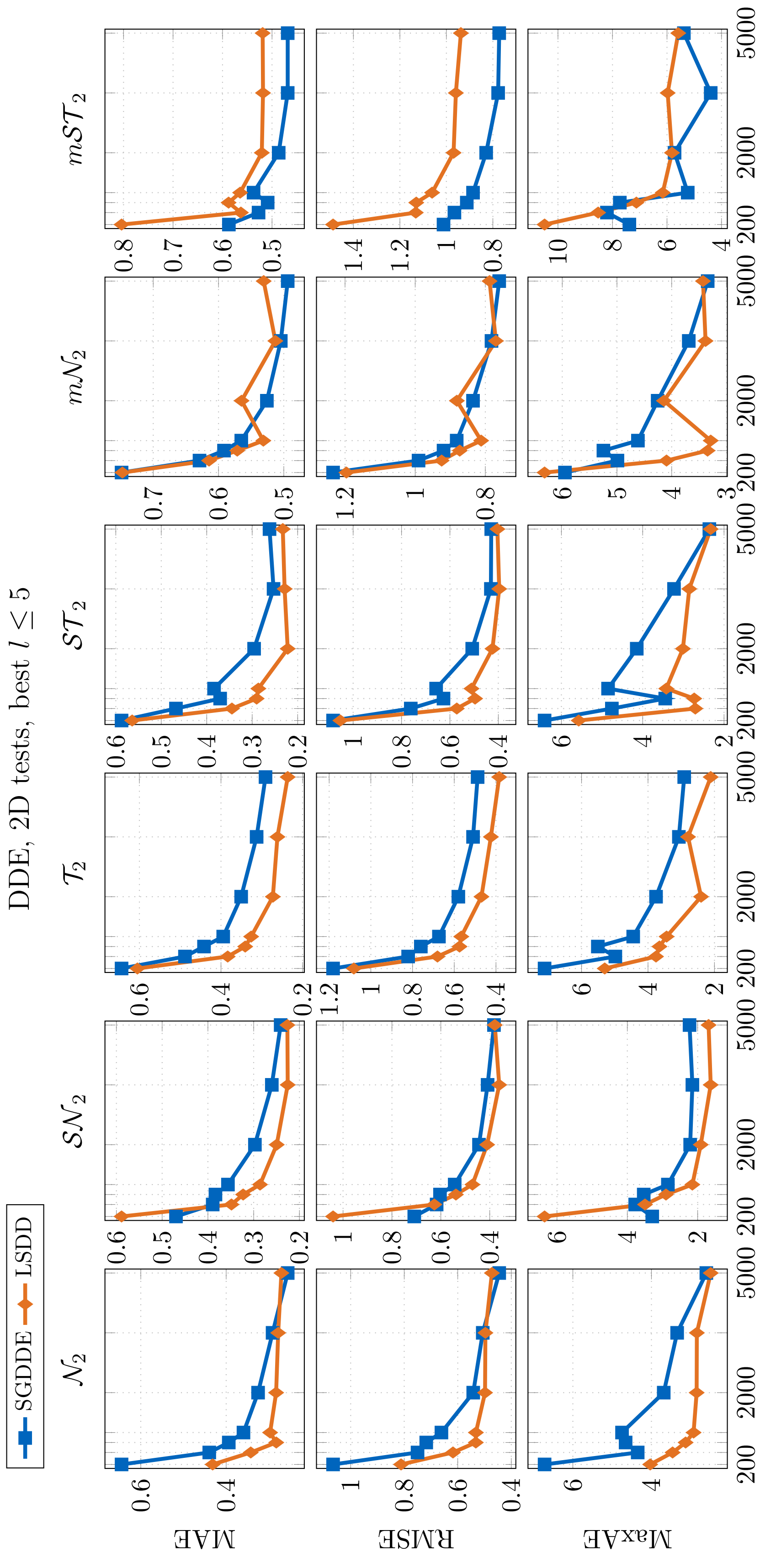


Figure 3.3: 2D DDE results on various datasets drawn from known single and mixed distributions. We observe that SGDDE obtains slightly under par results compared to LSDD, although with similarly good convergence rates with increasing dataset sizes.

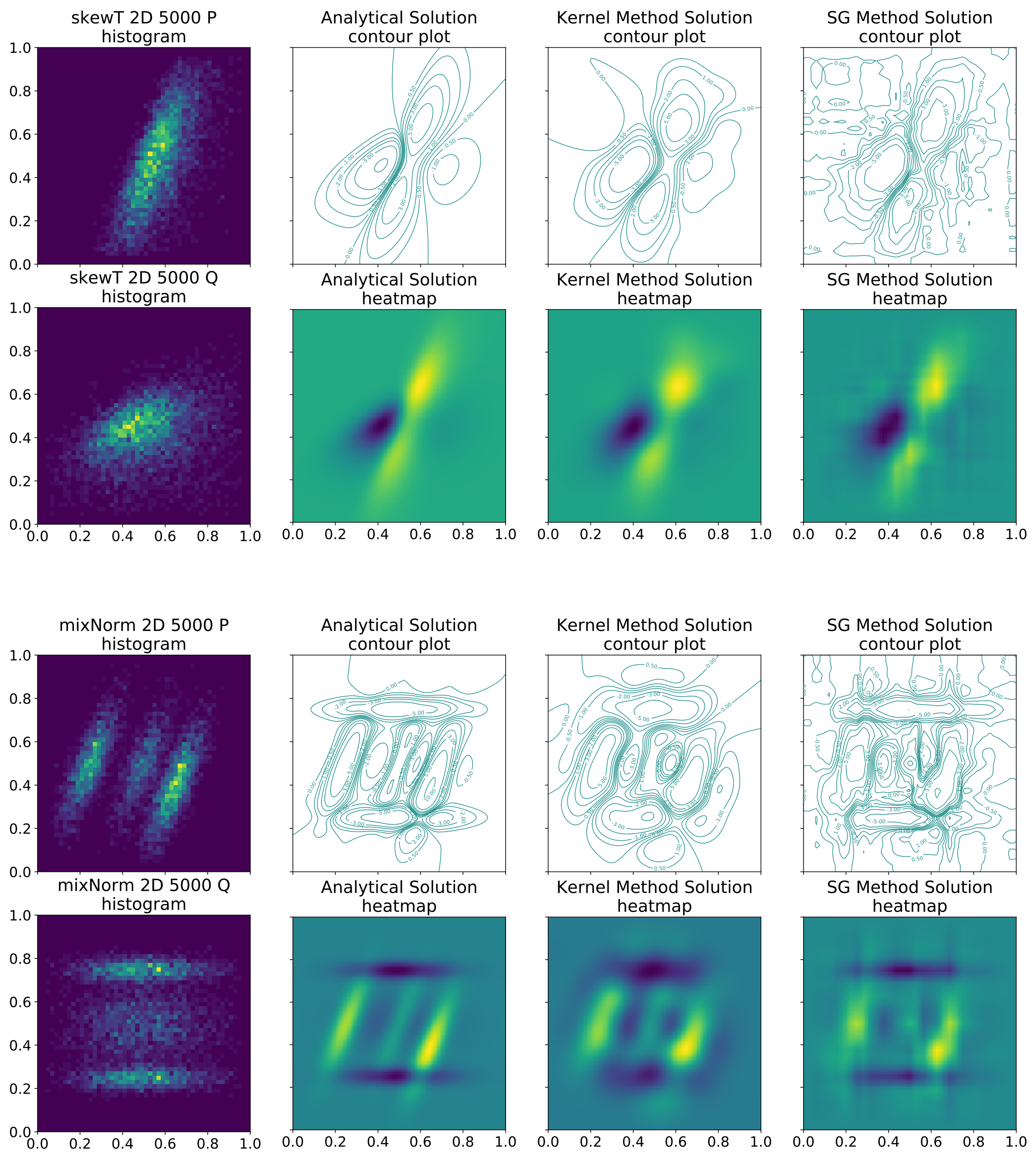


Figure 3.4: Visual inspection of the DDE results for the $\mathcal{ST}_2$ (*top*) and $m\mathcal{N}_2$ test scenarios. We compare the contour plot and heatmap of both LSDD and SGDDE for the best obtained parameters against the analytical solution, for the largest dataset size considered. We observe that both numerical solutions manage to capture well the target density difference. All other 2D test scenarios show similar visual properties.

Table 3.2: Parameters of the higher-dimensional DDE test scenarios.

test name		distribution parameters
$\mathcal{SN}_3$	P:	$\boldsymbol{\xi} = [0.5,\, 0.5,\, 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\, 0.015,\, 0.015]$, $\boldsymbol{\alpha} = [0,\, 2,\, -2]^T$
	Q:	$\boldsymbol{\xi} = [0.4,\, 0.4,\, 0.4]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\, 0.015,\, 0.015]$, $\boldsymbol{\alpha} = [0,\, -2,\, 2]^T$
$\mathcal{SN}_5$	P:	$\boldsymbol{\xi} = [0.5,\, 0.5,\, 0.5,\, 0.5,\, 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\, 0.01,\, 0.015,\, 0.015,\, 0.015]$, $\boldsymbol{\alpha} = [0,\, 0,\, 2,\, -2,\, 2]^T$
	Q:	$\boldsymbol{\xi} = [0.4,\, 0.4,\, 0.4,\, 0.4,\, 0.4]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\, 0.01,\, 0.015,\, 0.015,\, 0.015]$, $\boldsymbol{\alpha} = [0,\, 0,\, 2,\, 2,\, -2]^T$
$\mathcal{SN}_7$	P:	$\boldsymbol{\xi} = [0.5,\, 0.5,\, 0.5,\, 0.5,\, 0.5,\, 0.5,\, 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\, 0.01,\, 0.01,\, 0.015,\, 0.015,\, 0.015,\, 0.015]$, $\boldsymbol{\alpha} = [0,\, 0,\, 2,\, -2,\, 2,\, 2,\, -2]^T$
	Q:	$\boldsymbol{\xi} = [0.4,\, 0.4,\, 0.4,\, 0.4,\, 0.4,\, 0.4,\, 0.4]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\, 0.01,\, 0.01,\, 0.015,\, 0.015,\, 0.015,\, 0.015]$, $\boldsymbol{\alpha} = [0,\, 0,\, 2,\, 2,\, -2,\, -2,\, 2]^T$

$10000, 20000\}$, drawn from skew-normal distributions, whose parameters are given in Table 3.2. To have a fair comparison in terms of computational effort and accuracies, we use $b = 100, 200, 400$ kernels for LSDD, respectively, based on the increase in dataset sizes. For the sparse grid method we have additionally compared the performance between a fixed grid of level 6 and adaptive grids using the three refinement scores of (absolute) surplus (s), surplus-volume ($sVol$), and surplus-absolute-value ($sAbsVal$), as introduced in Chapter 2. For adaptivity we started with a level 3 grid and performed 5 adaptive steps of refining 15/20/30 grid points (for the 3/5/7-dimensional cases, respectively). For the regularization parameter we search for the best of 5 values $\lambda = \{10^{-1}, \ldots, 10^{-5}\}$.

The results of these higher-dimensional tests are presented in Fig. 3.5. While for the 3-dimensional case the kernel method performed better, as the dimensionality of the datasets increases we observe that the kernel-based approach becomes less accurate. As mentioned before, for these higher-dimensional cases the metrics are computed at the locations of the input datasets $\mathcal{S}_p \bigcup \mathcal{S}_q$, where we would expect LSDD to perform as best as possible for the parameters used. We also observe that the sparse grid method quickly saturates in terms in performance for the 5- and 7-dimensional cases due to the fact that we require a better combination of increase in both grid points and data points, not just the latter, in order to obtain a nice scaling in accuracy over the dataset sizes. Lastly, while in general the amount of adaptivity

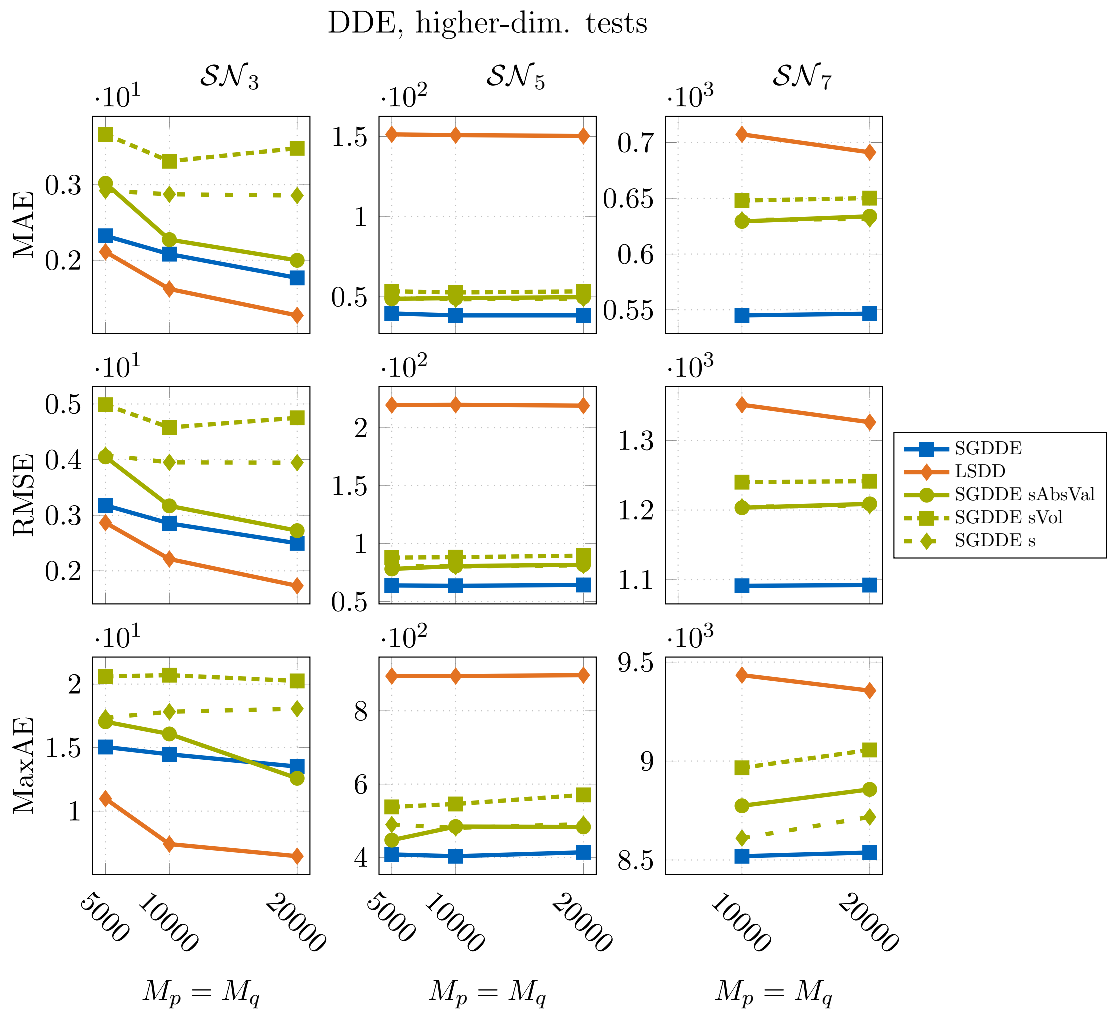


Figure 3.5: DDE results on higher-dimensional datasets. The error evaluations now take place at the input data points instead of an evaluation grid. While in 3 dimensions the kernel method obtains better accuracies, SGDDE takes over as the superior method already from dimension 5. From the 3 adaptive grids none has enough grid points to overtake the regular higher level grid in terms of accuracy, but we do observe that the surplus-absolute-value strategy provides the highest benefits.

we have considered did not push the metrics below those of the high level grid, we observe that the best refinement strategy remains the surplus-absolute-value, which Peherstorfer also used for the closely related method of SGDE [58].

Discussion of results. As expected from the similar type of obtained linear system like SGDE, SGDDE performed well overall, especially when enough grid and data points were invested. With a system matrix that is data-independent, the method can also scale well with the dataset. In our tests we have refrained from comparing runtimes, as neither the LSDD implementation nor the Python-wrapped SGDDE version we have used are optimal in terms of parallel performance. However, we observed in practice that SGDDE is faster than LSDD in both time to solution and evaluation time in single thread settings, even when accounting for cross-validation/parameters search space, therefore we expect any parallelized versions of the two to further capitalize on this performance difference.

Regarding accuracies we have shown that SGDDE can be a good choice even for smaller datasets, where usually sparse grid methods do not perform well. While the kernel-based approaches provide a smoother approximation (compared to our piecewise multi-linear sparse grid-based estimator), that might not always be required in practice. In higher dimensions and with enough data points the sparse grid method proved to be superior even when the number of unknowns were quite similar (grid points vs. number of kernels). This gives us hope that SGDDE could be a viable alternative to LSDD in some of the most used applications of density difference estimators.

Finally, an aspect where SGDDE can further improve based on LSDD is in properly handling hyperparameters. Currently there is no cross-validation parameter tuning process integrated in the SG++ data mining pipeline for methods that can handle multiple input datasets. Implementing such a procedure will unlock also for SGDDE the hyperparameter optimization strategies existing in the pipeline.

3.3 Density Ratio Estimation

Problem statement. The problem of *density ratio estimation* (DRE) can be formulated as follows:

> Given two sets of independent and identically distributed samples in $\Omega \subseteq \mathbb{R}^d$, $\mathcal{S}_p = \{\boldsymbol{x}_i^p\}_{i=1}^{M_p}$ with density $p(\boldsymbol{x})$, respectively $\mathcal{S}_q = \{\boldsymbol{x}_j^q\}_{j=1}^{M_q}$ with density $q(\boldsymbol{x})$, estimate the ratio $\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}$.

Existing approaches. While differences of probability densities look more natural to estimate, the ratio of two densities usually is a more powerful tool, which is why this quantity is also sometimes referred as the *importance* of the two distributions, a term most usually associated with the technique of importance sampling. This is also the context in which first contributions to direct density ratio estimation have been made, most notably the *kernel mean matching* (KMM) method [37] and the *Kullback-Leibler importance estimation procedure* (KLIEP) [79]. A more efficient approach that also borrows many of the advantages of the previous methods is the *least squares importance fitting* (LSIF) method and its more numerically stable counterpart, called *unconstrained LSIF* (uLSIF) [43], which is also the method we will be comparing our sparse grid approach to. Importance estimation has seen a very significant rise in interest, with a considerable amount of applications being linked to this learning task [80, 81]. Other more recent approaches that have seen good performance are the spectral series density ratio estimator of Izbicki et al. [38], which sees benefits in high-dimensional cases, and the nearest neighbor-based approach of Kremer et al. [46], which was used to tackle large astronomical datasets.

3.3.1 Sparse Grids Density Ratio Estimation

Mathematical derivation. Our algorithm for *sparse grid density ratio estimation* (SGDRE) starts with a similar loss function as the one used for LSIF (and, implicitly, uLSIF). That is partly because, as also stated in [43], this loss function formulation agrees with the various applications where the estimator can be used, and partly because this is a known technique of handling the implicit non-linearity of the ratio operation when moving to the variational description. Thus, in our sparse grid approach, we search for a function in a suitable space $r(\boldsymbol{x}) \in V$ that minimizes the regularized squared loss function

$$J(r) = \int_{\Omega} \left[r(\boldsymbol{x}) - \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \right]^2 q(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda r\|_{L^2}^2. \tag{3.17}$$

As such, the corresponding variational equation will be given by

$$\int_{\Omega} s(\boldsymbol{x})\, r(\boldsymbol{x})\, q(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int_{\Omega} s(\boldsymbol{x})\, p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_{\Omega} \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0, \, \forall s \in V. \tag{3.18}$$

Replacing the expectations in the second integral with the same empirical estimators from Eq. (2.35) and moving the second term to the right-hand side, we obtain

$$\frac{1}{M_q} \sum_{j=1}^{M_q} s(\boldsymbol{x}_j^q)\, r(\boldsymbol{x}_j^q) + \lambda \int_{\Omega} \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \frac{1}{M_p} \sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p), \, \forall s \in V. \tag{3.19}$$

Restricting our search to the sparse grid space $V = \mathcal{V}_N$ in the Ritz-Galerkin approach, i.e., $r(\boldsymbol{x}) \coloneqq \sum_{k=1}^{N} \alpha_k \varphi_k(\boldsymbol{x})$ and $s(\boldsymbol{x}) \coloneqq \varphi_l(\boldsymbol{x})$, we get

$$\begin{aligned}\sum_{k=1}^{N}\left[\alpha_k \frac{1}{M_q}\sum_{j=1}^{M_q}\varphi_l(\boldsymbol{x}_j^q)\,\varphi_k(\boldsymbol{x}_j^q)\right] + \lambda\sum_{k=1}^{N}\alpha_k\int_\Omega \Lambda\varphi_k(\boldsymbol{x})\,\Lambda\varphi_l(\boldsymbol{x})\;\mathrm{d}\boldsymbol{x} \\ = \frac{1}{M_p}\sum_{i=1}^{M_p}\varphi_l(\boldsymbol{x}_i^p),\;\; l=1,\ldots,N.\end{aligned} \tag{3.20}$$

In matrix-vector form, the resulting linear system of SGDRE to solve for surpluses $\boldsymbol{\alpha}$ is thus

$$\left(\frac{1}{M_q}\boldsymbol{B}^q\boldsymbol{B}^{qT} + \lambda\boldsymbol{C}\right)\boldsymbol{\alpha} = \frac{1}{M_p}\boldsymbol{B}^p\boldsymbol{e}^p, \tag{3.21}$$

where we use the same notation as in Eq. (3.6).

Properties of the linear system. As can be seen from Eq. (3.21), the system matrix for density ratio estimation is not data-independent anymore. Therefore, the system matrix now also grows with respect to the size of the input data, not just with respect to the grid size. However, it should be noted that the interaction between the two probability distributions p and q has been split in the linear system. For SGDDE, the right-hand side coalesced both contributions from $\mathcal{S}_p$ and $\mathcal{S}_q$, but for SGDRE the right-hand side now only grows with M_p, while the system matrix of the left-hand side grows with M_q. This is not very surprising, as the expectation under probability term we considered in the loss function of Eq. (3.17) already shows that the samples of distribution q bare a greater role in the final estimate.

The linear system we obtain in this density ratio scenario is very similar to the one for regression and classification which is covered extensively in Chapter 4. Without going here into more details, we can view the $\mathcal{S}_p$ samples as the target and the $\mathcal{S}_q$ samples as the source, such that our SGDRE algorithm learns a direct mapping between the two, which in essence is what the importance (or ratio) of the two represents.

Accuracy considerations. While stemming from a similar derivation based on the same spline smoothing idea, the nature of the density ratio estimator does not lend itself to a similar proof of consistency as in the case of SGDDE. With a linear system that resembles the one obtained in the case of regression tasks, we can only assess that a similar convergence behavior can be obtain as in those scenarios. While it is true that work on obtaining convergence rate bounds took place only for wavelet-based basis functions, which can form a Riesz frame with nice analytical properties [8], in practice sparse grid regression showed that it can obtain good results also for other bases, e.g., the hierarchical hat functions or the uniform B-splines basis [64]. Therefore, we expect that our sparse grid density ratio estimator will show similar behaviors (for both basis that form a Riesz frame, where theoretical bounds can be set, but also for more general bases).

Implementation details. Being another method that utilizes two datasets as inputs, SGDRE can reuse many of the interfaces and preprocessing code introduced for density difference estimation. However, no system matrix decomposition algorithms can be applied anymore. Also, the batch processing system of the pipeline is less powerful when it comes to SGDRE, as it works fundamentally differently. The main advantage of splitting the input datasets into batches in the previous density estimators was the ability to recover, on a fixed grid, the results on the full input datasets from the individual batch contributions, as only the right-hand side changed from one evaluation to the next. This is not the case anymore in the SGDRE case. However, batches can still be used in an implicit pseudo-concept drift adaptation, where we do not remember any of the previous batches, but allow grid adaptivity to carry on influences from past batches into the future. This aspect is however not covered in our work.

Performance considerations. Neither parallelizing across batches nor the ScaLAPACK variants for SGDE-like system matrix decompositions can be used for SGDRE. However, as our algorithm is similar to classical sparse grid-based regression/classification, optimizations such as the ones mentioned in Chapter 4 could be introduced in the data mining pipeline. We can expect that whenever these optimizations will actually be integrated into the pipeline the performance obtained will be quite similar to those presented there, especially considering that the right-hand side calculations can be done independently from the system matrix ones, with the number of additional operations in comparison to a usual regression/classification problem being at most in $o(1)$ with respect to the sizes of the grid and of the input datasets.

3.3.2 Numerical results

Goals. Similarly to SGDDE, SGDRE is also a brand new sparse grid method, therefore we perform tests to assess the properties of the method, strong points and weaknesses. This time however, with a regression-like type of linear system, we have less information on the approximation accuracies we can expect for such an estimator. Additionally, we want to assess to what degree our approach compares favorably to the equivalent kernel-based uLSIF method. We again use known data distributions in order to properly control the testing scenarios.

Quality criteria. For this method we use the same metrics of mean absolute error (MAE), root mean squared error (RMSE), and maximum absolute error (MaxAE) introduced in previously for DDE in order to compare both our sparse grid-based and the kernel-based methods against analytical solutions to the density ratio estimation problem on various datasets.

For one- and two-dimensional datasets we again evaluate our estimators on a grid of 100 equidistant points per dimension, while in the higher dimensional scenarios we use as evaluation points the reunion of all input data points, i.e., $\mathcal{S}_p \bigcup \mathcal{S}_q$.

The testing pipeline. Being also a method requiring two input datasets, we use mostly unchanged the testing pipeline for density difference estimation, simply using a python variable to differentiate between the two learning tasks. Therefore, again we use the Python language to perform all the tests, as well as a selection of various data distributions from which to sample our input datasets (see Appendix B).

Testing parameters. For the kernel-based uLSIF method we use an existing Python code[4] that implements actually a relative density ratio estimation kernel-based algorithm, which however can be reduced to the simple density ratio case via the $\omega = 0$ parameter (more details in the next section). The important aspect for our purposes is that this code follows, under this setting, the uLSIF algorithm as described in [43]. The same as for DDE, we use the code's 5-fold cross-validation to select the best hyperparameters from a search space of 9 values for the kernel width $\sigma_K = \{10^{-3}, 10^{-2.6875}, \ldots, 10^{-0.5}\}$ and 5 values for the regularization parameter $\lambda_K = \{10^{-4}, 10^{-3}, \ldots, 10^{0}\}$ (with the subscript K used here simply to differentiate these parameters from those for our sparse grid method).

Again we use the strategy of limiting the number of kernels used to $b = 100$ for the 1- and 2-dimensional test cases, increasing it to $b = \lfloor \frac{M}{50} \rfloor$ for the higher-dimensional datasets, where M is usually taken to be M_q. Therefore, while for the sparse grid method we work with matrices of sizes $M_p \times N$ and $M_q \times N$, where N is the number of grid points, in uLSIF the matrices used are of sizes $M_p \times b$ and $M_q \times b$, respectively. However, similarly to DDE, we still end up with a $N \times N$ system matrix in the case of SGDRE, respectively $b \times b$ for uLSIF.

1D results. As opposed to density differences, there is no expectation for density ratios to have "nice" properties even when the input datasets follow simple distributions. By "nice" we refer to having both a (mostly) compact support and bounded values. Therefore, for a first test we have considered the one-dimensional scenario

$$\mathcal{N}_1 := \begin{cases} \mathcal{S}_p \in \mathcal{N}\left(\mu = 0.55,\ \Sigma = 0.0125\right) \\ \mathcal{S}_q \in \mathcal{N}\left(\mu = 0.45,\ \Sigma = 0.05\right) \end{cases}$$

for which we try to approximate both the direct ratio P/Q and the inverse Q/P. For the first case the chosen distribution parameters produce a "nice" result, therefore we can default to using linear hat functions as sparse grid basis. For the inverse ratio, where we expect the opposite properties from the analytical result, we opt

[4]Python code written by Koji Makiyama and Ameya Daigavane, available at `https://github.com/hoxo-m/densratio_py`

for the modified linear basis functions for SGDRE in order to avoid increasing our grid size with boundary points. In both cases we fix the numerator dataset size at 200 and vary the denominator dataset size through the values $\{200, 500, 750, 1000\}$. This was done in order to not only assess the performance on smaller datasets, where sparse grids usually do not perform the best, but also because in the task of importance estimation the denominator density is the one for which more samples (and information) are usually available. For SGDRE we search for the best hyperparameters in a search space of 4 regularization parameters $\lambda = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and 3 grid levels $l = \{3, 4, 5\}$.

Results for best RMSE values per test are shown in Fig. 3.6. For the direct ratio case (P/Q), which we will call also the "linear" case, based on the nature of basis functions we have to use for sparse grids, we observe that, when good parameters are chosen via the uLSIF cross-validation algorithm, we obtained very similar results between our approach and the kernel-based one. Compared to LSDD, the cross-validation algorithm for uLSIF selects good parameters less consistently, the method being still prone to overfitting. In the specific set of results we show here this happened for M_p values of 500 and 750. In the inverse ratio case (Q/P), which we call also the "modlinear" case, we can observe that the sparse grid method improves consistently over the kernel-based method due to the fact that kernel positions are linked to data points and thus the approximation accuracies towards the boundaries of the computational domain suffer significantly. Overall, for both methods, we see that unilaterally increasing the denominator dataset size did not seem to have any direct effect on the overall accuracy of the estimation.

2D results. The next step was to evaluate multi-dimensional test cases. For this we started with the linear cases in two dimensions, i.e., those scenarios where we expect "nice" density ratios P/Q and thus the kernel method performs best and where we can use the linear basis functions for SGDRE. The parameters for the input datasets are tabulated in Table 3.3. We test a wide range of distributions to account for different scenarios that could appear in real datasets. For SGDRE we search for the best hyperparameters (regularization parameter and grid level) in the same space of values as for the 1D tests, and we again consider 7 different dataset sizes (with $M_p = M_q$) between 200 and 5000.

The results for minimal RMSE per test case are shown in Fig. 3.7. SGDRE looks to be able to come close to the accuracies of uLSIF even for smaller datasets, although the kernel-based approach proves to be more accurate. We also can observe a quite good converge rate with increasing dataset sizes, due to the fact that enough grid points are being invested. We again see here some minor inconsistencies in the cross-validation parameter selection for the kernel-based method which results in less than optimal parameters and thus less than optimal accuracies. Also visually these slightly overfitted uLSIF results are less problematic in the 2D scenarios than in the one-dimensional tests (as exemplified for a couple scenarios in Fig. 3.8),

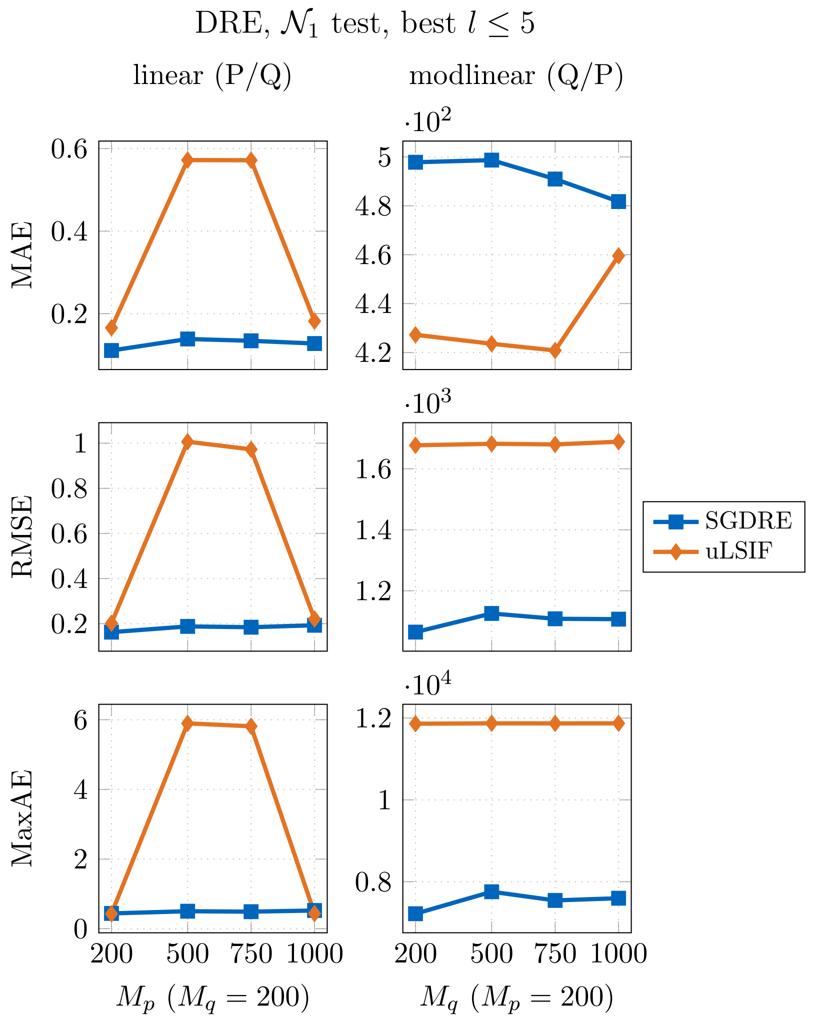


Figure 3.6: 1D results on direct (P/Q) and inverse (Q/P) density ratio estimation on normal-distributed input datasets. The P/Q ratio fades towards the boundary and thus we use linear basis functions on SGDRE; the opposite happens for the Q/P ratio, thus we need to use the modlinear basis. In general, we have observed some consistency issues with the uLSIF implementation, with non-optimal kernel parameters being sometimes selected in the cross-validation procedure, as exemplified here in the linear case. When best parameters have been chosen, we can see that SGDRE can obtain on par results to uLSIF. For the inverse ratio, which is non-zero towards the boundary, the modlinear basis functions show their superiority to the kernel method, which can only provide good accuracies at the input data points, not over the whole unit domain.

Table 3.3: Parameters of the 2D DRE test scenarios.

test name	distrib. type(s)	distribution parameters
$\mathcal{N}_2$	$\mathcal{N}$	P: $\boldsymbol{\mu} = [0.55,\ 0.5]^T$, $\boldsymbol{\Sigma} = [[0.008,\ 0.0015];\ [0.0015,\ 0.0065]]$ Q: $\boldsymbol{\mu} = [0.5,\ 0.35]^T$, $\boldsymbol{\Sigma} = [[0.04,\ 0];\ [0,\ 0.015]]$
$\mathcal{SN}_2$	$\mathcal{SN}$	P: $\begin{cases} \boldsymbol{\xi} = [0.55,\ 0.5]^T, \\ \boldsymbol{\Omega} = [[0.014,\ 0.002625];\ [0.002625,\ 0.011375]], \\ \boldsymbol{\alpha} = [-1,\ 2]^T \end{cases}$ Q: $\begin{cases} \boldsymbol{\xi} = [0.5,\ 0.35]^T,\ \boldsymbol{\Omega} = [[0.07,\ 0];\ [0,\ 0.02625]], \\ \boldsymbol{\alpha} = [-1,\ 1]^T \end{cases}$
$\mathcal{T}_2$	$\mathcal{T}$	P: $\begin{cases} \boldsymbol{\mu} = [0.55,\ 0.5]^T, \\ \boldsymbol{\Sigma} = [[0.008,\ 0.0015];\ [0.0015,\ 0.0065]], \\ \nu = 10 \end{cases}$ Q: $\begin{cases} \boldsymbol{\mu} = [0.5,\ 0.35]^T,\ \boldsymbol{\Sigma} = [[0.04,\ 0];\ [0,\ 0.015]], \\ \nu = 5 \end{cases}$
$\mathcal{ST}_2$	$\mathcal{ST}$	P: $\begin{cases} \boldsymbol{\xi} = [0.55,\ 0.5]^T, \\ \boldsymbol{\Omega} = [[0.01,\ 0.001875];\ [0.001875,\ 0.008125]], \\ \boldsymbol{\alpha} = [-1,\ 2]^T,\ \nu = 10 \end{cases}$ Q: $\begin{cases} \boldsymbol{\xi} = [0.5,\ 0.35]^T,\ \boldsymbol{\Omega} = [[0.05,\ 0];\ [0,\ 0.01875]], \\ \boldsymbol{\alpha} = [-1,\ 1]^T,\ \nu = 5 \end{cases}$
$m\mathcal{N}_2$	P: $0.35 \cdot \mathcal{N}$ $+0.2 \cdot \mathcal{N}$ $+0.45 \cdot \mathcal{N}$ Q: $0.4 \cdot \mathcal{N}$ $+0.3 \cdot \mathcal{N}$ $+0.3 \cdot \mathcal{N}$	P: $\begin{cases} \boldsymbol{\mu} = [0.25,\ 0.6]^T,\ \boldsymbol{\Sigma} = [[0.0015,\ 0.0005];\ [0.0005,\ 0.0045]] \\ \boldsymbol{\mu} = [0.5,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.0015,\ 0.0005];\ [0.0005,\ 0.0045]] \\ \boldsymbol{\mu} = [0.65,\ 0.4]^T,\ \boldsymbol{\Sigma} = [[0.0015,\ 0.0005];\ [0.0005,\ 0.0045]] \end{cases}$ Q: $\begin{cases} \boldsymbol{\mu} = [0.5,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.01]] \\ \boldsymbol{\mu} = [0.5,\ 0.75]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.001]] \\ \boldsymbol{\mu} = [0.5,\ 0.25]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.001]] \end{cases}$

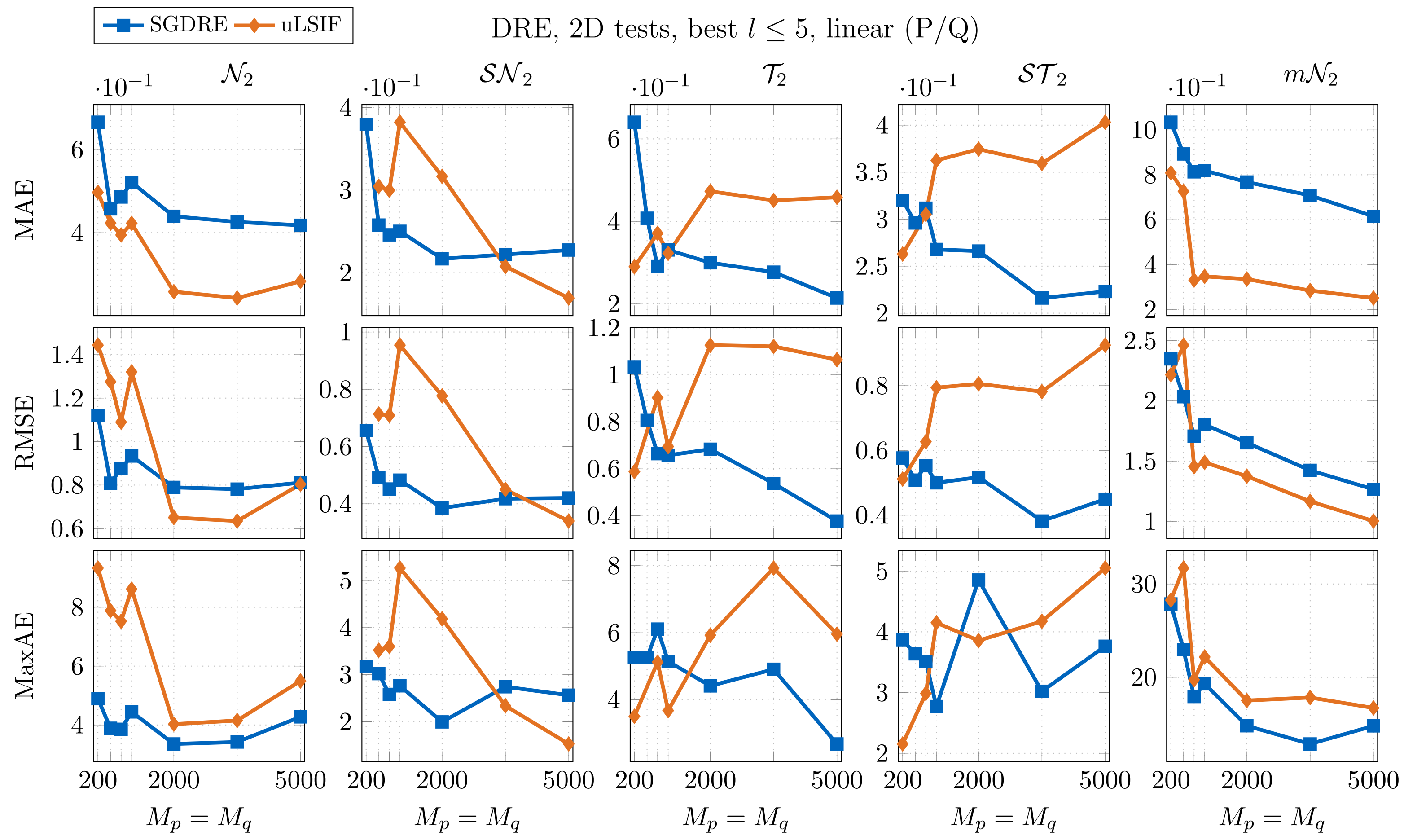
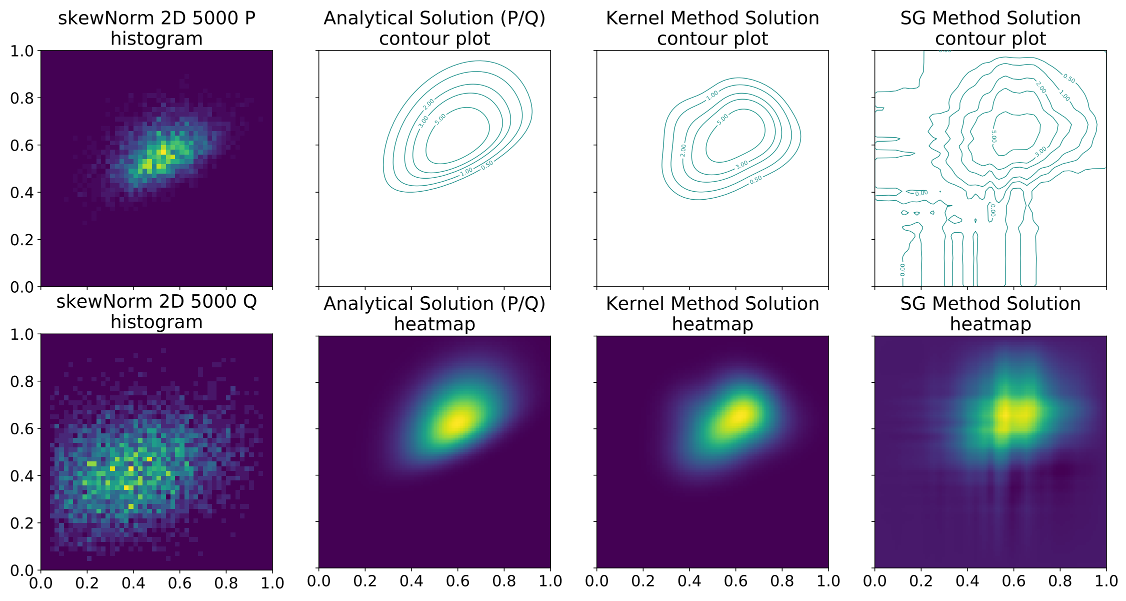


Figure 3.7: 2D DRE results on various input datasets drawn from known distributions. We look at density ratios scenarios which fade towards the boundary, where we can use linear basis functions. We observe again some problems with the cross-validation algorithm on uLSIF, which sometimes overfits the kernel parameters, although with less negative effects than in the 1D case. When parameters are well-chosen, uLSIF proves to be better, however SGDRE shows still decent accuracies and a good convergence rate.



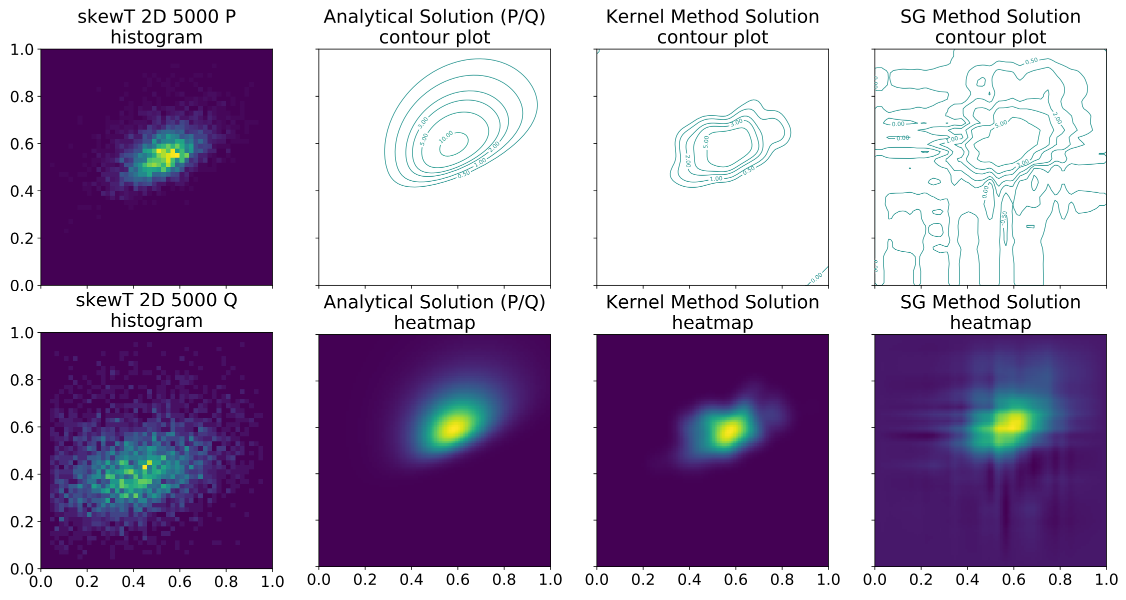


Figure 3.8: Visual inspection of the DRE results for the $\mathcal{SN}_2$ (*top*) and $\mathcal{ST}_2$ test scenarios. We compare the contour plot and heatmap of both uLSIF and SGDRE for the best obtained parameters against the analytical solution for the direct ratio P/Q, for the largest dataset size considered. We can observe here some of the overfitting problems that the integrated hyperparameter selection procedure of uLSIF does not always overcome, although the method still produces very good results nonetheless. All other 2D test scenarios show similar visual properties for the sparse grid-based approach, while the kernel-based method suffers from more severe overfitting in other cases, as revealed in the numerical results.

therefore overall uLSIF does look as quite a robust method, in line with existing results [43].

Discussion of results. For "nice" ratios, as was shown, the sparse grid approach with linear basis functions worked well, obtaining comparable results with the kernel method. We did observe for uLSIF a tendency to not select the best parameters, meaning that a better cross-validation approach might be needed, however the differences in obtained accuracies from the selected non-optimal hyperparameters were less strong in two dimensions that were found in the 1D case.

While performed, we will not present here results for the corresponding modlinear cases (i.e., for estimating inverse ratios Q/P for the same data distributions). Also results for higher dimensional datasets have been skipped. This is due to the fact that in multiple dimensions, in practice, it is impossible to actually predict, let alone ensure, boundedness of the target density ratio, which results in errors of very high magnitude for any kind of estimator, making comparisons between approaches unfeasible. These issues will be addressed in the next section with the introduction of relative density ratios.

3.4 Relative Density Ratio Estimation

Problem statement. The problem of *relative density ratio estimation* (RDRE) can be mathematically stated as [89]:

> Given two sets of independent and identically distributed samples in $\Omega \subseteq \mathbb{R}^d$, $\mathcal{S}_p = \{\boldsymbol{x}_i^p\}_{i=1}^{M_p}$ with density $p(\boldsymbol{x})$, respectively $\mathcal{S}_q = \{\boldsymbol{x}_j^q\}_{j=1}^{M_q}$ with density $q(\boldsymbol{x})$, estimate the ω-relative ratio $\frac{p(\boldsymbol{x})}{q_\omega(\boldsymbol{x})} = \frac{p(\boldsymbol{x})}{\omega \cdot p(\boldsymbol{x}) + (1-\omega) \cdot q(\boldsymbol{x})}$, $0 \leq \omega < 1$.

To avoid notation confusion between the relative density ratio parameter and the sparse grid variable we use here ω as main parameter for the method instead of α, which is used in literature.

Existing approaches. As shown by Kanamori et al. in [43], the problem of density ratio estimation can be ill-posed with regards to the possible unboundedness of the target function to be found. This is backed up by our own results with SGDRE. That is why Yamada et al. propose a slightly modified problem statement in the form of relative density ratio estimation [89], which we also import. Their *relative unconstrained least squares importance fitting* (RuLSIF) method is simply an extension of the uLSIF algorithm, in general offering some better situational convergence speed at a slightly higher computational cost.

3.4.1 Sparse Grid Relative Density Ratio Estimation

Mathematical derivation The derivations for *sparse grid relative density ratio estimation* (SGRDRE) will follow similar steps as for usual SGDRE, with the difference in the more complex denominator term. Thus, we start by searching for a function $r(\boldsymbol{x})$ in a suitable space V that minimizes the regularized squared loss function

$$J(r) = \int_\Omega \left[r(\boldsymbol{x}) - \frac{p\boldsymbol{x}}{q_\omega(\boldsymbol{x})} \right]^2 q_\omega(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda r\|_{L^2}^2. \tag{3.22}$$

The corresponding variational equation will be given by

$$\int_\Omega s(\boldsymbol{x})\, r(\boldsymbol{x})\, q_\omega(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int_\Omega s(\boldsymbol{x})\, p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0, \; \forall s \in V. \tag{3.23}$$

Replacing the expectations in the second integral with the empirical estimators (Eq. (2.35)), expanding the first term, and moving the second term to the right-hand side, we obtain

$$\begin{aligned} &\frac{\omega}{M_p} \sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p)\, r(\boldsymbol{x}_i^p) + \frac{1-\omega}{M_q} \sum_{j=1}^{M_q} s(\boldsymbol{x}_j^q)\, r(\boldsymbol{x}_j^q) \\ &\qquad + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \frac{1}{M_p} \sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p), \; \forall s \in V. \end{aligned} \tag{3.24}$$

Searching for a sparse grid solution $r(\boldsymbol{x}) \coloneqq \sum_{k=1}^N \alpha_k \varphi_k(\boldsymbol{x})$ in the space $V = \mathcal{V}_N$ with test functions $s(\boldsymbol{x}) \coloneqq \varphi_l(\boldsymbol{x})$ in the Ritz-Galerkin projection, we get

$$\begin{aligned} &\sum_{k=1}^N \left[\alpha_k \frac{\omega}{M_p} \sum_{i=1}^{M_p} \varphi_l(\boldsymbol{x}_i^p)\, \varphi_k(\boldsymbol{x}_i^p) \right] + \sum_{k=1}^N \left[\alpha_k \frac{1-\omega}{M_q} \sum_{j=1}^{M_q} \varphi_l(\boldsymbol{x}_j^q)\, \varphi_k(\boldsymbol{x}_j^q) \right] \\ &\qquad + \lambda \sum_{k=1}^N \alpha_k \int_\Omega \Lambda \varphi_k(\boldsymbol{x})\, \Lambda \varphi_l(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = \frac{1}{M_p} \sum_{i=1}^{M_p} \varphi_l(\boldsymbol{x}_i^p), \; l = 1, \ldots, N. \end{aligned} \tag{3.25}$$

In matrix-vector form, the resulting linear system to solve for surpluses $\boldsymbol{\alpha}$ is thus

$$\left(\frac{\omega}{M_p} \boldsymbol{B}^p \boldsymbol{B}^{pT} + \frac{1-\omega}{M_q} \boldsymbol{B}^q \boldsymbol{B}^{qT} + \lambda \boldsymbol{C} \right) \boldsymbol{\alpha} = \frac{1}{M_p} \boldsymbol{B}^p \boldsymbol{e}^p, \tag{3.26}$$

with the by now familiar notation of Eq. (3.6).

Properties of the linear system. Firstly we will investigate the meaning of the two edge cases:

- For $\omega = 0$, the relative ratio becomes the actual density ratio $\frac{p(x)}{q(x)}$, and our sparse grid linear system also becomes identical to the one for SGDRE (Eq. (3.21)), as expected.
- In the limit $\omega \rightarrow 1$, the relative ratio is the unit function, and our sparse grid algorithm becomes a least squares regression on the samples of distribution p, being fitted to those same $\mathcal{S}_p$ samples.

Therefore, the obtained linear system seems to maintain consistency with existing sparse grid approaches, with the ω-weights of the relative ratio denominator finding themselves now in the system matrix formulation. With still a data-dependent left-hand side, the properties of the obtained linear system for relative density ratio estimation are also carried over from SGDRE.

Accuracy considerations. Similarly to the particular case of density ratio estimation, for SGRDRE we can only expect to be able to assess some theoretical convergence rates for specific types of basis functions [8], while we hope to obtain for the hat functions and other more general bases convergence rates similar to the regular sparse grid regression solution.

Implementation details. Once SGDRE has been implemented, extending the method to SGRDRE is relatively straight-forward, with basically the same functionality lending itself to be reused with minimal changes. As stated, SGRDRE can also be used, in the limit case $\omega = 0$, to solve the direct density ratio estimation problem. We suggest for this purpose to always revert to using the SGDRE approach in order to avoid the (albeit relatively small) performance penalty incurred by the overhead of the relative density ratio implementation.

Performance considerations. The system matrix we obtain for relative density ratio estimation now extends the dependency from one of the input datasets to both of them, therefore the amount of computation needed for the left-hand side of the system almost doubles. However, as before, the computations involving $\boldsymbol{B}^p$ and $\boldsymbol{B}^q$ in the system matrix calculation can for the most part be done independently, therefore also in parallel (which is additional to the rest of the optimizations that can be done in the usual regression/classification scenario, as mentioned briefly for SGDRE and treated extensively in Chapter 4). Currently however, no such parallelization is implemented in our code, as it extends beyond the scope of our thesis.

3.4.2 Numerical results

Goals. The test we performed for SGRDRE had two purposes. On one hand we again wanted to assess the method itself, compared to a kernel-based counterpart, on various known distributions, to find strong and weak points of the approach and predict how well it could perform in various applications. On the other hand, this method is a direct extension to SGDRE, therefore we wanted to observe to what extent it can provide relief to the issues of density ratio unboundedness we have already noted in the previous section.

Quality criteria. Again we use the metrics of mean absolute error (MAE), root mean squared error (RMSE), and maximum absolute error (MaxAE) to compare both our sparse grid- and the kernel-based methods with the analytical solutions to the relative density ratio estimation problem on input datasets sampled from known distributions.

The evaluation for 1- and 2-dimensional datasets is performed once more on a grid of 100 equidistant points per dimension, while in the higher dimensional tests we evaluate at the reunion of all input data points, i.e., $\mathcal{S}_p \bigcup \mathcal{S}_q$.

The testing pipeline. As mentioned for DRE, all density ratio tests were carried out with the same code as for DDE, with a variable being used to distinguished between the two testing sub-pipelines. We also continue using the wide range of data distributions to sample from for our input datasets described in Appendix B.

Testing parameters. In our testing pipeline we compare our sparse grid method with an existing Python implementation of RulSIF[5], now for values of the ω parameter (named α in the RuLSIF implementation and literature [89]) other than 0. We keep the strategy of choosing hyperparameters via 5-fold cross-validation from 9 different kernel widths $\sigma_K = \{10^{-3}, 10^{-2.6875}, \dots, 10^{-0.5}\}$ and 5 regularization parameter values $\lambda_K = \{10^{-4}, 10^{-3}, \dots, 10^{0}\}$ (the subscript K used to distinguish from any similar sparse grid parameters).

Identical to the uLSIF method, only b kernels are used in order to limit the computational demands, with $b = 100$ for 1- and 2-dimensional tests, where we use smaller dataset sizes, and $b = \left\lfloor \frac{M}{50} \right\rfloor$ for the higher-dimensional tests, where M is usually the size of the denominator dataset (when $M_p \neq M_q$).

1D results. The first tests we performed extend the results shown for DRE by looking into the effects of the ω parameter for both direct (P/Q) and inverse (Q/P) ratios. We consider here $\omega = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The scenario parameters are

[5]Python code written by Koji Makiyama and Ameya Daigavane, available at `https://github.com/hoxo-m/densratio_py`

again

$$\mathcal{N}_1 := \begin{cases} \mathcal{S}_p \in \mathcal{N}\left(\mu = 0.55,\ \Sigma = 0.0125\right) \\ \mathcal{S}_q \in \mathcal{N}\left(\mu = 0.45,\ \Sigma = 0.05\right) \end{cases},$$

with sampled datasets of size 200 for the numerator input and $\{200, 500, 750, 1000\}$ for the denominator input. We again look for the best results for hyperparameters $\lambda = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and $l = \{3, 4, 5\}$.

The errors obtained in our tests are shown in Figs. 3.9 and 3.10, with minimal RMSE as main criterion. The first thing we can observe is a far more consistent cross-validation hyperparameter selection for RuLSIF, reflected in error values more closely packed in amplitude across the board. SGRDRE accuracies again are in line with the SGDRE results, with error closely matching those for RuLSIF. For the modlinear scenarios we more clearly see here the ability of the sparse grid-based approach to estimate the density ratios also outside of the support of the input datasets, which is one of the downsides of the kernel-based method. We also re-confirmed the fact that a unilateral increase of the denominator dataset size does not impact significantly the estimator, therefore a balanced increase of both input datasets sizes is preferred. One aspect to note, that is not explicitly visible from the data presented here, is that the sparse grid results were best in general for the lowest grid level $l = 3$, as the higher level grids showed already a slight overfitting. The rest of our tests will show that this was not a significant issue in higher-dimensional scenarios, which is anyway where sparse grids are actually most useful as a numerical method.

2D results. While for DRE we saw the performance of the sparse grid approach when the ratio to be estimated has a so-called “nice” distribution, in practice that is seldom the case. For the two-dimensional tests we wanted to assess the performance on quite unfavorable scenarios, where a non-zero ω value is needed to counteract an otherwise unbounded density ratio. The test scenarios are the same as those for DRE, i.e., those presented in Table 3.3, with the usual sparse grid hyperparameter search space being used and the previously considered dataset sizes, although now estimating the inverse (relative) ratios (Q/P) in each scenario.

Results for $\omega = 0.2$ and $\omega = 0.4$, based on minimal RMSE, are shown in Figs. 3.11 and 3.12, respectively. We see, as expected, across the board better accuracies from the sparse grid method compared to RuLSIF, which cannot provide good default estimates in areas where data points do not exist, in general, and towards the boundary of the numerical domain, in particular. We need to note that the very low error values obtained for the smallest datasets, which do not fit in the expected convergence rate curve we see for the larger input sizes, are a consequence of the fact that in our tests grids tended to perform in general surprisingly well on these datasets, which were not particularly axis-aligned or otherwise exhibiting properties traditionally considered well-suitable for sparse grids. One should not expect however this to always be the case. For completeness, we present also

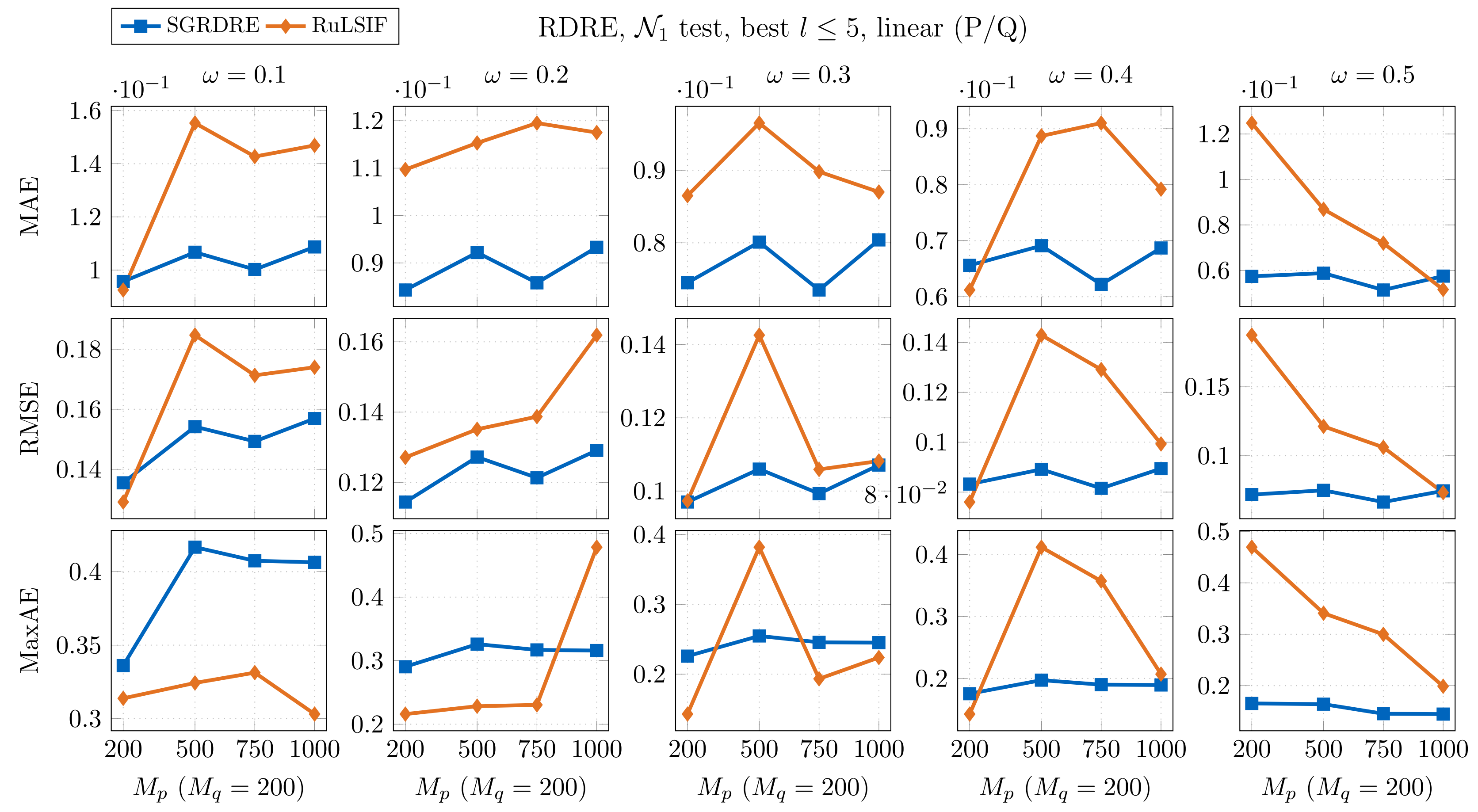


Figure 3.9: 1D RDRE results for varying ω values on normal-distributed input datasets. Direct ratios (P/Q) for these cases are bounded and taper towards the boundaries, which allow the use of linear basis functions, even before using the relative ratio parameter ω. We again see some negative effects of non-optimal kernel parameters from cross-validation inconsistencies, but at a lesser extent. SGRDRE obtains across the board on par results with the cases where the optimal kernel parameters are chosen.

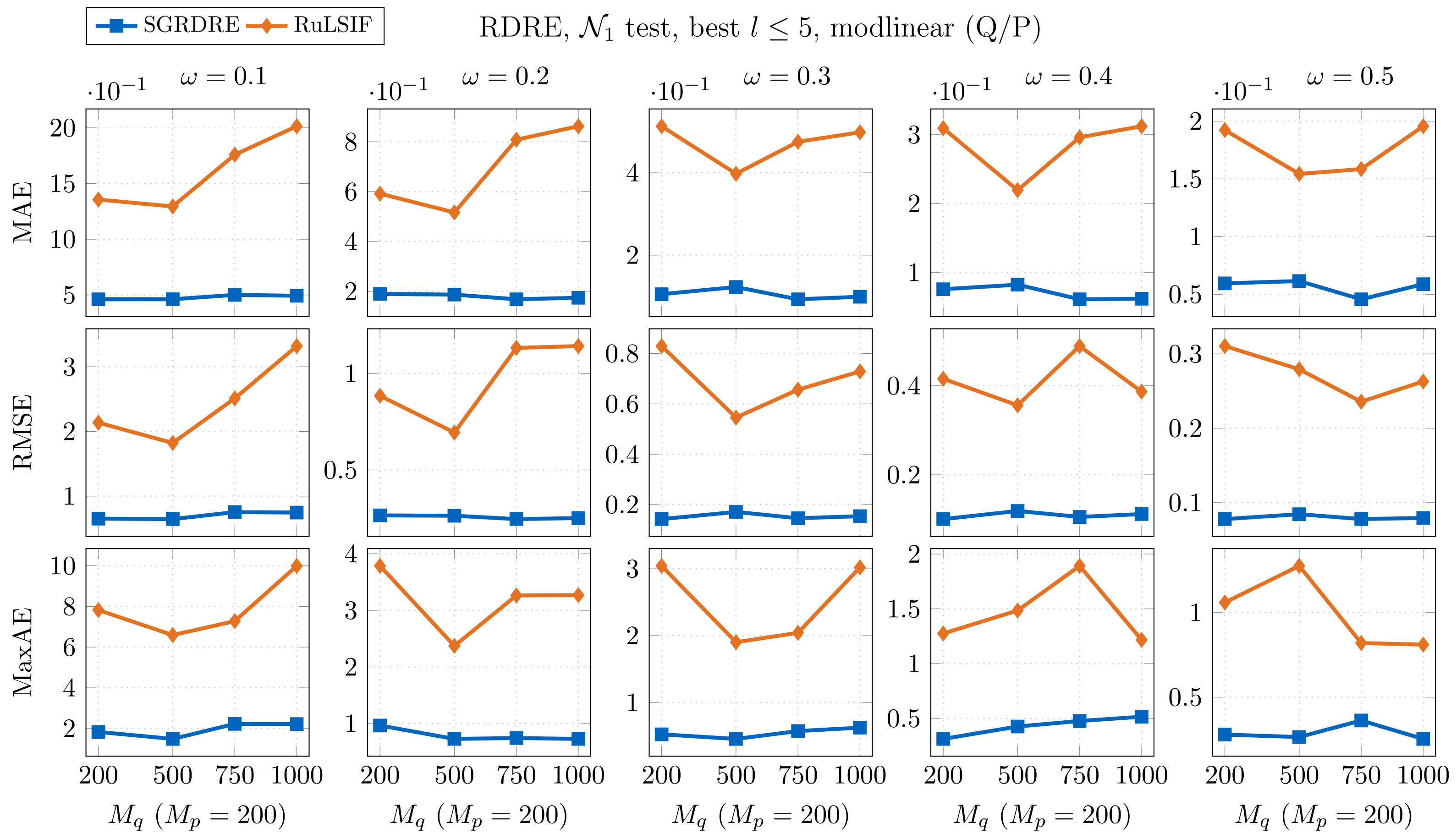


Figure 3.10: 1D RDRE results for varying ω values on normal-distributed input datasets. We now show errors obtained on the inverse ratios (Q/P), where we need to apply modlinear basis functions. As expected, we see across the board better results for the sparse grid method which can recover good estimates also close to the domain border, while the kernel method can only provide estimates at, or close to, the input data points.

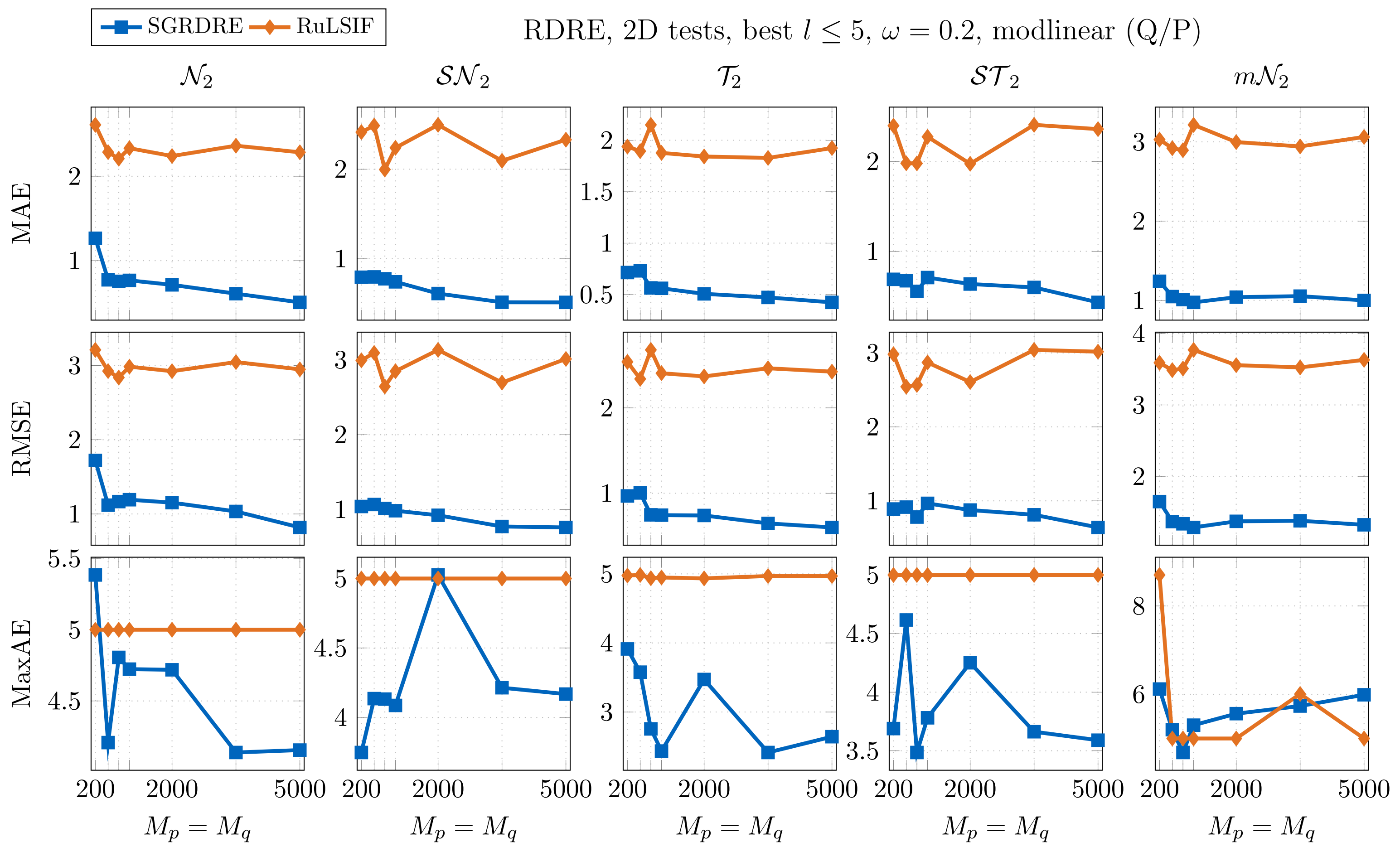


Figure 3.11: 2D RDRE results for $\omega = 0.2$ on various datasets, in the inverse density ratio scenarios (Q/P). Kernel-based methods can only hope to estimate the target function at data points, while the sparse grid-based approach can provide good accuracies over the whole domain. Thus, we obtain across the board better results for SGRDRE.

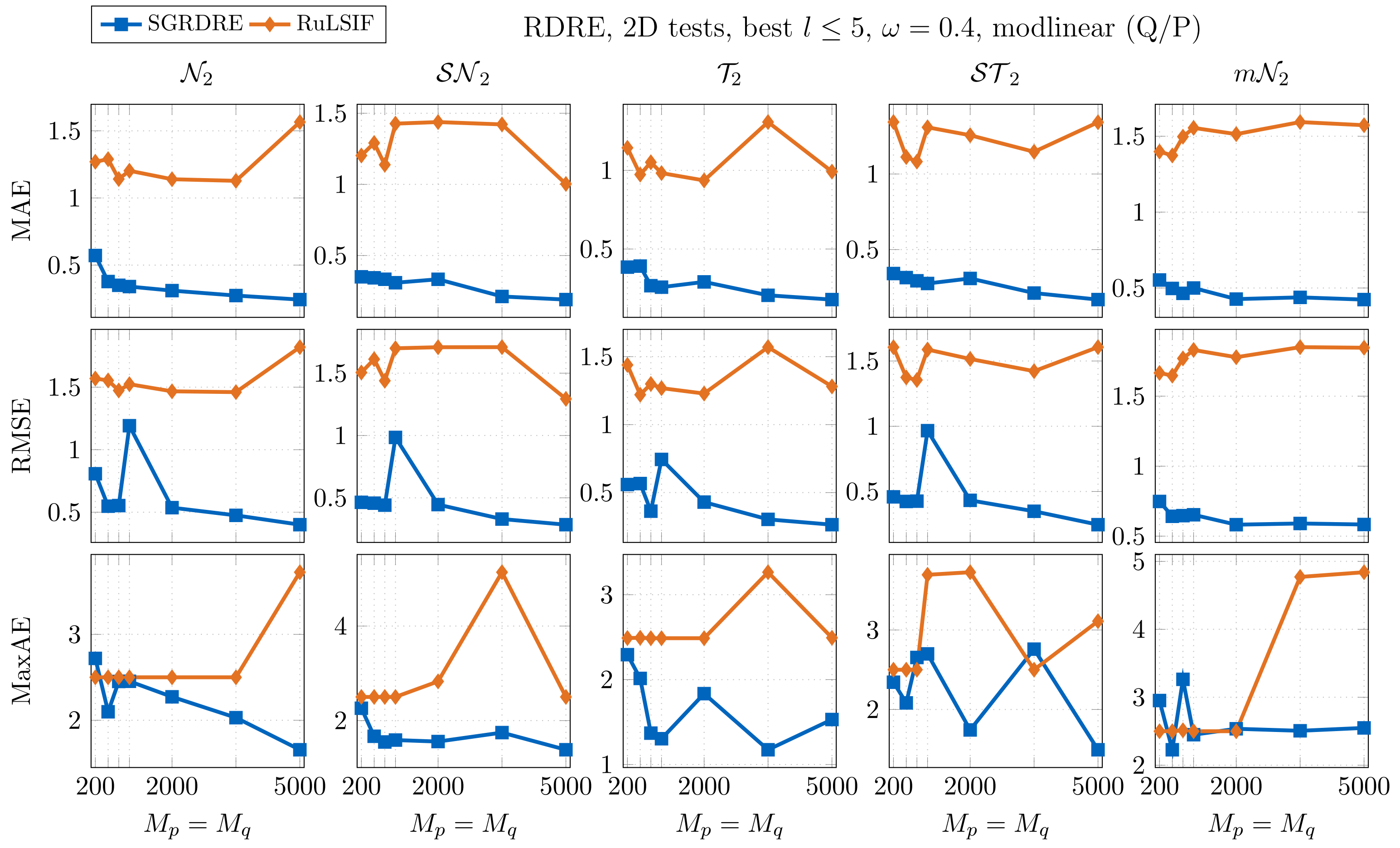


Figure 3.12: 2D RDRE results for $\omega = 0.4$ on various datasets, in the inverse density ratio scenarios (Q/P). The same as for $\omega = 0.2$, the kernel-based method can only estimate the target function at or close to data points. The sparse grid-based approach on the other hand can provide good results in the whole domain, and thus we obtain across the board better results for SGRDRE.

Table 3.4: Parameters of the higher-dimensional RDRE test scenarios.

test name	distribution parameters
$\mathcal{SN}_3$	P: $\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 2,\ -2]^T$ Q: $\boldsymbol{\xi} = [0.4,\ 0.4,\ 0.4]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ -2,\ 2]^T$
$\mathcal{SN}_5$	P: $\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.01,\ 0.015,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 0,\ 2,\ -2,\ 2]^T$ Q: $\boldsymbol{\xi} = [0.4,\ 0.4,\ 0.4,\ 0.4,\ 0.4]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.01,\ 0.015,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 0,\ 2,\ 2,\ -2]^T$

selected representative visual results for some of the test scenarios in Fig. 3.13. Similar visual differences have been obtained for all 2D test cases.

Higher-dimensional results. Having now a sparse grid method for estimating density ratios (even if modified) irrespective of the nature of the input datasets (by countering the possible unboundedness via the ω parameter), we have performed our final tests on higher-dimensional scenarios and, correspondingly, larger datasets. Specifically, we ran tests with sample sizes $M_p = M_q = \{5000, 10000, 20000\}$ drawn from skew-normal distributions of dimensionality 3 and 5, whose parameters can be found in Table 3.4, using a relative density ratio parameter $\omega = 0.1$. For SGRDRE we test both a fixed level 6 grid case, as well as 3 adaptive grids using the surplus, surplus-volume, and surplus-absolute-value adaptivity scores, respectively, using 5 refinement steps with 10/20 refined grid points per step for the 3/5-dimensional datasets, respectively. Again, we look for the best regularization parameter amongst 5 values $\lambda = \{10^{-1}, \ldots, 10^{-5}\}$.

Results for minimal RMSE per test are shown in Fig. 3.14. We observe again the superiority of SGRDRE to RuLSIF, even though now the errors are not computed in the whole domain, but at the data points, which is advantageous for kernel-based methods. Additionally, as opposed to what we saw for SGDDE, the adaptive grids here show on par result to the high level regular grid, meaning that the density ratio method can benefit easier from well-placed grid points. In terms of the best adaptivity strategy, no definitive conclusion could be drawn from our tests, with some differences only seen in the MaxAE values.

Discussion of results. Overall, for both DRE and RDRE, our sparse grid approach obtained on par or better results for the cases where density ratios had bounded, low-tailed distributions, where we could use linear basis functions. How-

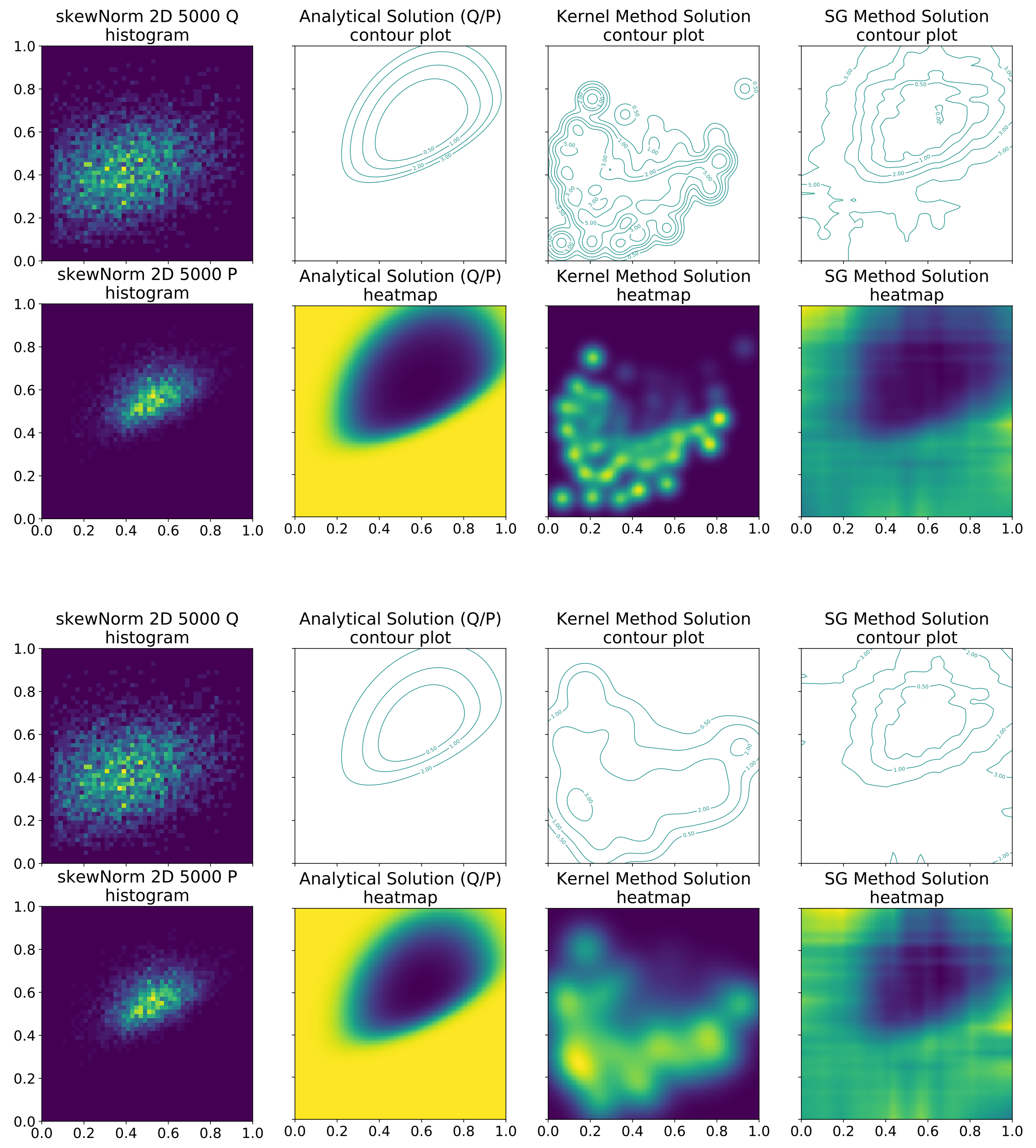


Figure 3.13: Visual inspection of the RDRE results for the $\mathcal{SN}_2$ $\omega = 0.2$ (*top*) and $\omega = 0.4$ test scenarios. We compare the contour plot and heatmap of both RuLSIF and SGRDRE for the best obtained parameters against the analytical solution for the inverse ratio Q/P, for the largest dataset size considered. We observe clearly the inability of the kernel-based approach to provide a good representation in the whole domain. Additionally, overfitting plagued the RuLSIF method for most of our test scenarios, with the cross-validation-based hyperparameter selection procedure proposed in literature not being able to consistently provide reasonable parameter values.

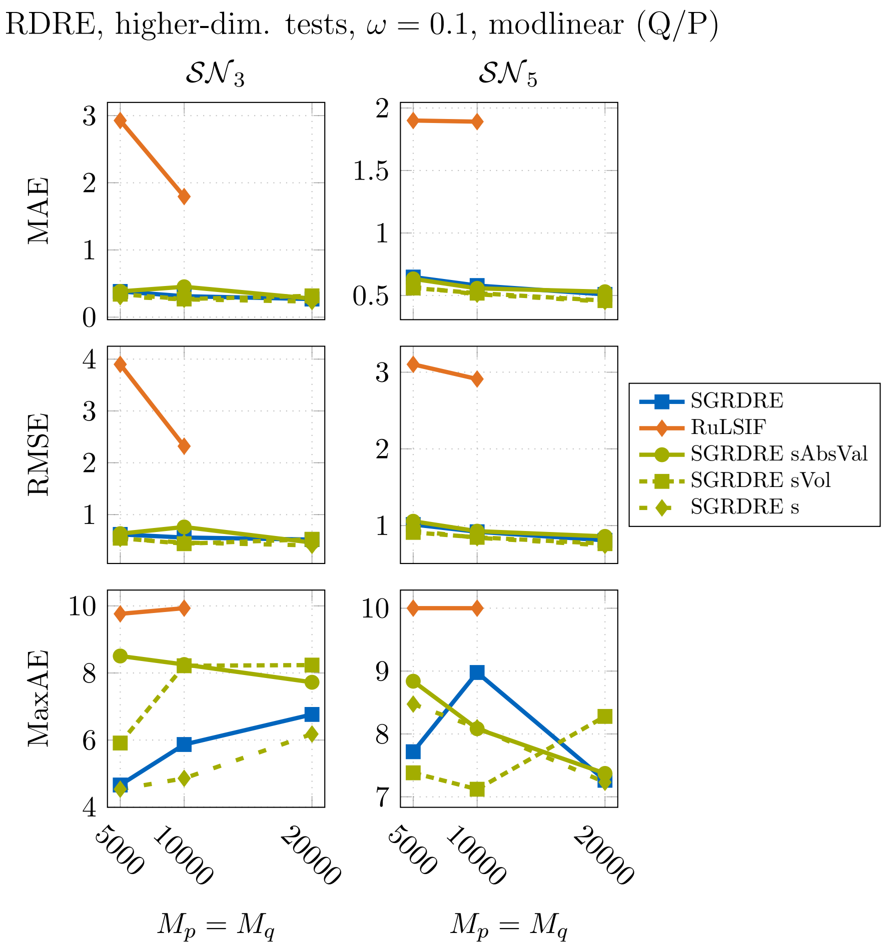


Figure 3.14: Higher-dimensional RDRE tests on skew-normal-distributed datasets, estimating the relative inverse ratio (Q/P) for $\omega = 0.1$. Evaluations are computed here at the input data points instead of an equidistant evaluation grid covering the whole domain. We observe that SGRDRE consistently outperforms RuLSIF, on both higher level regular grids and on adaptive grids. Runs on the largest datasets were skipped for RuLSIF due to significantly larger times to solution.

ever the best results were obtained on otherwise unfavorable density ratio distributions, where, irrespective of the boundedness-enforcing parameter $\omega > 0$, our sparse grid method can leverage the benefits of being able to properly cover the computational domain. In these cases the kernel method can only hope to estimate well at the input data points, however our higher-dimensional tests showed that even in this context the sparse grid method can obtain lower errors across the board.

Using non-optimal Python codes in terms of parallel performance for both RuLSIF and SGRDRE makes for an unfair comparison of performance in terms of runtimes, therefore we have refrained from presenting results in this regard. Running both methods on single threads however showed that RuLSIF is up to 1.5 orders of magnitude slower than SGRDRE in computing the solution, even when accounting for the time spent selecting the correct parameters, and up to 3 orders of magnitude slower in evaluating the estimators to compute the error metrics.

3.5 Density Derivative Estimation

Terminology and notations. In the case of density derivative estimation, the term "derivative" is used loosely to denote any particular *partial derivative of the probability density* that we want to approximate numerically. Such a derivative will be denoted by

$$\partial^{(\boldsymbol{j})} p(\boldsymbol{x}) = \frac{\partial^{|\boldsymbol{j}|}}{\partial x_1^{j_1} \partial x_2^{j_2} \dots \partial x_d^{j_d}} p(\boldsymbol{x}), \tag{3.27}$$

where $\boldsymbol{j} = (j_1, j_2, \dots, j_d)$ is a vector encoding the (up to d-dimensional and possibly mixed) partial derivative, and $|\boldsymbol{j}| = \sum_{k=1}^{d} j_k$ is the order of this partial derivative. So, for example, the fifth order mixed partial derivative $\frac{\partial^5}{\partial x_1^3 \partial x_3^2}$ applied to a 3-dimensional function will be encoded by the vector $\boldsymbol{j} = (3, 0, 2)$.

This way of defining partial derivatives, adopted from [72], has a quite obvious drawback: it is implicitly symmetric. What we mean by this is that it assumes the commutativity of all the mixed partial derivatives of lower or equal order as the one we require. So, for example, even the second order mixed partial derivative $\frac{\partial^2}{\partial x_2 \partial x_1}$ has no such vector encoding. While the mathematical derivations that follow are not strictly restricted to this encoding, as they work with any kind of partial derivatives, and while also other ways of encoding all possible partial derivatives of a certain order have been proposed (e.g., [42]), we argue that our notation is indeed sufficient. This stems from the fact that, for our approach of deriving our linear system, irrespective of whether the function we partially derive is continuously differentiable of high enough order or not (for the partial derivatives to all commute), the approximations we obtain from our numerical estimator are by design smooth enough: the basis functions used in both grid-based and kernel-based approaches are either infinitely continuously differentiable (e.g., polynomial basis or Gaussian functions), or can be chosen to be enough continuously differentiable everywhere (e.g., B-splines). Therefore the commutativity of the higher order partial derivatives is intrinsic to the mathematical treatment using sparse grids.

Problem statement. With the newly introduced notation in Eq. (3.27), the problem of *density derivative estimation* (DDerivE) can be formulated as:

> Given a set of independent and identically distributed samples in $\Omega \subseteq \mathbb{R}^d$, $\mathcal{S}_p = \{\boldsymbol{x}_i^p\}_{i=1}^{M_p}$ with density $p(\boldsymbol{x})$, estimate the value of $\partial^{(\boldsymbol{j})}p(\boldsymbol{x}) = \frac{\partial^{|\boldsymbol{j}|}}{\partial x_1^{j_1} \partial x_2^{j_2} \dots\ partial x_d^{j_d}} p(\boldsymbol{x})$, $\boldsymbol{j} = (j_1, j_2, \dots, j_d)$, $|\boldsymbol{j}| = \sum_{k=1}^d j_k$, under the assumption that $\partial^{(\boldsymbol{j})}p(\boldsymbol{x})$ is well-defined.

Existing approaches. Two relatively similar approaches have been developed to tackle this task using kernel-based methods. The first algorithm, which we will also compare our own sparse grids results against, is the one of Sasaki et al., who introduce the *mean integrated squared error for derivatives* (MISED) method [72]. Their idea is to find an analytical formulation for computing the weights of a Gaussian kernel model that minimizes a regularized least squares loss function between the model and the actual partial derivative of interest. The second algorithm is a standard kernel density derivative estimation, on top of which Chacón and Duong propose different strategies to find the best bandwidths such that some mean squared error between all partial derivatives of the kernel estimator and of the real density is minimized [42]. Recently another approach, using a shrinkage type kernel estimator, was developed by Mahmoudi et al. [50].

3.5.1 Sparse Grid Density Derivative Estimation

Mathematical derivation. We will start our derivation of the linear system for our *sparse grid density derivative estimation* (SGDDerivE) algorithm by looking for a function in a suitable space $r_{\boldsymbol{j}}(\boldsymbol{x}) \in V$ that minimizes the same kind of regularized least squares cost function as in [72]:

$$J(r_{\boldsymbol{j}}) = \int_\Omega \left[r_{\boldsymbol{j}}(\boldsymbol{x}) - \partial^{(\boldsymbol{j})}p(\boldsymbol{x}) \right]^2 \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda r_{\boldsymbol{j}}\|_{L^2}^2. \tag{3.28}$$

The corresponding variational equation is naturally given by

$$\int_\Omega s(\boldsymbol{x})\, r_{\boldsymbol{j}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int_\Omega s(\boldsymbol{x})\, \partial^{(\boldsymbol{j})}p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r_{\boldsymbol{j}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0,\ \forall s \in V. \tag{3.29}$$

This time, different from the previous variational formulations, we need to further process this equation. The second term is not directly accessible. However, the same as in [72], we can use repeated integration by parts to solve this issue, such that the second term becomes

$$\int_{\Omega} s(\boldsymbol{x})\, \partial^{(\boldsymbol{j})} p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = (-1)^{|\boldsymbol{j}|} \int_{\Omega} \partial^{(\boldsymbol{j})} s(\boldsymbol{x})\, p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x}, \tag{3.30}$$

under the assumption (which we will call the *boundary smoothness condition*) that products of the form $\partial^{(\boldsymbol{j}')} s(\boldsymbol{x})\ \partial^{(\boldsymbol{j}'')} p(\boldsymbol{x})$ cancel out on the boundary $\partial\Omega$ for any pair of derivative encoding vectors $(\boldsymbol{j}', \boldsymbol{j}'')$ that satisfy $\boldsymbol{j}', \boldsymbol{j}'' < \boldsymbol{j},\ \boldsymbol{j}' + \boldsymbol{j}'' = \boldsymbol{j}$. Here, the operations on the encoding vectors are taken component-wise, with the inequality being considered in the sense that it applies strictly in at least one component.

This is, at first sight, a quite restrictive assumption, however, as we will discuss shortly, we will see that this is not the case. Therefore, with this condition in place, we can obtain the equivalent variational equation

$$\int_{\Omega} s(\boldsymbol{x})\, r_{\boldsymbol{j}}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} - (-1)^{|\boldsymbol{j}|} \int_{\Omega} \partial^{(\boldsymbol{j})} s(\boldsymbol{x})\, p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} + \lambda \int_{\Omega} \Lambda s(\boldsymbol{x})\, \Lambda r_{\boldsymbol{j}}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = 0,\ \forall s \in V. \tag{3.31}$$

Using the estimates from samples of Eq. (2.35) and moving the second term to the right-hand side, we get

$$\int_{\Omega} s(\boldsymbol{x})\, r_{\boldsymbol{j}}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} + \lambda \int_{\Omega} \Lambda s(\boldsymbol{x})\, \Lambda r_{\boldsymbol{j}}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = \frac{(-1)^{|\boldsymbol{j}|}}{M_p} \sum_{i=1}^{M_p} \partial^{(\boldsymbol{j})} s(\boldsymbol{x}_i^p),\ \forall s \in V. \tag{3.32}$$

Searching for the sparse grid solution $r_{\boldsymbol{j}}(\boldsymbol{x}) := \sum_{k=1}^{N} \alpha_k^{(\boldsymbol{j})} \varphi_k(\boldsymbol{x})$ in the space $V = \mathcal{V}_N$ with $s(\boldsymbol{x}) := \varphi_l(\boldsymbol{x})$ in the Ritz-Galerkin projection, we finally obtain

$$\begin{aligned} \sum_{k=1}^{N} \alpha_k^{(\boldsymbol{j})} \int_{\Omega} \varphi_k(\boldsymbol{x})\, \varphi_l(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} + \lambda \sum_{k=1}^{N} \alpha_k^{(\boldsymbol{j})} \int_{\Omega} \Lambda\varphi_k(\boldsymbol{x})\, \Lambda\varphi_l(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} & \\ = \frac{(-1)^{|\boldsymbol{j}|}}{M_p} \sum_{i=1}^{M_p} \partial^{(\boldsymbol{j})} \varphi_l(\boldsymbol{x}_i^p),\ l = 1, \ldots, N, & \end{aligned} \tag{3.33}$$

or, in matrix-vector form,

$$(\boldsymbol{R} + \lambda \boldsymbol{C})\, \boldsymbol{\alpha}^{(\boldsymbol{j})} = \frac{(-1)^{|\boldsymbol{j}|}}{M_p} \partial^{(\boldsymbol{j})} \boldsymbol{B}^p \boldsymbol{e}^p =: (-1)^{|\boldsymbol{j}|} \partial^{(\boldsymbol{j})} \boldsymbol{b}^p. \tag{3.34}$$

By $\partial^{(\boldsymbol{j})} \boldsymbol{B}^p$ we denote the matrix of evaluations of the partial derivatives of basis functions at sample points $\mathcal{S}_p$, i.e., $\partial^{(\boldsymbol{j})} \boldsymbol{B}_{l,i}^p = \partial^{(\boldsymbol{j})} \varphi_l(\boldsymbol{x}_i^p)$.

While the method we just introduced can be applied for any $\boldsymbol{j}$, for the purpose of our thesis we only consider the particular case of first order derivative estimation, i.e., $|\boldsymbol{j}| = 1$, meaning we estimate elements of the gradient $\nabla p(\boldsymbol{x})$ by solving the system

$$(\boldsymbol{R} + \lambda \boldsymbol{C})\, \boldsymbol{\alpha}^{(k)} = \frac{-1}{M_p} \partial_k \boldsymbol{B}^p \boldsymbol{e}^p =: -\partial_k \boldsymbol{b}^p, \tag{3.35}$$

for a given dimension (or direction) $k \in \{1, \ldots, d\}$, where $\partial_k \boldsymbol{B}^p_{l,i} = \partial_k \varphi_l(\boldsymbol{x}^p_i)$.

The boundary smoothness condition. As stated in the mathematical derivation, an essential assumption is that products of the form $\partial^{(\boldsymbol{j}')} s(\boldsymbol{x})\ \partial^{(\boldsymbol{j}'')} p(\boldsymbol{x})$ cancel out on the boundary $\partial\Omega$ of our domain for any pair $(\boldsymbol{j}', \boldsymbol{j}'')$ that satisfy $\boldsymbol{j}', \boldsymbol{j}'' < \boldsymbol{j},\ \boldsymbol{j}' + \boldsymbol{j}'' = \boldsymbol{j}$. In the derivation of Sasaki [72], for the kernel-based method, the underlying assumption is that the domain Ω is the whole real space $\mathbb{R}^d$, where both the kernel functions, usually Gaussian in nature, as well as in general the distribution of any well-behaved probability density and its partial derivatives fade to zero at infinity. In our case however, where the domain is restricted to the compact hypercube $[0, 1]^d$, we cannot guarantee anymore that our basis functions fade at the boundary, even on grids with no boundary points. (General basis functions are more likely than not to be non-zero at the boundary, for example every B-spline-based basis.) Instead we have to impose a restriction on the data distribution p and its derivatives instead, i.e., $\partial^{(\boldsymbol{j}'')} p(\boldsymbol{x}) = 0, \forall x \in \partial\Omega, \forall \boldsymbol{j}'' < \boldsymbol{j}$. Of course, mathematically most well-known data distributions are defined in the whole multi-dimensional real space $\mathbb{R}^d$, however practically we can scale our input dataset such that it lives strictly inside the sparse grid domain and its distribution tail has negligible values, and thus influence, at the boundary.

Properties of the linear system. The linear system we obtain in density derivative estimation is extremely similar to the one in SGDE, with the only changes appearing in the right-hand side term. Indeed, for the edge case $\boldsymbol{j} = \boldsymbol{0}$, we would recover the linear system of Eq. (2.39), under the understanding that a zeroth-order derivative leaves a function unchanged. Therefore, the SGDDerivE algorithm imports all the properties, and thus also the benefits, of the SGDE algorithm.

Accuracy considerations. Similarly to SGDE and SGDDE, under the same assumptions of a solution $r \in \mathcal{V}_N \subset H^2_{mix}$ to the linear system Eq. (3.34) for a density function p with $\partial^{\boldsymbol{j}} p \in H^2_{mix}$, but also with the additional boundary smoothness condition used in the mathematical derivation, we can show that the obtained sparse grid method is consistent:

$$\Pr\left(\lim_{M_p,\, N \,\to\, \infty} \left\| r - \partial^{(\boldsymbol{j})} p \right\|^2_{L^2} = 0 \right) = 1. \tag{3.36}$$

This simply asserts that as we approach infinitely many samples from the input distributions on a sparse grid with a number of grid points approaching infinity our numerical solution approaches the exact density derivative.

The proof steps are quite similar to those for SGDDE. If $\tilde{r} \in \mathcal{V}_N$ is the sparse grid interpolant of the target density derivative $\partial^{(\boldsymbol{j})}p$, we can write the error as

$$\left\|r - \partial^{(\boldsymbol{j})}p\right\|_{L^2}^2 = \left\langle r - \partial^{(\boldsymbol{j})}p, r - \tilde{r}\right\rangle_{L^2} + \left\langle r - \partial^{(\boldsymbol{j})}p, \tilde{r} - \partial^{(\boldsymbol{j})}p\right\rangle_{L^2}. \tag{3.37}$$

For the first term of Eq. (3.37), as we look for the behavior at the limit $M_p \to \infty$, we can assume no overfitting takes place, so $\lambda = 0$. Furthermore, as both r and $\tilde{r}$ are functions represented on sparse grids, we can denote $s := r - \tilde{r} \in \mathcal{V}_N$ and with Eq. (3.32) we obtain:

$$\begin{aligned}\left\langle r - \partial^{(\boldsymbol{j})}p, r - \tilde{r}\right\rangle_{L^2} &= \left\langle r - \partial^{(\boldsymbol{j})}p, s\right\rangle_{L^2}\\ &= \langle r, s\rangle_{L^2} - \left\langle \partial^{(\boldsymbol{j})}p, s\right\rangle_{L^2}\\ &= \langle r, s\rangle_{L^2} - (-1)^{|\boldsymbol{j}|}\left\langle p, \partial^{(\boldsymbol{j})}s\right\rangle_{L^2}\\ &= \frac{(-1)^{|\boldsymbol{j}|}}{M_p}\sum_{i=1}^{M_p}\partial^{(\boldsymbol{j})}s(\boldsymbol{x}_i^p) - (-1)^{|\boldsymbol{j}|}\mathbb{E}_p(\partial^{(\boldsymbol{j})}s(\boldsymbol{x})),\end{aligned} \tag{3.38}$$

where the boundary smoothness condition in effect "switched" the derivative from the density p to the sparse grid function s.

For the second term of Eq. (3.37) we have the upper bound

$$\begin{aligned}\left|\left\langle r - \partial^{(\boldsymbol{j})}p, \tilde{r} - \partial^{(\boldsymbol{j})}p\right\rangle_{L^2}\right| &\le \left\|r - \partial^{(\boldsymbol{j})}p\right\|_{L^2} \cdot \left\|\tilde{r} - \partial^{(\boldsymbol{j})}p\right\|_{L^2}\\ &\le \frac{1}{4}\left\|r - \partial^{(\boldsymbol{j})}p\right\|_{L^2}^2 + \left\|\tilde{r} - \partial^{(\boldsymbol{j})}p\right\|_{L^2}^2\\ &\le \frac{1}{4}\left\|r - \partial^{(\boldsymbol{j})}p\right\|_{L^2}^2 + \mathcal{O}\left(2^{-(n+1)l} \cdot l^{d-1}\right),\end{aligned} \tag{3.39}$$

where for the first two steps we have again used the Cauchy–Bunyakovsky-Schwarz inequality and the modified Young's inequality (Eq. (3.11)), respectively. For the third step, as we will use B-splines as our basis functions to match the required degree of smoothness for the derivatives, we use the more general sparse grid error bound for spline basis as derived by Sickel and Ullrich [74], where n is the degree of the splines. Note that this is also consistent with previously introduced error bounds, as for degree $n = 1$ one recovers the piecewise-linear functions and thus also their interpolation convergence behavior on sparse grids.

Putting the results of Eqs. (3.38) and (3.39) together into Eq. (3.37), we finally obtain after a small restructuring:

$$\begin{aligned}\left\|r - \partial^{(\boldsymbol{j})}p\right\|_{L^2}^2 \le\ &\frac{4}{3}\,\mathcal{O}\left(2^{-(n+1)l} \cdot l^{d-1}\right) +\\ &\frac{4}{3}(-1)^{|\boldsymbol{j}|}\left(\frac{1}{M_p}\sum_{i=1}^{M_p}\partial^{(\boldsymbol{j})}s(\boldsymbol{x}_i^p) - \mathbb{E}_p(\partial^{(\boldsymbol{j})}s(\boldsymbol{x}))\right).\end{aligned} \tag{3.40}$$

For $N \to \infty$ the sparse grid interpolation error tends to zero, irrespective of the spline degree n. The second term of Eq. (3.40) also goes to zero in a probabilistic sense, i.e., $\Pr\left(\lim_{M_p\to\infty} \frac{1}{M_p}\sum_{i=1}^{M_p} \partial^{(\boldsymbol{j})} s(\boldsymbol{x}_i^p) = E_p(\partial^{(\boldsymbol{j})} s(\boldsymbol{x}))\right) = 1$, as the samples, even though distributed according to density p, will fill the whole domain of $\partial^{(\boldsymbol{j})}p$ as well in the limit. This thus completes the proof of the consistency asserted by Eq. (3.36).

Implementation details. With only a single input dataset, density derivative estimation is indeed the closest in terms of implementation to the original density estimation. The main differences come in the handling of the computation of the right-hand side. The more usual sparse grid bases, like the hat functions or the polynomial basis, are not continuously differentiable, therefore their use could lead to numerical issues at the points of discontinuity. B-splines however do not have that issue, as a correctly chosen degree will ensure the required order of continuous differentiability. Thankfully, the SG++ library already allows the handling of B-spline variants, as well as their partial derivatives, with fast, analytical formulas for their evaluation.

The batch processing concept also applies to SGDDerivE quite directly. Using the same notation as for SGDDE, the update rule for the right-hand side after k batches will be, in the generic case:

$$(-1)^{|\boldsymbol{j}|}\partial^{(\boldsymbol{j})}\boldsymbol{b}^{p(k)} = \frac{M_p^{(k-1)}}{M_p^{(k)}}(-1)^{|\boldsymbol{j}|}\partial^{(\boldsymbol{j})}\boldsymbol{b}^{p(k-1)} + \frac{\tilde{M}_p^{(k)}}{M_p^{(k)}}(-1)^{|\boldsymbol{j}|}\partial^{(\boldsymbol{j})}\tilde{\boldsymbol{b}}^{p(k)}. \tag{3.41}$$

Similar to SGDDE, the treatment of concept drift for SGDDerivE was also not implemented in our work, but its mathematical derivation can be found in Appendix A.2.

Performance considerations. The implementation of density derivative estimation carries over all the performance benefits available for SGDE, therefore both parallelization across batches and the ScaLAPACK variants of the system matrix decompositions were easily integrated into the SGDDerivE implementation.

3.5.2 Numerical results

Goals. As with the previously introduced new estimators, for SGDDerivE we were interested is assessing its numerical properties, including strong points as well as possible caveats, i.e., aspects which are of practical interest and most of which might not be directly evident just from the mathematical derivations. To this purpose we have ran simulations of artificial datasets sampled from various single and mixed distributions, which offered useful, controlled scenarios in which to test out various parameters for our sparse grid-based method. Additionally, we have compared all results to an implementation of the MISED kernel-based approach.

Quality criteria. We have again considered useful to apply here as well the three previously introduced metrics, i.e., MAE, RMSE, and MaxAE, which can provide a varied overview on the accuracy of our estimator. While all three values are important in each test case, we use RMSE as the deciding factor on the best results for a set of problem and solution parameters.

The testing pipeline. For the problem of density derivative estimation only one input dataset is required, therefore we have created a distinctive, yet very similar in functionality, testing pipeline based on the one used for the problems of DDE and (R)DRE.

This code was once more written in Python, with the same testing steps being carried out as described for SGDDE: generating input datasets by sampling from some well-known data distributions (see Appendix B), generating the corresponding configuration files for the SG++ data mining pipeline, running the sparse grid code and the corresponding kernel-based estimator, then finally compute errors, generate plots (to aid in interpreting the results, where viable), and save everything to file for any later post-processing.

Testing parameters. While for the two-dataset complex density estimators mentioned so far we were able to use existing Python implementations, for density derivative estimation this was not possible. However, a closely related, in terms of programming language, MATLAB implementation[6] of the MISED algorithm as described in [72] was available. We have therefore translated this code into Python in order for proper integration into the rest of our testing pipeline.

The implementation of MISED also contains its own cross-validation process for selecting hyperparameters. For our simulations we ran 5-fold cross-validation on a parameter space of 10 kernel widths $\sigma_K = \left\{10^{-3}, 10^{-2.(6)}, \ldots, 10^{-0.(3)}, 10^{0}\right\}$ and 5 regularization parameter values $\lambda_K = \{10^{-4}, 10^{-3}, \ldots, 10^{0}\}$. These values were chosen to reasonably cover all test cases in our unit hypercube computational domain, with the density of the parameter search grid chosen to balance out the need to obtain good estimations with the extra computational effort of the cross-validation process.

For the sparse grid runs we will specify for each test scenario the parameters we will use. What is valid for all simulations is the use of B-spline basis functions. Specifically, we use the modified not-a-knot B-spline basis of, which, on the one hand, satisfies the hierarchical splitting condition to allow exact approximations on sparse grids (as described in Section 2.1.4) and, on the other hand, keeps the grid size reasonable by excluding boundary points. For these splines we choose the minimal degree that can deliver continuous first order derivatives, as required by the DDerivE task, which is 3.

[6]MATLAB code written by Hiroaki Sasaki, available at `https://sites.google.com/site/hworksites/home/software/lsdrf`

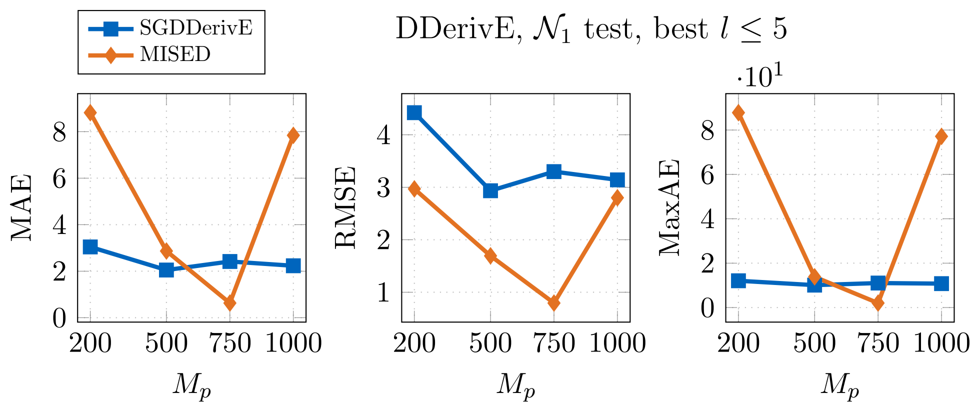


Figure 3.15: 1D DderivE results on normal-distributed datasets of various sizes. We observe in practice less robustness from the MISED cross-validation process, with overfitting influencing the results. However, despite these issues, MISED provided better accuracies than SGDDerivE in our tests.

1D results. The first tests we have performed were done on one-dimensional data, meant to provide a first validation of the approach. We considered as input datasets $\mathcal{S}_p \in \mathcal{N}_1\,(\mu = 0.65, \Sigma = 0.01)$ of sizes $M_p = \{200, 500, 750, 1000\}$, testing thus relatively small datasets, where the kernel-based approach is expected to work well and our sparse grid method is more challenged. We ran SGDDerivE with 4 regularization parameter values $\lambda = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ and on 3 regular grid sizes $l = \{3, 4, 5\}$, selecting as best the parameter combination that results in the minimal RMSE.

The results for the one-dimensional tests are shown in Fig. 3.15. Multiple observations can be made. Firstly, we again reveal some issues with the simple cross-validation process of the kernel-based method, MISED not always selecting the optimal choice of hyperparameters. This means that a better hyperparameter tuning process might be needed to increase the robustness of the method. Secondly, despite possible shortcomings, MISED significantly performed better in these tests. Lastly, although for brevity not explicitly shown here, in practice we see that higher grid levels already show a tendency of SGDDerivE to overfit, whose effects are mildly kept in check by the fact that we use degree 3 B-splines.

2D results. Interesting test cases with respect to sparse grids begin with two dimensions, where we can begin to see the reduction in grid points brought by this numerical method. Therefore we continued our testing of SGDDerivE with 2D scenarios on datasets sampled from single and mixed distributions, whose parameters are listed in Table 3.5. Based on the 1D results, we extended the number of dataset sizes we tested in these scenarios in order to see if, and how, the accuracies improve for our sparse grid method with more data points. To this purpose we ran here 9 different dataset sizes from 200 to 10000, on 3 different-sized regular grids, with

Table 3.5: Parameters of the 2D DDerivE test scenarios.

test name	distrib. type(s)	distribution parameters
$\mathcal{N}_2$	$\mathcal{N}$	$\boldsymbol{\mu} = [0.4,\ 0.5]^T$, $\boldsymbol{\Sigma} = [[0.02,\ 0.01];\ [0.01,\ 0.04]]$
$\mathcal{SN}_2$	$\mathcal{SN}$	$\boldsymbol{\xi} = [0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = [[0.08,\ 0.02];\ [0.02,\ 0.06]]$, $\boldsymbol{\alpha} = [5,\ -2]^T$
$\mathcal{T}_2$	$\mathcal{T}$	$\boldsymbol{\mu} = [0.4,\ 0.5]^T$, $\boldsymbol{\Sigma} = [[0.02,\ 0.01];\ [0.01,\ 0.04]]$, $\nu = 10$
$\mathcal{ST}_2$	$\mathcal{ST}$	$\boldsymbol{\xi} = [0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = [[0.035,\ 0.0175];\ [0.0175,\ 0.0525]]$, $\boldsymbol{\alpha} = [3,\ -2]^T$, $\nu = 5$
$m\mathcal{N}_2$	$0.4 \cdot \mathcal{N}$ $+0.3 \cdot \mathcal{N}$ $+0.3 \cdot \mathcal{N}$	$\begin{cases} \boldsymbol{\mu} = [0.35,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.01,\ 0.0075];\ [0.0075,\ 0.02]] \\ \boldsymbol{\mu} = [0.5,\ 0.5]^T,\ \boldsymbol{\Sigma} = [[0.03,\ 0];\ [0,\ 0.01]] \\ \boldsymbol{\mu} = [0.6,\ 0.4]^T,\ \boldsymbol{\Sigma} = [[0.01,\ 0.0075];\ [0.0075,\ 0.02]] \end{cases}$
$m\mathcal{ST}_2$	$0.5 \cdot \mathcal{ST}$ $+0.5 \cdot \mathcal{ST}$	$\begin{cases} \boldsymbol{\xi} = [0.45,\ 0.55]^T,\ \boldsymbol{\Omega} = [[0.025,\ 0];\ [0,\ 0.01]] \\ \quad \boldsymbol{\alpha} = [0,\ 3]^T,\ \nu = 3 \\ \boldsymbol{\xi} = [0.55,\ 0.45]^T,\ \boldsymbol{\Omega} = [[0.01,\ 0];\ [0,\ 0.025]] \\ \quad \boldsymbol{\alpha} = [3,\ 0]^T,\ \nu = 3 \end{cases}$

$l = \{3, 4, 5\}$ for datasets up to size 3500 and $l = \{4, 5, 6\}$ for the larger datasets, to make sure we don't saturate too fast neither the number of grid points.

Results for minimal RMSE per test case are shown in Fig. 3.16, for both dimensions along which we compute the density derivative. Overall we see that our approach was not as successful as the previously showed complex estimators, even though we see that convergence rates are similar overall. We observe that in general MISED is less affected by the choice of derivative dimension, with accuracies more closely matched together than the sparse grid approach. In practice we have observed also visually that SGDDerivE in its current formulation is prone to produce numerical artifacts in the domain, in the direction perpendicular to the derivative dimension chosen. This can be seen in Fig. 3.17 for a representative test scenario. It is not directly clear how related this is to our choice of basis functions, or if it is more related to the linear system itself and the cost function chosen, but one point of further tests could be the use of boundary points, as well as other types of B-spline-based bases.

Lastly, we have to note that for several test cases MISED could not produce reasonable results, even after a couple reruns, due to the 5-fold cross-validation process not being able to combat the inherent overfitting tendency of the method. This happened mostly for the small datasets and more accentuated for mixed distributions. Where errors were at least an order of magnitude over the expected values we opted in the graphs here to simply ignore those values, which would otherwise make the plots not understandable.

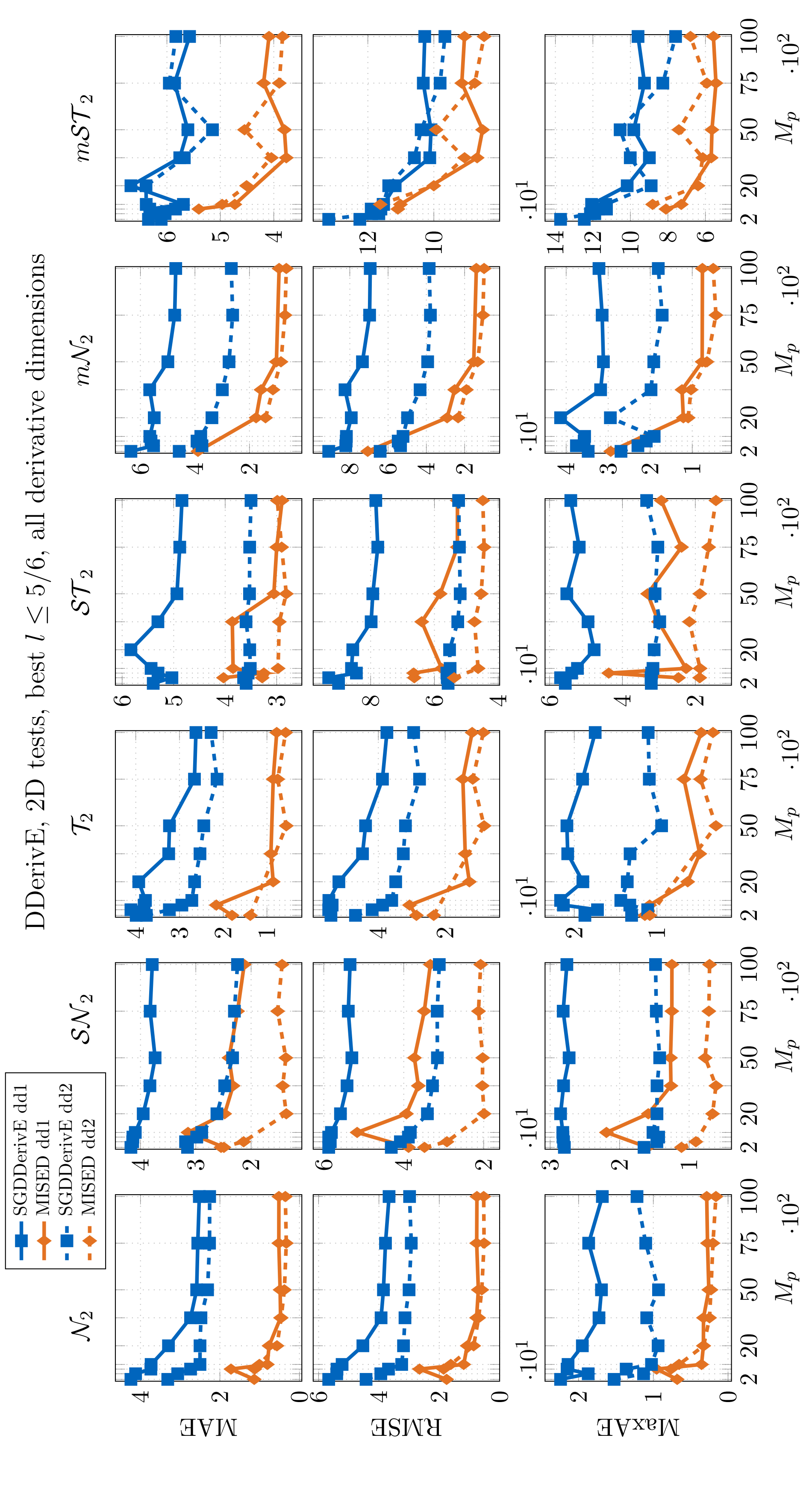


Figure 3.16: 2D DDerivE results on various datasets, for both possible derivative dimension choices. Across the board MISED performed better than SGDDerivE, which suffered from numerical artifacts, even on low level grids. To note as well that for most of the scenarios the lowest errors were obtained on these low level grids, seen here in the poor convergence rate for the largest dataset sizes, where the minimal tested grid level was increased from 3 to 4.

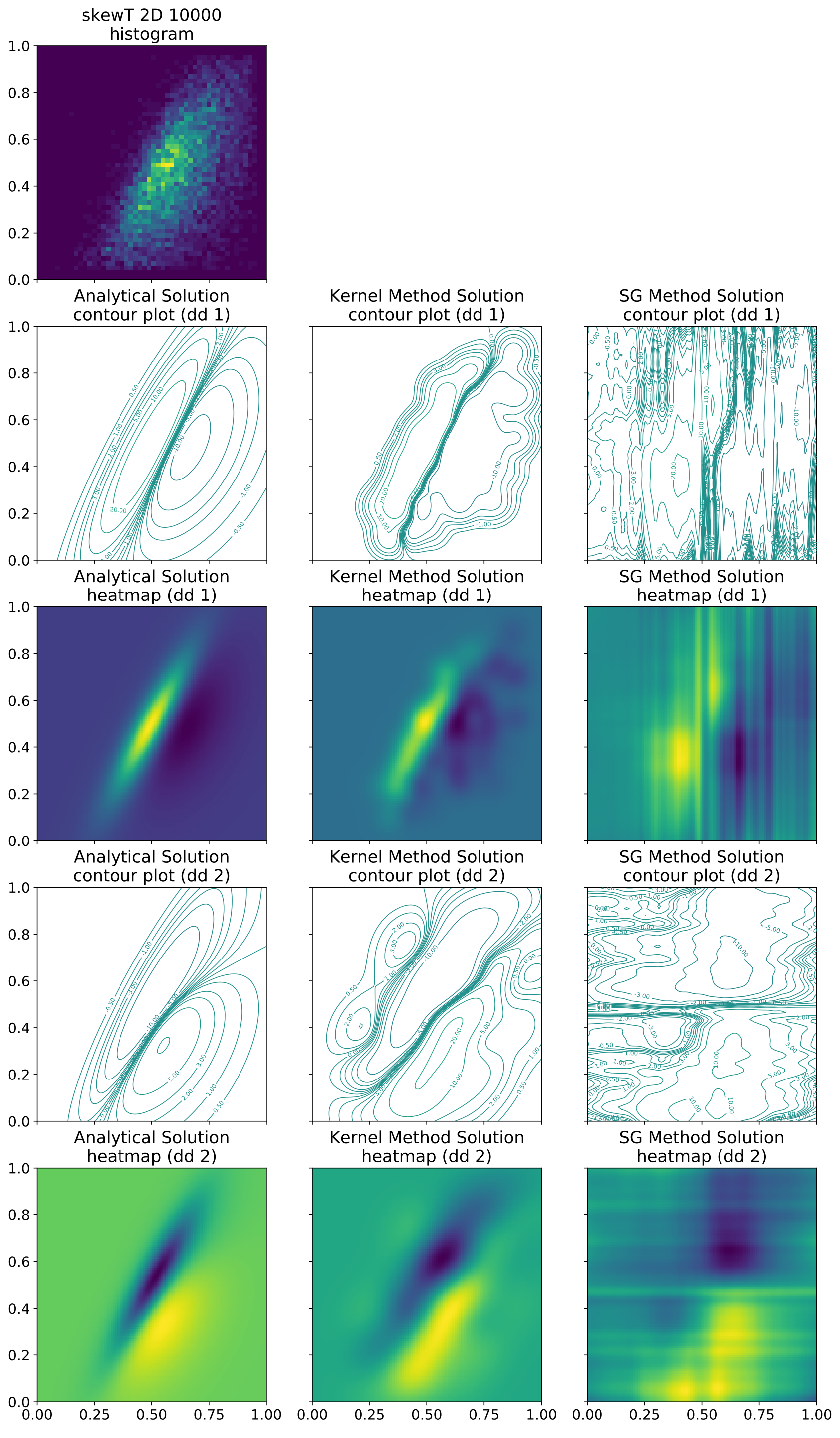


Figure 3.17: Visual inspection of the DDerivE results for the $\mathcal{ST}_2$ test scenario. Both derivative dimensions are shown. We compare the contour plot and heatmap of both MISED and SGDDerivE for the best obtained parameters against the analytical solution of the target density derivative, for the largest dataset size considered. We can observe clearly the inclination of the sparse grid solution to align itself with the corresponding direction of the derivative dimension, resulting in an also visibly poor approximation. On the other side, although suffering from a subpar parameter selection procedure, MISED produces useful and relatively accurate results.

Table 3.6: Parameters of the higher-dimensional DDerivE test scenarios.

test name	distribution parameters	derivative dimensions
$\mathcal{SN}_3$	$\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 2,\ -2]^T$	1, 2
$\mathcal{SN}_5$	$\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.01,\ 0.015,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 0,\ 2,\ -2,\ 2]^T$	1, 3
$\mathcal{SN}_7$	$\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \text{diag}[0.01,\ 0.01,\ 0.01,\ 0.015,\ 0.015,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 0,\ 2,\ -2,\ 2,\ 2,\ -2]^T$	1, 4

Higher-dimensional results. While the two-dimensional tests were not satisfactory for our approach, for completeness we did also run a few higher-dimensional tests, firstly to see whether sparse grids can at least bring something suitable where the kernel methods might struggle, and secondly to assess whether adaptivity can provide some relief where regular grids cannot. In particular, for the sparse grid method we ran simulations with the same array of choices of regularization parameter $\lambda = \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$ on a regular grid of level 5 and on three adaptive grids, one for each refinement strategy (surplus, surplus-volume, and surplus-absolute-value), starting on a level 3 grid and performing 5 refinement steps of 10/15/20 grid points for dataset dimensionality 3/5/7, respectively.

The input data distribution parameters for these test scenarios are given in Table 3.6, and the corresponding results are shown in Fig. 3.18. We again see higher errors obtained with SGDDerivE than with MISED. However, even the relatively small amount of grid adaptivity seems to be able to provide some improvements, especially visible in the lower-dimensional cases. Once more, as for DDE, of the three refinement strategies, our results suggest that surplus-absolute-value works best overall.

Discussion of results. Our density derivative estimator proved to have several issues that might not have been straight-forward to detect from the mathematical formulation. All our tests, both those presented here, as well as additional tests we have ran which for brevity we have not included here, give us no indication that our boundary smoothness condition would have any negative influence on the numerical results or that it was a too strong assumption. In fact, visually, the one-dimensional results performed well also on datasets with very wide support. Therefore, further tests on grids with boundaries as well as other suitable basis functions should be carried out in order to more definitively trace the root cause of the relatively poor performance of the method.

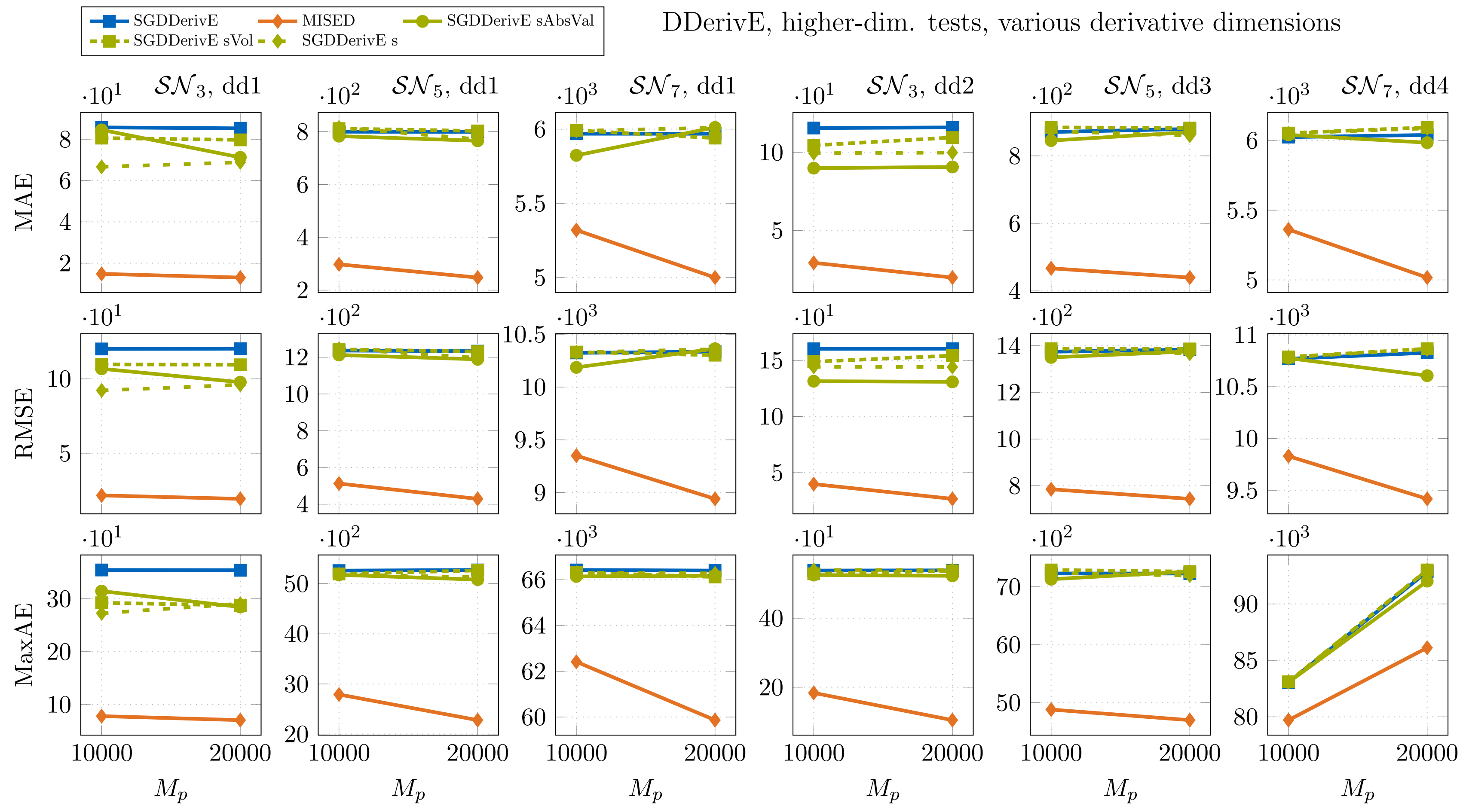


Figure 3.18: Higher-dimensional DDerivE results on skew-normal-distributed datasets. SGDDerivE showed again across the board worse results compared to MISED, however the adaptive grids did provide some relief by small, but significant, improvements on the accuracy.

Our tests were not meant to give any comparison between the methods in terms of runtime, due to the fact that our Python codes were not optimal with respect to any parallel performance. While these numbers are not shown here explicitly, in single thread runs we have observed that SGDDerivE does not manage to provide better runtimes than MISED, most likely due to the extra costs of using the B-spline basis of degree 3.

3.6 Density Derivative Ratio Estimation

Problem statement. Using the notation introduced in Eq. (3.27), the problem of *density derivative ratio estimation* (DDerivRE) can be formulated as follows:

> Given a set of independent and identically distributed samples in $\Omega \subseteq \mathbb{R}^d$, $\mathcal{S}_p = \{\boldsymbol{x}_i^p\}_{i=1}^{M_p}$ with density $p(\boldsymbol{x})$, estimate the ratio $\frac{\partial^{(\boldsymbol{j})} p(\boldsymbol{x})}{p(\boldsymbol{x})}$, under the assumption that $\partial^{(\boldsymbol{j})} p(\boldsymbol{x})$ is well-defined.

Existing approaches. Our literature review has shown that there has been extremely little interest not only in devising a direct method for estimating density derivative ratios, but also in investigating the usefulness of such a quantity. The main contribution, on which we also base our sparse grid algorithm, comes from Sasaki et al., who have introduced an algorithm to fit a kernel-based model under a regularized squared loss. They then show that this *least squares density derivative ratios* (LSDDR) method allows the implementation of mode seeking and ridge detection iterative algorithms [71]. To the best of our knowledge, LSDDR looks to be the only unified approach to handle derivative ratios of all orders developed prior to our sparse grid-based algorithm.

3.6.1 Sparse Grid Density Derivative Ratio Estimation

Mathematical derivation. For our *sparse grid density derivative ratio estimation* (SGDDerivRE) algorithm we use a similar cost function as the LSDDR method. Therefore, the regularized squared loss we try to minimize for function $r_{\boldsymbol{j}}(\boldsymbol{x})$ in a suitable space V is given by

$$J(r_{\boldsymbol{j}}) = \int_\Omega \left[r_{\boldsymbol{j}}(\boldsymbol{x}) - \frac{\partial^{(\boldsymbol{j})} p(\boldsymbol{x})}{p(\boldsymbol{x})} \right]^2 p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \|\Lambda r_{\boldsymbol{j}}\|_{L^2}^2. \tag{3.42}$$

The corresponding variational equation is then

$$\int_\Omega s(\boldsymbol{x})\, r_{\boldsymbol{j}}(\boldsymbol{x})\, p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} - \int_\Omega s(\boldsymbol{x})\, \partial^{(\boldsymbol{j})} p(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r_{\boldsymbol{j}}(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x} = 0, \forall s \in V. \tag{3.43}$$

Using the same integration by parts Eq. (3.30) under the same boundary smoothness condition as for density derivative estimation (that products of the form $\partial^{(\boldsymbol{j}')} s(\boldsymbol{x})\, \partial^{(\boldsymbol{j}'')} p(\boldsymbol{x})$ cancel out on the boundary $\partial\Omega$ for any pair of derivative encoding vectors $(\boldsymbol{j}', \boldsymbol{j}'')$ that satisfy $\boldsymbol{j}', \boldsymbol{j}'' < \boldsymbol{j},\ \boldsymbol{j}' + \boldsymbol{j}'' = \boldsymbol{j}$), the second term can be rewritten as

$$\int_\Omega s(\boldsymbol{x})\, \partial^{(\boldsymbol{j})} p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = (-1)^{|\boldsymbol{j}|} \int_\Omega \partial^{(\boldsymbol{j})} s(\boldsymbol{x})\, p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \tag{3.44}$$

resulting in the equivalent variational equation

$$\begin{aligned}\int_\Omega s(\boldsymbol{x})\, r_{\boldsymbol{j}}(\boldsymbol{x})\, p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} - (-1)^{|\boldsymbol{j}|} \int_\Omega \partial^{(\boldsymbol{j})} s(\boldsymbol{x})\, p(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \\ +\lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r_{\boldsymbol{j}}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = 0,\ \forall s \in V.\end{aligned} \tag{3.45}$$

Introducing the empirical estimators (Eq. (3.3)) and moving the second term to the right-hand side, we obtain

$$\frac{1}{M_p}\sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p)\, r_{\boldsymbol{j}}(\boldsymbol{x}_i^p) + \lambda \int_\Omega \Lambda s(\boldsymbol{x})\, \Lambda r_{\boldsymbol{j}}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} = \frac{(-1)^{|\boldsymbol{j}|}}{M_p}\sum_{i=1}^{M_p} s(\boldsymbol{x}_i^p),\ \forall s \in V. \tag{3.46}$$

Applying the sparse grid Ritz-Galerkin projection $r_{\boldsymbol{j}}(\boldsymbol{x}) \coloneqq \sum_{k=1}^N \alpha_k \varphi_k(\boldsymbol{x})$ and $s(\boldsymbol{x}) \coloneqq \varphi_l(\boldsymbol{x})$ for $V = \mathcal{V}_N$, we get

$$\begin{aligned}\sum_{k=1}^N \left[\alpha_k^{(\boldsymbol{j})} \frac{1}{M_p}\sum_{i=1}^{M_p} \varphi_k(\boldsymbol{x}_i^p)\, \varphi_l(\boldsymbol{x}_i^p)\right] + \lambda \sum_{k=1}^N \alpha_k^{(\boldsymbol{j})} \int_\Omega \Lambda\varphi_k(\boldsymbol{x})\, \Lambda\varphi_l(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \\ = \frac{(-1)^{|\boldsymbol{j}|}}{M_p}\sum_{i=1}^{M_p} \varphi_l(\boldsymbol{x}_i^p),\ \ l = 1, \ldots, N.\end{aligned} \tag{3.47}$$

In matrix-vector form, the resulting linear system to solve for surpluses $\boldsymbol{\alpha}^{(\boldsymbol{j})}$ is thus

$$\left(\frac{1}{M_p}\boldsymbol{B}^p \boldsymbol{B}^{pT} + \lambda \boldsymbol{C}\right)\boldsymbol{\alpha}^{(\boldsymbol{j})} = \frac{(-1)^{|\boldsymbol{j}|}}{M_p}\partial^{(\boldsymbol{j})} \boldsymbol{B}^p \boldsymbol{e}^p. \tag{3.48}$$

As it was the case with density derivative estimation, for the purposes of the current work we will focus solely on the first-order partial derivatives case $|\boldsymbol{j}| = 1$, i.e., we will estimate elements of $\nabla p(\boldsymbol{x})/p(\boldsymbol{x})$ by solving the system

$$\left(\frac{1}{M_p}\boldsymbol{B}^p \boldsymbol{B}^{pT} + \lambda \boldsymbol{C}\right)\boldsymbol{\alpha}^{(k)} = \frac{-1}{M_p}\partial_k \boldsymbol{B}^p \boldsymbol{e}^p, \tag{3.49}$$

for any given dimension (or direction) $k \in \{1, \ldots, d\}$.

Properties of the linear system. Similarly to the density derivative case, the linear system we obtain is consistent with previous results, in the sense that for the edge case $\boldsymbol{j} = \mathbf{0}$ one recovers the $\omega \to 1$ edge case of relative density ratio estimation. Overall, the linear system is again in the style of the sparse grid regression/classification algorithm, with both the system matrix and the right-hand side depending now on the same sample set $\mathcal{S}_p$.

Accuracy considerations. Like for SGDRE, for density derivative ratio estimation we can expect to be able to prove some theoretical convergence rates for specific types of basis functions [8], however in the general case we can only hope to have up to similar accuracies in practice as those for sparse grid regression problems, with the assumption that the more restrictive setting (in the sense that we impose more restrictions on the distribution of data points and the behavior at the domain boundary) does not have some pronounced numerical effect that is not existent for the regular regression problem.

Implementation details. SGDDerivRE combines the implementation restrictions of the two methods it resembles: the use of batches needs to be done with care, as similarly to SGDRE we cannot recover the full dataset results from the individual batch contributions, and, similarly to SGDDerivE, we have to restrict our choice of bases to B-spline functions of correct degree in order to obtain a useful approximation on the sparse grid.

Performance considerations. Unsurprisingly, the parallelization possibilities of our density derivative ratio estimator are basically the same as the ones mentioned for SGDRE. Thus, again, we would be restricted to a regression/classification-like parallelization effort in order to obtain higher levels of performance from our code, something which was not implemented.

3.6.2 Numerical results

Goals. The tests we performed were meant to assess in general the numerical properties of our SGDDerivRE method, and in particular to determine possible strong and weak points of our approach. For this purpose we ran multiple scenarios with various parameters and on various datasets sampled form known distributions, which provided a controlled setting for our analysis. Additionally, we have compared in all scenarios our sparse grid-based approach to an equivalent implementation of the kernel-based method LSDDR in order to determine to what extend the kernel-based method could be replaced in practice by our sparse grid approach.

Quality criteria. The same as with the rest of the complex estimators, the metrics we will be using in our tests to evaluate the properties of our approach are MAE, RMSE and MaxAE. For each test we ran we will compute these errors for each set of parameters, then selecting as best the results that obtain the minimal RMSE.

The testing pipeline. For density derivative ratio estimation we use the same testing pipeline written in Python that we have created for density derivative estimation, with a simple variable used to differentiate the two pipelines. The same freedom of choosing all relevant hyperparameters exists as for other methods and the testing pipeline work in the same way: we first create our input datasets by sampling from known distributions (see Appendix B), we create the corresponding input configuration files for SGDDerivRE, we run our method, as well as the corresponding kernel-based LSDDR, then compute the three errors, create plots (where needed, for visual inspection), and finally save all data to file for any further post-processing.

Testing parameters. Similarly to the density derivative estimation case, here as well we had to rely only on a MATLAB implementation of the LSDDR method[7], based on the algorithm described in [71], which we then translated into Python in order to integrate it in our testing pipeline.

In our tests we have used for LSDDR 5-fold cross-validation, following the strategy described in the literature, to choose the best hyperparameters from 10 possible kernel widths $\sigma_K = \left\{10^{-3}, 10^{-2.(6)}, \ldots, 10^{-0.(3)}, 10^{0}\right\}$ and 5 different regularization parameter values $\lambda_K = \{10^{-4}, 10^{-3}, \ldots, 10^{0}\}$. These values were chosen to give a good trade-off between a proper coverage of tested input distributions and a reasonably dense parameter search space.

For our SGDDerivRE implementation we rely on the modified not-a-knot degree 3 B-spline basis, for the same reasons as for density derivative estimation: reasonable grid sizes by excluding boundary points, fulfillment of the hierarchical splitting condition, and enough smoothness of the basis functions.

1D results. The first tests we have performed were on one-dimensional data, where we can easily check whether our approach provides reasonable solutions. To this end we have sampled datasets $\mathcal{S}_p \in \mathcal{N}_1 (\mu = 0.65, \Sigma = 0.01)$ of sizes $M_p = \{200, 500, 750, 1000\}$, searching for our sparse grid-based approach the best results for grid levels $l \in \{3, 4, 5\}$ and regularization parameters $\lambda = \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$.

These results, comparing SGDDerivRE to LSDDR in terms of minimal RMSE, are shown in Fig. 3.19. We observe quite a better performance from our approach compared to the kernel-based one, which happens for the same reason as for (relative) density ratio estimation: error evaluations take place over the whole domain, while the kernel-based method can only provide a good estimate where input data points exist, while SGDDerivRE can approximate the target function in the whole

[7]MATLAB code written by Hiroaki Sasaki, available at `https://sites.google.com/site/hworksites/home/software/lsdrf`

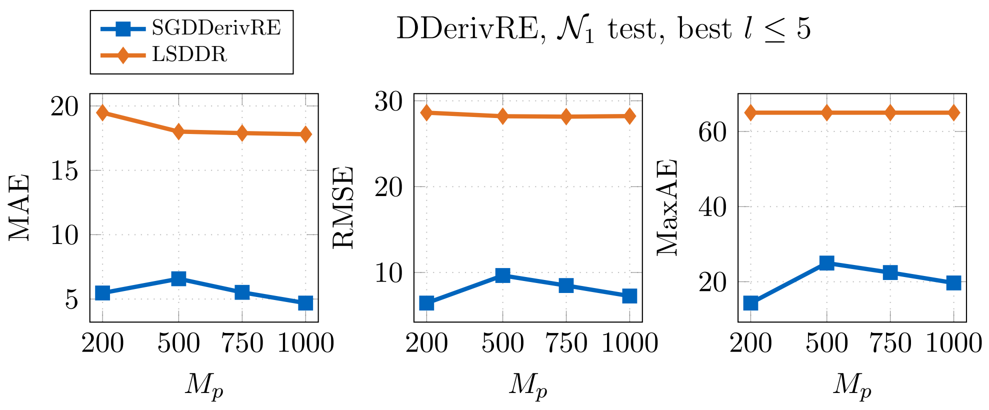


Figure 3.19: 1D DDerivRE results on normal-distributed datasets. We obtain better accuracies on sparse grids, as shown here, over the whole unit domain, but also when restricted to the support of the input dataset. As such, our approach can generalize better. To note that the nature of the density derivative ratio here is such that a lot of regularization is required if higher level grids are used.

computational domain. While for brevity not shown here, our tests showed however that the sparse grid method, at least for low grid levels, where for the chosen regularization parameter values we could avoid overfitting, provides equally good estimates also when limited to the support of the input dataset.

2D results. our next batch of tests were performed on two-dimensional datasets, where we can actually begin to see the benefits of using a sparse grid-based approach in terms of a reduction in grid sizes. The scenarios we investigated here are described in Table 3.7, with known single and mixed distributions being used to generate input datasets of 9 various sizes between 200 and 10000. For our sparse grid method we test 3 different grid levels, correlated with the input dataset sizes, where for $M_p < 5000$ we consider $l \in \{3, 4, 5\}$, while for $M_p \geq 5000$ we test $l \in \{4, 5, 6\}$, a decision done in order to always provide enough grid points to SGDDerivRE.

Results are shown in Fig. 3.20. We observe again that we obtain lower errors with SGDDerivRE than with LSDDR. Some additional aspects, which were observed practically, need to be mentioned. Firstly, we did not observe in these tests the same kind of numerical artifacts that affected SGDDerivE, which could be a consequence of the fact that the linear system in this case includes data point evaluations on both sides of the equation. Secondly, while the best accuracies were for the most part obtained for the sparse grid approach on the lower level grids tested, a consequence of our metrics being computed by evaluating in the whole domain, on level 5 grids we still obtained errors which, although higher, were still below those for LSDDR, but at a degradation of the local aspects of the numerical solution akin to overfitting. This is shown for selected results in Fig. 3.21. Lastly, the hyperparameter selection cross-validation process of LSDDR was proven to be less stable than expected. To

Table 3.7: Parameters of the 2D DDerivRE test scenarios.

test name	distrib. type(s)	distribution parameters
$\mathcal{N}_2$	$\mathcal{N}$	$\boldsymbol{\mu} = [0.4,\, 0.5]^T$, $\boldsymbol{\Sigma} = [[0.02,\, 0.01];\, [0.01,\, 0.04]]$
$\mathcal{SN}_2$	$\mathcal{SN}$	$\boldsymbol{\xi} = [0.5,\, 0.5]^T$, $\boldsymbol{\Omega} = [[0.08,\, 0.02];\, [0.02,\, 0.06]]$, $\boldsymbol{\alpha} = [5,\, -2]^T$
$\mathcal{T}_2$	$\mathcal{T}$	$\boldsymbol{\mu} = [0.4,\, 0.5]^T$, $\boldsymbol{\Sigma} = [[0.02,\, 0.01];\, [0.01,\, 0.04]]$, $\nu = 10$
$\mathcal{ST}_2$	$\mathcal{ST}$	$\boldsymbol{\xi} = [0.5,\, 0.5]^T$, $\boldsymbol{\Omega} = [[0.035,\, 0.0175];\, [0.0175,\, 0.0525]]$, $\boldsymbol{\alpha} = [3,\, -2]^T$, $\nu = 5$
$m\mathcal{N}_2$	$0.4 \cdot \mathcal{N}$ $+0.3 \cdot \mathcal{N}$ $+0.3 \cdot \mathcal{N}$	$\left\{\begin{array}{l} \boldsymbol{\mu} = [0.35,\, 0.5]^T,\ \boldsymbol{\Sigma} = [[0.01,\, 0.0075];\, [0.0075,\, 0.02]] \\ \boldsymbol{\mu} = [0.5,\, 0.5]^T,\ \boldsymbol{\Sigma} = [[0.03,\, 0];\, [0,\, 0.01]] \\ \boldsymbol{\mu} = [0.6,\, 0.4]^T,\ \boldsymbol{\Sigma} = [[0.01,\, 0.0075];\, [0.0075,\, 0.02]] \end{array}\right.$
$m\mathcal{ST}_2$	$0.5 \cdot \mathcal{ST}$ $+0.5 \cdot \mathcal{ST}$	$\left\{\begin{array}{l} \boldsymbol{\xi} = [0.45,\, 0.55]^T,\ \boldsymbol{\Omega} = [[0.025,\, 0];\, [0,\, 0.01]] \\ \quad \boldsymbol{\alpha} = [0,\, 3]^T,\ \nu = 3 \\ \boldsymbol{\xi} = [0.55,\, 0.45]^T,\ \boldsymbol{\Omega} = [[0.01,\, 0];\, [0,\, 0.025]] \\ \quad \boldsymbol{\alpha} = [3,\, 0]^T,\ \nu = 3 \end{array}\right.$

this end, the results shown here for $M_p \geq 3500$, where most problems were observed, are the best obtained over 4 different runs, and even with these extra steps taken reasonable results were not always obtained. In the plots any missing values of the kernel method correspond to obtained errors 5 times larger than on sparse grids, which we avoided to show in order to make the plots readable.

Higher-dimensional results. Lastly, the final tests looked at the behavior for higher dimensional datasets, where we can also more clearly assess the accuracies we obtain when evaluating strictly at the data points. The parameters for these test scenarios are presented in Table 3.8. We considered datasets of sizes $M_p = \{10000, 20000\}$ and for the sparse grid runs we used the same array of regularization parameters λ and both a fixed level 5 grid and the usual three adaptive grids obtained by starting with a regular level 3 grid and applying 5 adaptive steps with 10/15 refinement points per step for dimensionality 3/5, respectively, using the surplus, surplus-volume, and surplus-absolute-value refinement criteria.

Results for minimal RMSE per test case are presented in Fig. 3.22. For the larger dataset sizes we could not obtain results with LSDDR in a reasonable amount of time, therefore those entries have been skipped. We can observe that overall we obtain better results form our approach than from the kernel-based method for the level 5 grids, as well as for most of the adaptive ones. In terms of best refinement criterion, surplus-absolute-value provided again the more consistent improvements

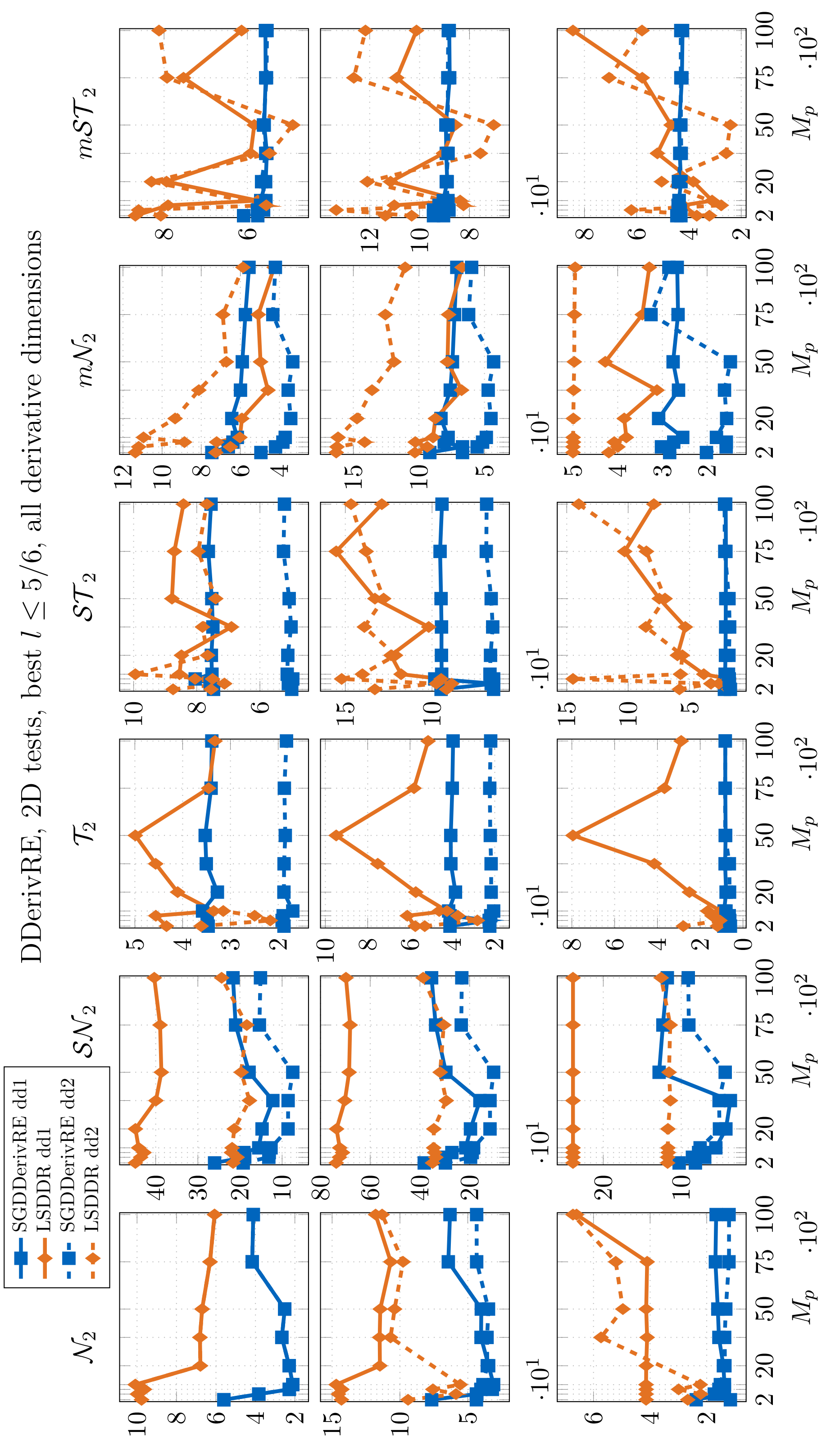


Figure 3.20: 2D DDerivRE results on various datasets sampled from single and mixed known distributions. In general we observe that SGDDerivRE performs better, being able to generalize in the whole domain, not just close to the data points. However, one has to note that results on level 5 grids can be less accurate, although they better fit the data and errors are still lower than for the kernel method. LSDDR can indeed obtain nice results close to the data points, but the cross-validation parameter selection process proposed in literature is not stable: its results here are over 4 separate runs, with overfitting still taking place in many of the test cases.

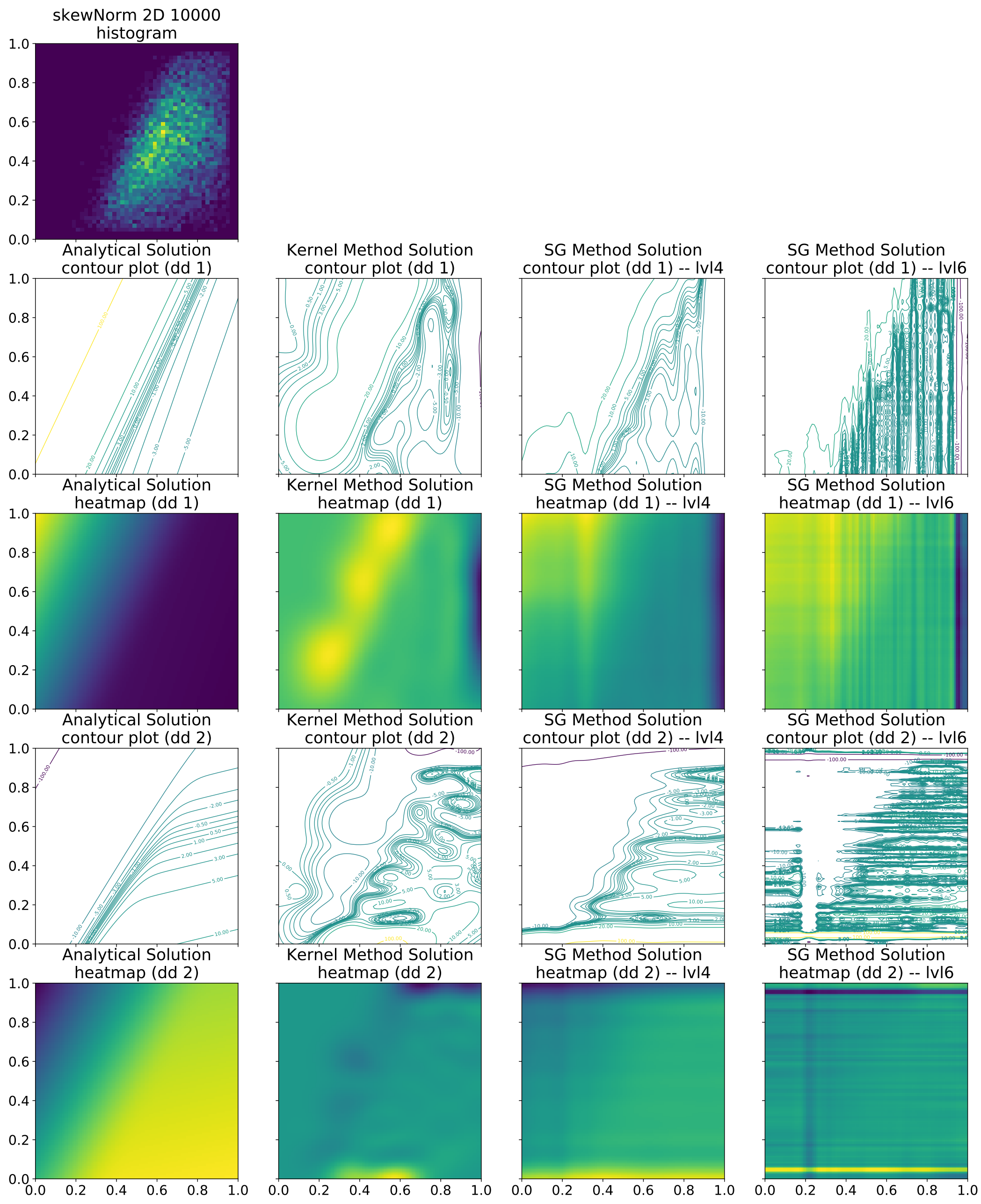


Figure 3.21: Visual inspection of the DDerivRE results for the $\mathcal{SN}_2$ test scenario. Both derivative dimensions are shown. We compare the contour plot and heatmap of both LSDDR and SGDDerivRE for the best obtained parameters against the analytical solution of the target density derivative, for the largest dataset size considered. For the sparse grid method we show both the best results obtained with a level 4 grid, as well as the approximation on the corresponding level 6 grid. We can observe from the contour plots that both numerical methods provide relatively similar results in the input dataset support, but the sparse grid method generalizes better. On the other hand, for larger sparse grids we again see similar direction-wise numerical artifacts as in the DDerivE case, meaning that this issue was not resolved completely here, just delayed by the inclusion of the data points in the linear system's matrix as well.

Table 3.8: Parameters of the higher-dimensional DDerivRE test scenarios.

test name	distribution parameters	derivative dimensions
$\mathcal{SN}_3$	$\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \mathrm{diag}[0.01,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 2,\ -2]^T$	1, 2
$\mathcal{SN}_5$	$\boldsymbol{\xi} = [0.5,\ 0.5,\ 0.5,\ 0.5,\ 0.5]^T$, $\boldsymbol{\Omega} = \mathrm{diag}[0.01,\ 0.01,\ 0.015,\ 0.015,\ 0.015]$, $\boldsymbol{\alpha} = [0,\ 0,\ 2,\ -2,\ 2]^T$	1, 3

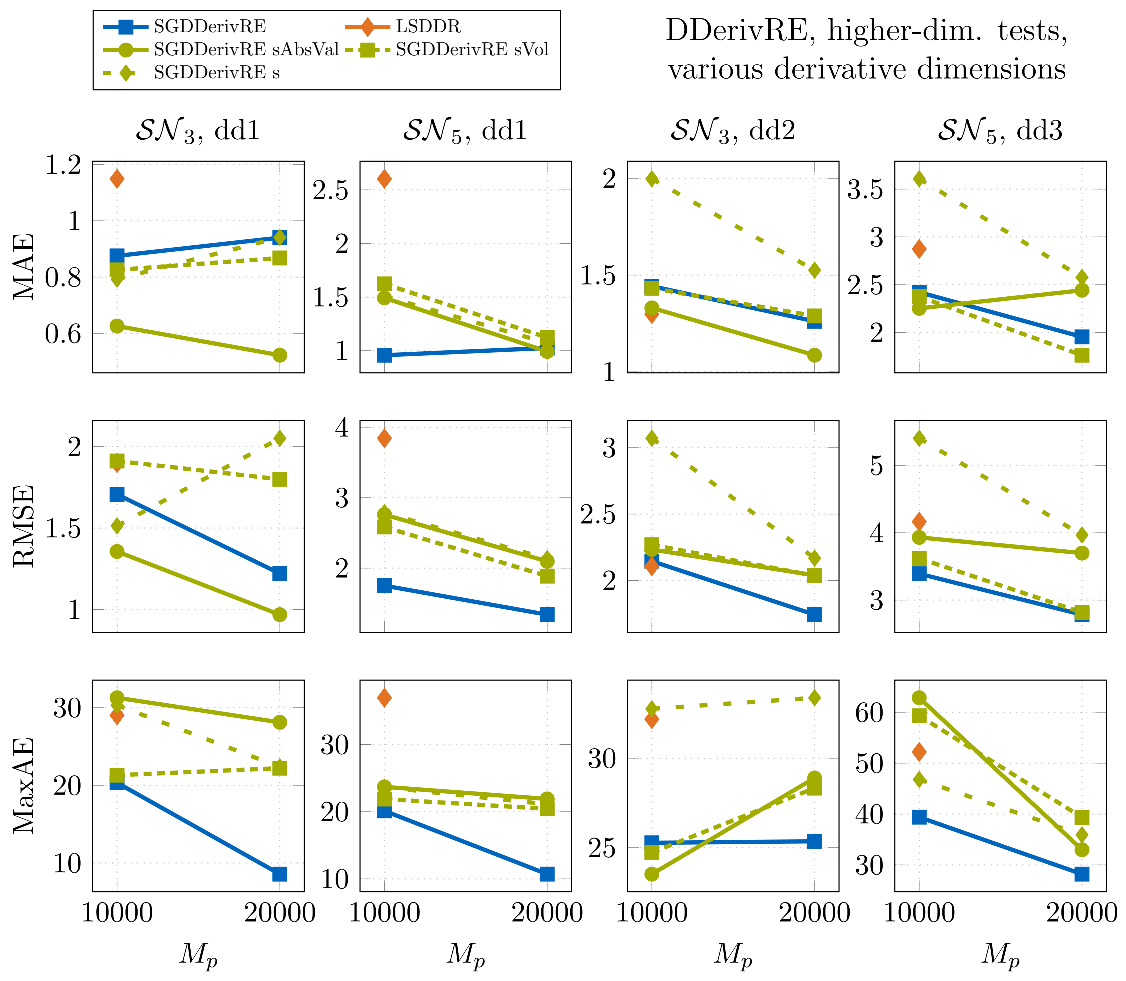


Figure 3.22: Higher-dimensional DDerivRE results on skew-normal distributed datasets. While here the evaluations are performed at the data points, thus eliminating the issue of generalizing in the whole domain, SGDDerivRE still shows smaller errors than LSDDR. In general, also the adaptive grids show better accuracies than the kernel-based method, with the surplus-absolute-value refinement criterion showing to be the most robust.

in all scenarios, although in a couple of tests it got superseded by the surplus-volume refinement.

Discussion of results. In terms of approximation quality, our sparse grid approach performed well, especially considering the poor performance of the density derivative estimator, results showing SGDDerivRE to be both stable and more consistently accurate than LSDDR. An open question remaining for future endeavors is how well these results translate into applications of the density derivative ratio estimator. Particularly, as our sparse grid approximation is in general less smooth than a Gaussian kernel-based estimator and our , whether our sparse grid approach can be used as a part of a hill-climbing process to efficiently find modes and ridges in datasets, something where LSDDR was shown to perform very well [71].

As before, for this estimator we again did not provide explicitly any runtime comparisons, due to the nature of the implementations and the limitations of the Python code. However, we can make some observations from our experience on single threaded runs. For datasets of all dimensions the sparse grid approach reaches a solution in similar time as the kernel-based method, if we take into account the time spent finding the best hyperparameters for both methods. Wehn adaptivity comes into play, as well as in general when evaluating the resulting estimator however, SGDDerivRE falls behind LSDDR due to the extra cost incurred by the B-spline basis.

It also has to be noted that from a practical perspective higher order density derivatives in general are expected to quickly start loosing numerical benefits when approximated by sparse grids due to the corresponding increase in degree of the basis functions required, leading to an unsustainable increase in computational effort. Therefore, and correspondingly valid for SGDDerivE as well, while the mathematical formulations for density derivative and density derivative ratio hold for all possible choices of $\boldsymbol{j}$, one should, unless explicitly proven otherwise, restrict itself in practice to using at most second order derivatives ($|\boldsymbol{j}| \leq 2$) and correspondingly only degree 3 (spline) basis.

3.7 Summary

In this chapter we have taken a look with an algorithmic focus at a certain type of learning task, specifically the issue of estimating functions of densities, introducing to this end completely novel approaches of solving these problems by means of sparse grids. For each method we have delved into the mathematical derivations needed to obtain a solvable linear system, their algebraic properties, implementation and performance aspects, and lastly performed various tests to assess their numerical capabilities compared to known similar kernel-based methods.

Our two-dataset methods proved to be the most successful, with the sparse grid density difference estimator matching closely the previously known good performance of the SGDE method of Peherstorfer, and the sparse grid density ratio estimator introducing a novel use of the regression/classification type of linear system. For both methods our results showed that they are at least competitive, if not better, than kernel-based counterparts and possessed nice properties of practical relevance. With the kernel-based approaches having found various further applications, these good results will hopefully thus open the door for sparse grids to be used as well in such applications, previously not considered (e.g., [78, 80, 81, 48]).

Our second set of methods, with single dataset inputs, struggled however to obtain the same levels of performance in numerical tests, even though they came from sound mathematical formulations. Our proposed solution to the density derivative estimation problem was plagued by issues that were only revealed in visual and numerical tests, and as such further tests would need to be carried out in order to fully asses the method's practical applicability. On the other hand, the sparse grid density derivative ratio estimator was more robust and produced consistently better results than its kernel-based counterpart, however more tests will be required to assess whether usual applications of derivative ratios can be handled by our sparse grid method (e.g., [71]).

Overall, these results make us hopeful that our new algorithms will find their usefulness moving forward into expanding the already large array of applications of sparse grids in learning tasks, as well as providing new algorithmic tools previously not considered.

4 Contributions with a High-Performance Computing Focus

High-performance computing (HPC) has become today a quite indispensable aspect of numerical software solutions. With ever growing amounts of data that have to be evaluated and processed fast and reliably, codes have to be able to cope with these demands by utilizing not only the best algorithmic approaches existing, but also by harnessing the underlying hardware capabilities available to their fullest. While of course ideally one would always look for simply the fastest time to solution, in practice many factors do come into play when discussing progress in HPC. One aspect is related to the hardwares being released, which are becoming increasingly complex, so significant time and resources have to be put into understanding them, as treating each new processor simply as a plug-and-play device for one's code will most likely underutilize the available resources brought by the combination of the normal generational leap in fabrication technology and the changes to existing hardware architecture. This is not always an easy task, as one has to rely both on their own analysis and experimentation, as well as the results of others, where available. Another aspect that has to be taken into account is that pure algorithmic improvements can sometimes provide more increase in performance than code optimizations or better hardware. However, while of course one should look to have the best underlying algorithms, legacy codes, even when avoidable, can still offer valuable insights, not least in the area of HPC.

In our view on high-performance development in learning problems we will focus on the tasks of regression and classification. Until the introduction of the density estimation-based approach of Peherstorfer [59], classification using sparse grids was exclusively done with the method of Pflüger [64], which can also be used to perform regression. As was discussed in Chapter 3, the data-independence of the system matrix in the SGDE algorithm gives an advantage when it comes to the size of the problems that can be tackled, with the computational bottleneck being transfered to the right-hand side (as was pointed out also in [69]). That however is not the case with the standard regression/classification algorithm of Pflüger. Therefore, this approach has seen significant development especially in terms of optimizing the evaluation of basis functions at data points, which is the core operation required in order to solve the task's resulting linear system. The biggest contribution in the HPC direction was done by Heinecke [34], who has implemented a highly efficient parallel and scalable version of the standard sparse grid evaluation step. While subsequently Pfander et al. [63] have shown superior performance by using a different, subspace-based, approach to perform these evaluations, our work has been based on that of Heinecke as part of the Intel Parallel Computing Center project at the Leibniz Supercomputing Center and the Technical University of Munich. This code uses an intrinsics-based implementation with an intuitive and straightforward approach to parallelization on various architectures (including Intel CPUs), all resulting in and from a better, in-depth understanding of the hardware.

Before delving further into this chapter, we need to make a small note on terminology specifics. As mentioned, in the sparse grid approach to solve the tasks of regression and classification, the resulting linear systems are virtually identical and therefore we can treat these two scenarios, otherwise different from a machine learning perspective, homogeneously. However, for the sake of coherence, we will proceed to use specifically the term 'classification' in the code optimization study Section 4.1, and the term 'regression' in the context of the time series prediction scenario in the subsequent Section 4.2, with the clear understanding that practically the same resulting linear system in the sparse grid approach, and thus the same code implementation, is being used to solve both of these machine learning tasks.

This chapter is structured as follows: we will begin with an introduction into the high-performance code for the regression/classification scenario using sparse grids implemented by Heinecke. Next we will shortly present the specifications of the hardware for which we have targeted our legacy code extensions, then explain in more detail the steps we took to reach the numerical results which will be shown afterwards and which focus on a classification scenario. These results are based mainly on our previously published material [70]. This study of the performance we can achieved with our code will be paired in the second part of the chapter with a concrete application that would benefit from the low time to solution we can provide. Specifically, we will show how such an optimized, fast code allows to easily study the use of spatially adaptive sparse grids for regression-based time series prediction on real financial data, where existing work was only done previously using the combination technique.

4.1 Code optimization study

4.1.1 State of the legacy code

The code we used as a basis for our optimization effort is a legacy implementation by Heinecke [34], based on a past version of the SG++ library. In the following we will shortly present details of his approach and implementation which are relevant to our subsequent optimization work, with further details to be found in the source material.

Algorithm implementation. As the linear system in Eq. (2.44) is being solved in the SG++ library using the conjugate gradient (CG) method, one has to mainly focus on optimizing the two matrix-vector multiplications with the matrices $\boldsymbol{B}$ and $\boldsymbol{B}^T$, called *multT* and *mult*, respectively. In order to allow for an easy parallelization, Heinecke opted for an iterative approach, where one computes all the contributions of all basis functions at all data points, which are then cumulated at the end of the traversal. While many of these contributions will be zero for any given data point, as they lie in the support of only part of the basis functions, the trade-off is worthwhile due to the fact that it allows a straight-forward distribution of the workload by parallelizing across the loops. As stated in the original work by Heinecke, such an approach results in compute-bound, embarrassingly-parallel operations suitable for running on modern architectures.

For a better use of the *single-instruction multiple-data* (SIMD) vector operations on CPU architectures, Heinecke opted for manual loop unrolling, where operations are being split further across chunksizes reflective of the vector width of the hardware. Taking into account the amount of available registers for SIMD operations, one can compute the chunksize that maximizes the amount of data points that can be operated upon at the same time. Additionally, to make sure this distribution works, the input dataset is padded from its initial size M upwards to the nearest multiple of the SIMD vector length, allowing for exact looping. The padding is done by repeating the last data point - target pair as many times as necessary, which has otherwise no impact on the actual classification results.

A feature of this code that made it attractive for the purpose of our optimization study is its reliance on *intrinsic functions* (or *intrinsics* for short) that allow for a real low-level control of the mathematical operations through manual vectorization. Intrinsics are CPU-specific instructions that the compiler treats differently than the regular language-specific ones. Therefore, one of the main tasks of porting the computational kernels to a new CPU architecture lies in the updating of these intrinsics for the new target hardware.

The implementation of these operations allowed both shared (using OpenMP) and distributed (using MPI) memory parallelism. Each of the two operations, *mult* and *multT*, consist normally of 3 main loops: over the dataset, over the grid points, and over the dimension of the space. To remove loop carried dependencies and provide enough computational intensity per loop iteration, the first operation is parallelized across the dataset, while the second one is over the grid points, with a static scheduling strategy chosen for the OpenMP implementation. Additionally, explicit prefetching instructions allow the hiding of memory latencies in order to remove idle times in the computational kernels.

Distributed memory parallelism. The distributed memory parallelization with MPI, which allows scaling also beyond the node level, was obtained by implementing several different communication schemes in order to synchronize the work performed on different chunks of the data. The first scheme, called *Alltoallv*, distributes the chunks of data to each process and then performs a synchronization step at the end of the operation (be it *mult* or *multT*) to gather the individual contributions with a call to the *MPI_Allgatherv*() function. This approach clearly distinguishes between the computation and communication parts of the matrix-vector operations, however at the cost of poor scalability for a high number of processes.

The second scheme proposed in the original implementation, *Async*, performs a better overlap of the computation and communication parts by performing point-to-point asynchronous transfer of contributions between the working processes. This was shown to outperform the previous scheme, however it still does not allow high scaling at a cluster level parallelism due to the increased communication overhead that comes in those scenarios. This scheme could lend itself to

Lastly, the third scheme is called *Allreduce*. As the name suggests, it proposes the use of the *MPI_Allreduce*() function to perform one single final accumulation of the results at the end of a combined *mult* and *multT* operation. This approach requires a change in the iterative traversal for the second matrix-vector operation, as now *multT* also needs to be parallelized across the dataset. This however is a small drawback, considering the fact that this scheme allows each process to store only a small part of the data at any given moment, lending itself to be used on very large datasets.

Handling the sparse grid basis functions. For each architecture, we not only have two computational kernels, one for each of the two main matrix-vector operations, but we also have two variants of each, depending on whether we employ linear or modlinear basis functions. Each of them comes with their own advantages and caveats. For both bases, in order to have a higher level of efficiency in the traversal of the grid points, the overall data structure of the grid had to be changed from an array-of-structures (AoS) to a structure-of-arrays (SoA) one. The evaluation in the case of the linear basis function is quite straight-forward, however the usual sparse grid restrictions do come into play here as well: in order to correctly approximate on the boundary of the domain one needs to invest substantially more grid points. The modified linear basis provides relief to this problem while maintaining high classification accuracy, as was demonstrated in the existing results [34], however at the price of a more complex implementation. The performance hit is somewhat offset by a masking technique and the precomputation of certain intermediate values, which alleviate an otherwise costly 4-way conditional.

4.1.2 The optimization process

Goals. With an otherwise unique take on the optimization process, by using manual vectorization techniques and with a good understanding of both the algorithms and the hardware on which they should run, this legacy code has shown already its capacity to perform very well on various CPU and GPU architectures, part of a larger project withing the Intel Parallel Computing Center (IPCC) at the Leibniz Supercomputing Center (LRZ) in Garching, Germany. The purpose of our work, set under the same project umbrella, lied in studying the performance gains one can expect to obtain from this implementation on newer architectures, under the same approach to the optimization process. Our target processor, Intel's second generation Xeon Phi, code-named *Knights Landing* (KNL), was one of the first chips to implement the wider AVX-512 instruction extension on x86 machines[1] and it came with various additional hardware novelties, all of which made it an attractive architecture to investigate with our application. Especially interesting from an academic perspective is represented also by the scaling performance between different generations of processors, as such comparisons with its direct precursor, the AVX2-based Haswell architecture, were also performed.

The target architecture. The specific CPU for which we optimized our code is the Xeon Phi 7210-F model installed in the CoolMUC3[2] segment of the Linux Cluster[3] system of the LRZ. This processor has 64 cores, with 2 vector processing units (VPUs) per core capable of fused multiply-add (FMA) operations, and it implements the AVX-512 SIMD extension. Each core allows 4-way hyperthreading at a nominal frequency of 1.3GHz. In terms of storage, this chip contains a 96GB DDR4 distant memory and, as a novelty at the time of its introduction, a 16GB MCDRAM high-bandwidth closer memory.

The CoolMUC3 cluster contains 148 such KNL nodes in an Intel Omnipath-based network, whose optimal operating temperature is maintained though a water cooling system. At the time of our optimization runs, the cluster operated in a mode that imposed the nodes to run with Intel Turbo Boost enabled, which makes the operational frequency to follow a predefined set of bins depending on the arithmetical intensity of the instructions executed[4] (an aspect that comes into play when interpreting results in terms of fraction of peak performance). Overall, the cluster has a theoretical peak of 394Tflop/s (2.662Tflop/s per node) and a LINPACK performance of 255Tflop/s (1.723Tflop/s), both in double precision arithmetic.

The introduction of the high-bandwidth MCDRAM memory is not the only novelty, as well source of both complexity and flexibility, of the KNL processor. The cache architecture of the chip also suffered a redesign compared to previous chips. While each core of a KNL chip still has its own level 1 cache and the level 2 cache is shared between tiles (with two cores each), the level 3 cache has been completely

[1]The 512-bit SIMD units were first introduced by Intel on the Many Integrated Core (MIC) PCIe card, named Knights Ferry. The first generation Xeon Phi, code-named Knights Corner (KNC), also used 512-bit wide vectors, was produced as a coprocessor. This makes the KNL the first stand-alone Intel processor to implement this SIMD extension.

[2]`https://doku.lrz.de/display/PUBLIC/CoolMUC-3`

[3]`https://doku.lrz.de/display/PUBLIC/Linux+Cluster`

[4]Information on the frequency bins could originally be found in Intel's 2nd Gen Xeon Phi product brief (`https://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-processor-product-brief.html`). This is however not the case anymore at the time of this thesis being written. One has to rely instead on other, reliable and long-term available, online sources of information, for example: `https://www.nersc.gov/assets/Uploads/Using-KNL-Processors-Feb2019.pdf`

removed. Instead, a two-directional communication mesh is introduced in order to maintain coherence between all level 2 caches. Additionally, the fast MCDRAM can sometimes substitute partially the need for a level 3 cache (as will be described shortly).

To allow users to better leverage these architectural changes in the KNL die, two set of operational modes exist[5]. The *clustering modes* describe how we expect the data to be distributed across compute cores in order to facilitate better locality at the tile level (i.e., level 2 cache). Three options exist:

- The *all-to-all* mode is the simplest and, as such, also the less efficient. It proposes a uniform distribution of the memory addresses across all tiles, and therefore it is only suitable for debugging purposes.
- The *quadrant* mode splits the tiles into 4 logical sections (i.e., hidden to the programmer), each with its own memory address controller. This allows for overall lower latencies of level 2 cache misses, and therefore it is one of the preferable modes of operation for most codes. Moreover codes requires virtually no additional work in adapting them to run with this mode. On some processors versions (different to the one we used in our tests) there is a *hemisphere* variant of this mode, where the only difference is that the logical split takes place in halves instead.
- There are codes that are non-uniform memory access (NUMA)-aware, where one takes additional care to make sure each processor works on data located as close by to it on chip as possible in order to maximize performance. For such cases there exists the *SNC* (sub-NUMA clustering) mode. This is especially helpful in the case of memory bound, NUMA-aware codes where one wants to pin compute threads and their active memory to specific NUMA nodes. The KNL chip used in our optimizations supports the *SNC-4* option, where the split is into 4 NUMA nodes, with other KNL models allowing also a hemisphere split (*SNC-2*).

The *memory modes* describe a second, independent set of ways in which the KNL node can operate. They control the manner in which the MCDRAM high-bandwidth memory (HBM) is being treated by the processor. Again, three options exist:

- In *flat mode*, all the memory addresses of the MCDRAM are mapped as regular address space, the same as the large DDR4 main memory. This is first, and for most codes, best memory mode to work with. For applications that fit inside the 16GB limit, one has to only make sure at runtime to

[5]While many best practice guides on the KNL modes exist, we have found the following to be most useful:
`https://colfaxresearch.com/knl-numa/`
`https://colfaxresearch.com/knl-mcdram/`

allocate all memory to the fast MCDRAM by means of, e.g., the *numactl* instruction. Even for larger applications this mode can be beneficial as long as one allocates manually the bandwidth-critical data structures to the faster memory.

- The *cache mode* is meant to somewhat bridge the gap left by the exclusion of the level 3 cache from the KNL chip architecture. As the name suggests, in this mode the MCDRAM takes effectively the role of a level 3 cache. While sometimes beneficial, especially due to the fact that no explicit memory mapping or code changes are needed to utilize it efficiently, for most scenarios this mode simply adds extra latency and as such its use-case is limited.
- The most complex mode is the *hybrid* one. Drawing from the benefits and the drawbacks of both previous two modes by splitting the HBM into a cache and an addressable part, this option can be useful for very large applications where only some parts of the code allow allocations to take place in the HBM.

Details of the code optimization process. Being a demonstrable compute bound implementation, the first and main task of the optimization process was to update the intrisincs from those targeting AVX2 instructions to the new AVX-512 wide SIMD versions.[6] For the most part the transition could be done easily, as direct correspondent instructions could be found. In terms of new capabilities of the new instruction set is the possibility for improved embedded broadcasting, which theoretically provide a speed-up, although the effect is hard to quantify as our code does not depend heavily on such operations. On the other hand, as the KNL chip only implements the very basic AVX-512 SIMD instructions, some options to optimize the code, such as the usage of more complex masking operations on multiple integer valued registers (found in the AVX-512DQ extension) could not be implemented (they are however available on the more recent Xeon architectures).

Regarding the OpenMP implementation, some additional interesting aspects were discovered during the optimization process that are of note. First of all, we found no reason to switch from a static (i.e., manual) partitioning to a dynamic one, as the workload is quite evenly distributed between processes. We have found however that this is not necessarily the case when using only part of a compute node (i.e., using only part of the threads). For a low to medium number of cores active dynamic scheduling managed to get a slight improvement over the static partitioning, however this effect could not hold up for the entire 64 cores due to the ever growing overhead of context switching.

A second interesting find relates to the loop chunk size used for unrolling the dataset loop. The idea is to compute this value such that it relates to the number of actual available SIMD registers of the processor in question and the amount of dataset instances that can actually physically be processed at the same time. For the previous Haswell architecture, 16 SIMD registers are available, out of which

[6] `https://software.intel.com/sites/landingpage/IntrinsicsGuide/`

Table 4.1: Parameters and accuracy for the sparse grid binary classification problem on the 5D chessboard dataset. We use the same settings as in existing results [34].

basis function	λ	grid level	number of refinements	refinement points	CG iterations	classification accuracy
linear	10^{-6}	3	6	100	250	93%
modlinear	10^{-7}	5	8	100	250	92%

2 are needed to store current grid point information (level and index) and 2 are needed to store intermediate results and act as buffers. This leaves 12 registers for loading the dataset instances and to store the final results, 6 each respectively. Heinecke proposed for this chunk size to be expanded from the thus computed value of 24 for AVX2 nodes to 96 for newer AVX-512 architectures, with a factor of two accounting for the doubling of the vector width and another factor of two from the doubling of available SIMD registers. From a similar calculation as before however, one would expect from this doubling 26, not 24, free registers to be split between data points and result vectors. Surprisingly, this did not hold up in practice, where a respectively higher chunk size of 104 produced less efficient code. Analyzing the resulting assembly codes, we concluded that this effect is due to the compiler's optimization efforts, with the extra 2 SIMD registers we considered to be idle being put to better use as intermediary buffers instead.

Lastly, regarding the three MPI communication schemes, as they are independent of the underlying code optimizations performed, we were able to utilize them with virtually no changes for the comparison tests with the OpenMP parallelizations at the node and cluster levels.

4.1.3 Numerical results

The testing scenario. In order to assess the results of our optimization process, we wanted to use a known scenario, one where good parameters for the sparse grid algorithm are known and which allows us to easily scale the problem size. Our choice was the binary classification of the *chessboard* artificial dataset in $d = 5$ dimensions, with 3 regions per dimension, whose targets are given by the formula:

$$y_i := \prod_{k=1}^{d} \begin{cases} -1, & \text{if } \frac{1}{3} < \boldsymbol{x}_{i,k} \leq \frac{2}{3} \\ 1, & \text{otherwise} \end{cases}, \ 1 \leq i \leq M. \tag{4.1}$$

Good sparse grid parameters for this dataset, as well as the classification accuracy are known from existing results (Table 4.1). Additionally, the dataset possesses a simple generating rule which allows us to easily set up datasets of various sizes.

For comparison purposes, we have run core and node tests also on a previous architecture, specifically the Xeon E5-2697 v3 (Haswell) compute nodes of the

CoolMUC-2[7] segment of the Linux-Cluster system of the LRZ. These nodes are 28-cores dual-socket CPUs, with two-way hyperthreading, a nominal frequency of 2.6GHz, and connected via an FDR14 Infiniband interconnect. The peak performance of this 812-node system is 1400Tflop/s (1.724Tflop/s per node).

The code was compiled using Intel's C++ compiler v17, which implements the OpenMP 4.5 standard, with Intel MPI v2017 used for the distributed memory parallelization. Throughout our tests we have used the usual recommended compiler flags for KNL as found in many performance optimization guides[8].

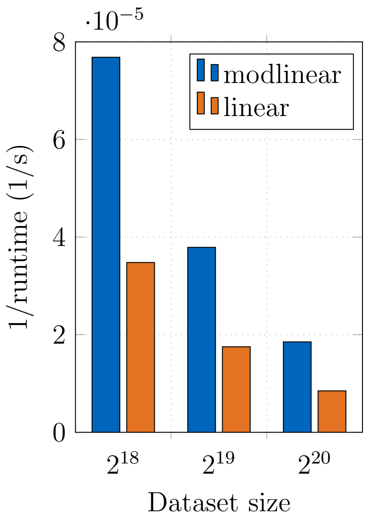
Performance expectations. The nature of algorithms that one employs will determine the amount of theoretical performance one should expect from running code on modern architectures. In the case of our classification algorithm, the kernels of the two matrix-vector operations contain only one FMA operation, with little other instructions that can be performed in parallel. As such, the theoretical performance peak of the linear basis version of the two kernels can only slightly exceed 50%. The modified linear basis implementation, with its more complex structure and branching conditionals, can respectively only obtain about a third of the peak performance the hardware can offer. While these ideal values can hardly ever be reached by non-trivial compute kernels, we will show that our implementation does come close to these predicted values.

Single thread and single core results. The tests we perform are meant to assess both the strong and weak scaling capabilities of the optimized code. As such, we run all tests on datasets of three sizes: 2^{18}, 2^{19}, 2^{20}. We chose these sizes as they are similar to those used in the legacy results, while still providing enough computational workload per thread when moving to the full chip. As a baseline we have the single thread results, shown in Fig. 4.1, which already confirm the previously known good weak scaling characteristics of the compute kernels. To reduce the influence of spinning from threads, especially for low thread counts, we report performance in terms of inverse runtime instead of number of floating-point operations per second (Flop/s), with the same understanding that higher values are better.

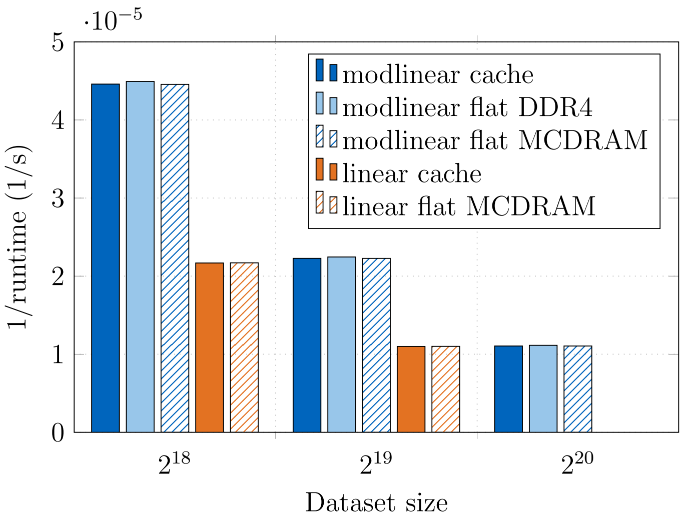
For the test runs on a full KNL core we are interested in the behavior under hyperthreading, the results being shown in Fig. 4.2 and Fig. 4.3 for the modlinear and linear basis functions cases, respectively. KNL is argued to be able to utilize the full performance of a core with just a single thread, however that can happen only under ideal situations, where there are latencies between operations. In our computational kernels that is not the case, with KNL instructions usually having slightly higher latencies than their Haswell counterparts, so using multiple threads allows those latencies to be hidden. Using more than one thread brings forth also

[7]`https://doku.lrz.de/display/PUBLIC/CoolMUC-2`

[8]e.g., the PRACE (Partnership for Advanced Computing in Europe) guide for KNL, available at: `http://www.prace-ri.eu/best-practice-guide-knights-landing-january-2017/`



(a) Single thread results on Haswell



(b) Single thread results on KNL

Figure 4.1: Comparison of single thread results. Higher values are better. The Haswell runs are done with the legacy version of the code as a baseline for our later comparisons. On KNL we run our optimized code using the operating modes that are suitable for our code, i.e., the cache memory mode, the flat memory mode with allocation in main DDR4 memory, and the flat memory mode with allocation in HBM. All three settings use the quadrant clustering mode. To be noted that for the largest datasets the KNL runtime for linear basis functions exceeded the maximum allocation of 48 hours allowed on the CoolMUC-3 cluster, and therefore those results are missing.

the discussion on thread pinning, which makes sure that threads do not migrate during execution, which would be detrimental to any performance testing. For all our runs we have used compact affinity, where threads prefer to fill in partially populated cores first. As our kernels use a manual split into chunks in a consecutive manner, allocating threads also consecutively in the KNL die should offer optimal data locality, and thus provide the best outcome. In fact, similar conclusions have been reported in literature for applications that utilize data access patterns such as ours [39], although one should in general be otherwise cautious when generalizing such assumptions.

The single core results show that, for both the linear and modlinear basis functions, the kernels perform better with 2 and 4 threads per core, as expected. While possible, choosing 3 threads per core is not in practice recommended, as the KNL architecture splits some of its internal workloads into quarters to be distributed among threads, and as such the best utilization of resources is to be expected for an even number of threads.

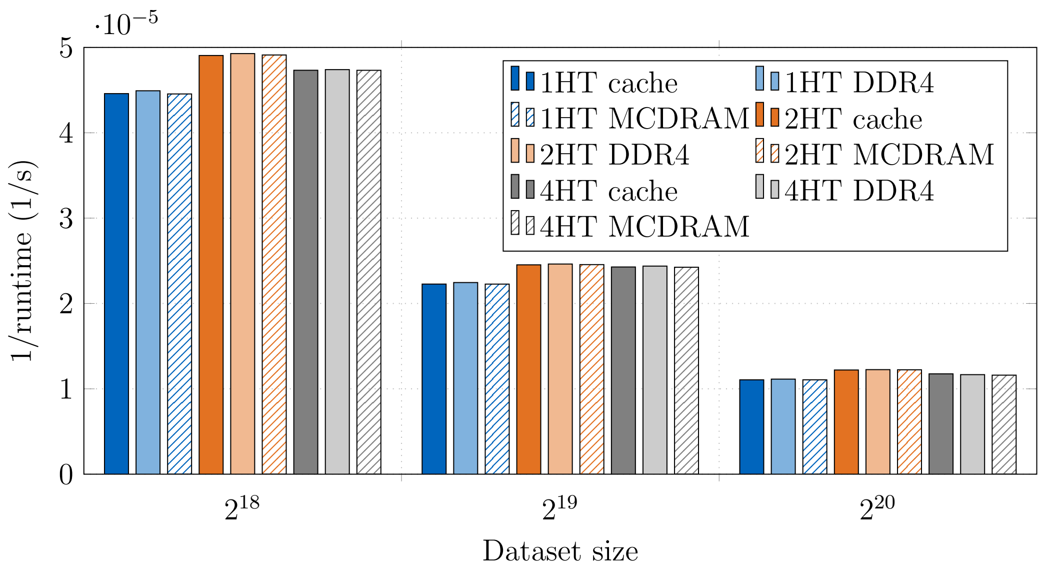


Figure 4.2: Single core results on KNL for the modlinear basis functions kernels. Higher values are better. The usual 3 memory modes are investigated (cache, flat with default DDR4 allocation, flat with explcit MCDRAM allocation), using the quadrant clustering mode. Between the three hyperthreading options, both 2 and 4 threads per core improve performance, with 2 proving to be optimal.

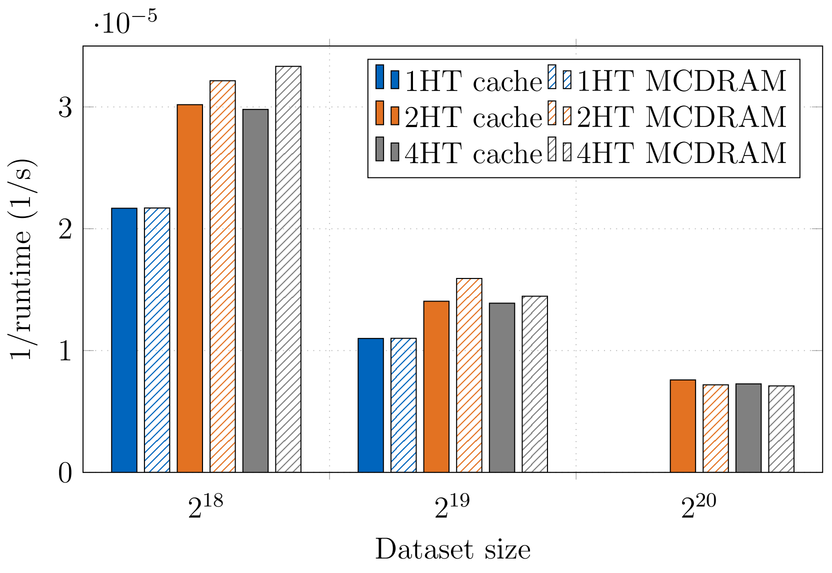


Figure 4.3: Single core results on KNL for the linear basis functions kernels. Higher values are better. Two memory modes are shown (cache, and flat with explicit MCDRAM allocation), using the quadrant clustering mode. Between the three hyperthreading options, both 2 and 4 threads per core improve performance significantly. Single thread runs for the largest dataset are missing due to runtime budget constraints on the CoolMUC-3 cluster.

Full node results. To provide more insight in the performance of the KNL node, we ran our three-sized classification problems for all possible core counts, with both 2 and 4 threads per core. We use the now usual quadrant clustering and the flat memory mode. To make sure we take into account the possible differences between the explicit HBM memory allocation (using the *numactl -p* command) and the default main memory usage, we ran all tests in both the flat DDR4 and flat MCDRAM settings. For the so-called "pure" MPI runs we kept the hyperthreads pinned to OpenMP threads and only applied the three different communication schemes across the cores. With the modlinear basis functions providing superior results in the existing study on older architectures, while also being the more challenging implementation, we also focused the remainder of the study only on this case. The most important results of these tests are shown in Figs. 4.4 and 4.5.

The best node performance was obtained using the pure OpenMP implementation. While threads are being pinned by the explicit affinity settings of the runs, we observe a worsening of the performance when using 4 threads per core which we can attribute to the added overhead and context switching between the worker threads. As expected, there is virtually no difference between the two flat memory modes due to the fact that the whole application (input dataset included) fits into the HBM. Using as a performance measure the unit of Gflop/s per core it is also easy to notice the almost perfect strong scaling characteristics of the compute kernels.

For the pure MPI results, we see that the communication overhead does not allow it to reach the performance level of the OpenMP runs. While more relevant for cluster level runs, our results also show the expected differences between the three communication schemes, with the Allreduce approach providing best. Again, using 4 threads per core proves to give a less stable performance when increasing the numbers of cores used. A slightly strange, yet repeatable, result is given by the noticeable drop in performance when using between 37 and 51 cores, across all three communication schemes, only when 2 threads per core are used. While this does not impact in any way the overall node performance or the conclusions drawn based on the data as presented thus far, it is still a peculiarity that might warrant further study in the future in order to determine more precisely its source.

Of noticeable importance was the realization of the quite significant performance hit incurred by the operating system (OS). While the first generation Xeon Phi took the form of the KNC coprocessor, where one would simply offload the computations to, and thus allowing for a full use of its computational resources, KNLs are full processors and therefore require to run some form of OS which manages the chip's operations. While on older architectures, like Haswell, these influences are less noticeable, our results on KNL show that here one has to dedicate up to a full core to these tasks, such that the best performance of the node level runs were obtained when using 63 cores of the node. From an operating point of view, this result is one of the reasons why the quadrant clustering mode was chosen as our default setting, as the SNC mode would have suffered even more from such an imbalance in the workload distribution on 63 cores. From a performance point of view, any hybrid OpenMP+MPI scheme would also suffer from this imbalance, as it would require

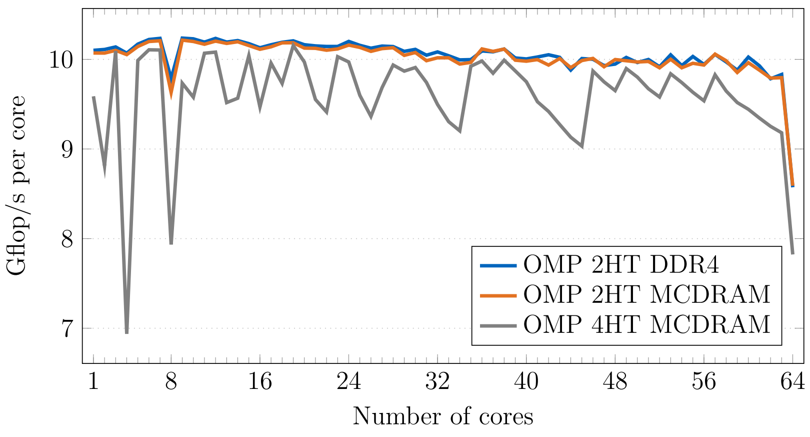


Figure 4.4: Node level results on KNL using pure OpenMP. We compare the 2- and 4-threaded runs, as well as confirming the equivalence of the two flat memory modes for applications that fit in the HBM. Using 2 threads per core gives better and more reliable performance across various numbers of cores. Using Gflop/s per core as a unit of performance one can better observe the very good strong scaling characteristics of the computational kernels. Runs were performed for the modlinear basis implementation, using quadrant clustering mode on an input dataset of size 2^{18}. We emphasize the differences in performance by showing a windowed scale between circa 7 and 10 Gflop/s. On the scale from 0 to 10 Gflop/s the two 2-threaded results would be indistinguishable.

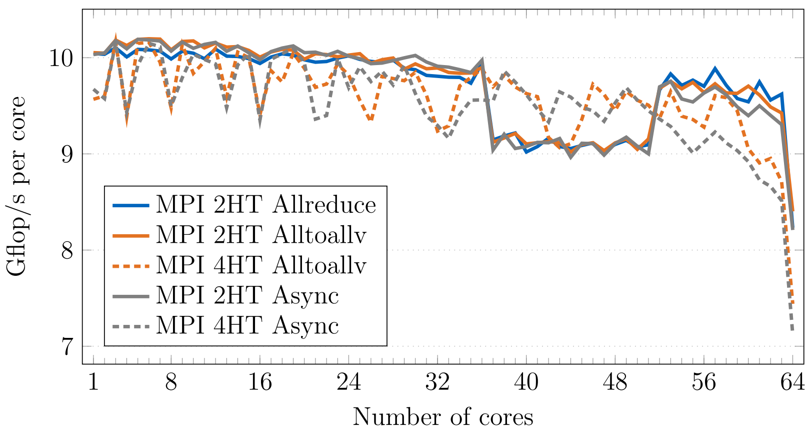


Figure 4.5: Node level results on KNL using pure MPI. We compare the 2- and 4-threaded settings, as well as the three different communication schemes. Results show a slightly worse performance compared to the pure OpenMP versions. As in that scenario, 2 threads per core deliver the best results, with the Allreduce approach coming ahead of the other two MPI schemes. Runs were performed for the modlinear basis implementation, using quadrant clustering mode and flat MCDRAM memory mode on an input dataset of size 2^{18}. We keep the same scale window as for the OpenMP results for a fair comparison. Again, on the full scale between 0 and 10 Gflop/s the 2-threaded results would be indistinguishable.

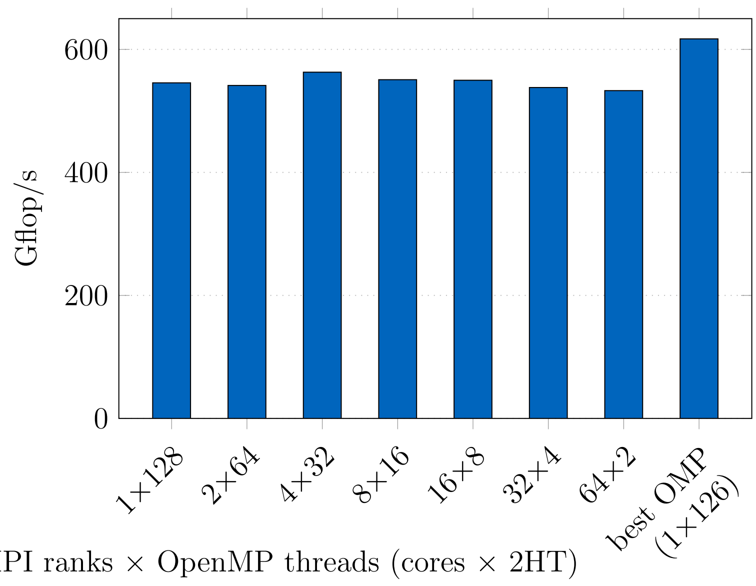
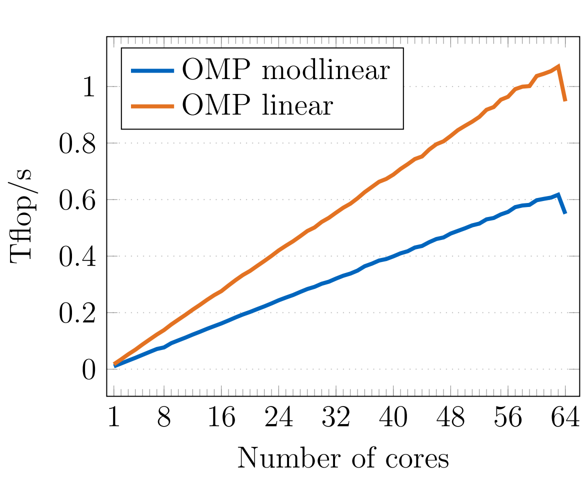


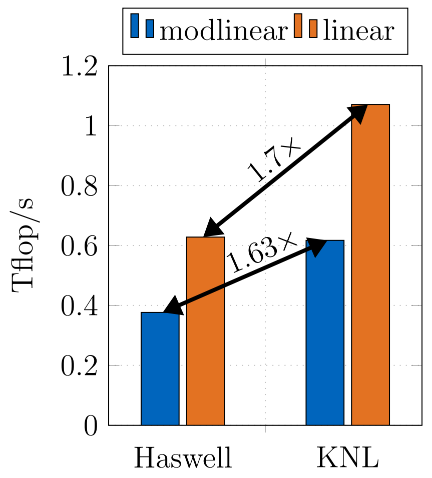
Figure 4.6: Node level comparison between the OpenMP+MPI (Allreduce) hybrid scheme and the optimal OpenMP version on KNL. All possible combinations of MPI ranks and OpenMP pinned threads are shown, with 2 OpenMP threads pinned per core. With the quite strong OS influences, the performance of the hybrid schemes, which run on the full 64 cores, suffers in comparison to the 63-core OpenMP run. All runs are for the modlinear basis functions implementation, using the quadrant clustering mode and the flat MCDRAM memory mode, on an input dataset of size 2^{18}.

the use of the full KNL die, which results in worse results than either the pure MPI or OpenMP versions (Fig. 4.6).

Node level performance discussion. In order to assess the performance gain obtained from our optimizations by comparing to previous architectures, we ran the same classification scenario with the best settings on both a Haswell node of the CoolMUC-2 cluster and a KNL node of the CoolMUC-3 system. For Haswell our runs recreated the existing results, with the linear basis functions implementation reaching around 50% of the theoretical peak performance, while the modlinear version brings that down to about 30%, both values being computed to take into account the influence of the Turbo Boost feature. The results on KNL see a small dip of these values proportionally to the chip's theoretical peak performance. This is however still in line with the existing results on other Intel CPU architectures, as well as those on different GPUs, which, like the KNL, suffer in performance when not enough FMA operations are available in the applications being run. In fact, on GPUs, results as low as 30% of the peak performance for the linear basis functions kernels has been observed [34]. Overall, we have obtained with our optimized code a performance increase on KNL versus the Haswell node of circa 1.7× for the linear



(a) KNL node scaling results for the best settings. Runs done for all core counts, using pure OpenMP, two hyperthreads per core, thread pinning with compact affinity, quadrant clustering mode and flat MCDRAM memory mode, for the modlinear basis functions version of the classification problem on a dataset of size 2^{18}.



(b) KNL speedups versus Haswell for the two versions of the optimized sparse grid classification computational kernels. We obtained a significantly high percentage of the theoretical peak performance speedup of 2.28× for the particular CPUs considered.

Figure 4.7: KNL node scaling results and speedups versus Haswell.

basis functions version and circa 1.63× for the modlinear case, out of a possible theoretical peak speedup of 2.28× (computed at base chip frequencies). These results are depicted in Fig. 4.7.

HBM limit results. Our last target for our KNL study was to investigate the performance capabilities of this architecture at the size limit of the HBM, where differences between the memory modes are expected to influence significantly the results. For this purpose we ran the classification problem on training datasets of 2^{26}, 2^{27}, and 2^{28} instances. To be noted that when talking about applications that fit or not in the MCDRAM we have to also consider the memory footprint of the code, as well as, in our classification scenario, the testing datasets of sizes 2^{22}, 2^{23}, and 2^{24}, respectively. (The powers of two are easier to handle and allow for a better interpretation of the results.) Therefore, our biggest dataset, at a total footprint of circa 20GB, lies slightly outside the HBM limit of 16GB, the second biggest scenario can fit comfortably in MCDRAM, and the smallest test case is meant as a control case, at a total memory footprint of circa 5GB. In order to obtain meaningful results (or any, due to the size of the datasets and runtime restrictions), we performed cluster level runs on multiple nodes, using the best node level settings (63×2 OpenMP threads in compact affinity, with the nodes running in quadrant clustering mode) and as many MPI ranks as nodes considered, using

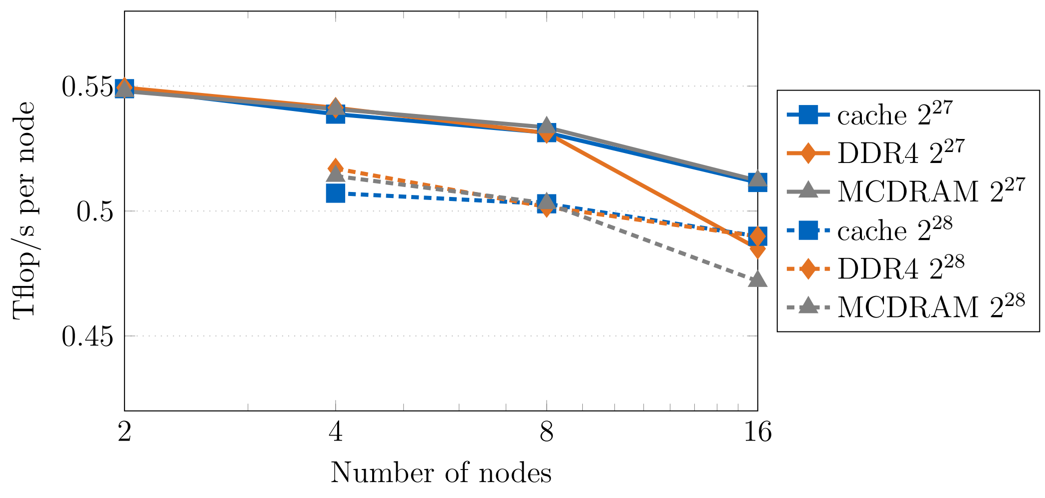


Figure 4.8: KNL results at the HBM size limit. In order to be able to fit in the CoolMUC-3's runtime limit we had to perform cluster level runs, i.e., using multiple compute nodes. Runs done using the best node level settings, i.e., quadrant clustering mode, 63×2 OpenMP threads per node with hyper-threading enabled, compact affinity thread pinning. For inter-node communication we use the Allreduce MPI scheme. Results show that as a general good practice rule one should use the flat memory mode, with the MC-DRAM allocation if the application fits into the HBM 16GB limit, and the default DDR4 allocation for larger problems.

the Allreduce communication scheme. The results for the two largest scenarios are depicted in Fig. 4.8.

Overall, with the correct memory allocation, both cache and flat modes deliver similar performance with increasing number of nodes. For datasets that fit in the HBM, all three settings tested perform on par for a low number of nodes, however the effects of the slower DDR4 memory become pronounced as we reach 16 nodes. On the other hand, the problem exceeding the HBM limit shows, as expected, an opposite effect. While the cache mode and the flat DDR4 allocation deliver equivalent results, the flat MCDRAM mode shows the worst performance, as this mode cannot accommodate fully all the application's memory footprint. Overall, we conclude that, even at the cluster level, ideally our classification should run in quadrant flat MCDRAM mode as long as the HBM limit is not reached, while for larger problem sizes the quadrant flat DDR4 mode is recommended.

Lastly, while the runs on multiple nodes were prompted by the runtime constraints such large input datasets imposed, we can also make a few remarks on the scaling behavior at the cluster level compared to existing results on older CPU architectures. While numbers are slightly lower on the KNL than on Haswell, the efficiency still drops very slowly with increasing node counts such that at 16 nodes we still obtain an efficiency of over 90%, which means that even without additional optimizations based on KNL features the Allreduce communication scheme still delivers great performance.

Discussion of results. Our work on porting the legacy code to the KNL processor resulted in further understanding of both this particular compute kernel, the previously chosen implementation details, but also the target hardware. Our test runs up to node level showed that this code ran optimal on two hyperthreads and 63 of the available 64 cores due to the nature of the hardware and the latencies in the AVX-512 intrinsics. We have also re-argued the main implementation decisions of Heinecke existing in the legacy code, now put also in the context of the more complex set of settings of the KNL architecture. Overall, we obtained a significant portion of the theoretical speedup compared to the Haswell architecture, with the flat MCDRAM quadrant setting proving to be the best for our data access patterns and kernel parallelization strategy.

Finally, larger tests, at the magnitude of the high-bandwidth memory, required runs on multiple nodes. Even though no extra work was performed to optimize the MPI communication schemes at the cluster level, as this exceeded our intended purpose, our tests showed that, firstly, good scaling still occurs when multiple nodes are involved with the existing schemes and, secondly, that, while naturally the decision on the storage location to be used (MCDRAM or DDR4) should be made based on the size of the problem to be solved, dataset sizes had little impact on the rest of the KNL settings to be used for optimal performance, namely quadrant and flat clustering and memory modes, respectively.

4.2 Application: Regression-based Time Series Prediction

Having a fast, optimized sparse grid solution to the classification/regression problem as described at the start of this chapter allows us now to explore also other related learning tasks. Specifically, we will present a study on the applicability of spatially adaptive sparse grids to the problem of time series prediction for large real-life financial datasets, where existing work only used non-adaptive grids under the Combination Technique approach. The motivation for our work was multi-fold. First of all, we test to what extent we can replicate those existing results under special circumstances. In our particular case, no access to the original dataset is possible, however an alternative equivalent dataset can be constructed from open sources. More details will be given when discussing the preprocessing step of the algorithm. Secondly, by applying the same method on similar data, we can give additional insights into new aspects of both the learning algorithm, as well as the type of input data itself. Lastly, with existing work focusing only on the easily parallelizable Combination Technique, we extend the applicability of this sparse grid-based algorithm to spatially adaptive grids by leveraging the short time-to-solution of the optimized regression code presented in the previous section.

4.2.1 The approach

Time series represent a very particular type of datasets, and as such also the learning tasks to be performed on them are not only specific, but also offer particularities associated to this type of input [16]. Out of all these problems we focus on the task of time series prediction, which can be defined as:

> Given a sequence of ordered values in time, called a *time series*, predict one or more functions of its future values (in time).

We have kept the definition quite vague and all-encompassing specifically because many variations exist. For example, the time series can contain values equidistant in time, at specific time intervals between them, or at random times. The prediction as well can be limited to finding one or more values at specific future times, or trying to predict as best as possible a continuous time interval in the future.

The study we recreate and extend is the time series prediction method using sparse grids of Garcke et al., as described in [24] and summarized in [25]. While similar to another approach using sparse grids [9], this method is designed to allow the handling of multiple related (or *coupled*) time series at once. The basic idea is to transform the input time series data into a feature space where a regression problem can be solved. Then our initial prediction task transforms into a simple evaluation of the learned regression function in the feature space.

Borrowing the notation in the original approach, one starts in this algorithm with a clean dataset of the type

$$\mathcal{T} \coloneqq \{t_j, f_r(t_j)\},\ j = 1, \dots, J,\ r = 1, \dots, R, \tag{4.2}$$

where f_r are R related time series, t_j are points in time, called *ticks*, with $t_{j+1} = t_j + \tau$ for some fixed time interval τ, and J represents the number of time steps available in all time series. In this context the task of prediction can then be redefined as using the values at ticks $[t_j - (K-1)\tau,\ t_j]$ of all time series f_r to approximate values at a future time $t_j + \hat{k}\tau$, where K and $\hat{k}$ denote fixed numbers of ticks. This approach of using a certain amount of past values of a time series is the main idea of the *delay embedding* approach. We now have $R \cdot K$ individual values at our disposal that can help us predict what happens at one future time for any of the time series. In practice, one can ether choose to use these time series values directly or transform them into more suitable and application specific values. While several options are proposed in the original work of Garcke et al., here we only present the one used for the financial dataset, i.e., the *normalized discrete first derivative*:

$$\tilde{f}'_{r,k}(t_j) = \frac{f_r(t_j) - f_r(t_j - k\tau)}{k\tau f_r(t_j - k\tau)},\ r = 1, \dots, R, \tag{4.3}$$

where k is a number of ticks, called the *back tick*, with $k\tau$ thus representing the discrete time step of this finite difference-based approximation. The value $\tilde{f}'_{r,k}(t_j)$ can be applied at K values, forming a set of 1D feature points we can use in our regression problem.

The target of the prediction also depends on the actual data used and its applications. For our finance scenario we try to predict not an exact future time series value, but the *normalized difference* of one of the time series, i.e.,

$$y(t_j) = \frac{f(t_j + \hat{k}\tau) - f(t_j)}{f(t_j)}. \tag{4.4}$$

The normalized discrete first derivative and this normalized difference can then be applied over the multiple input time series set $\mathcal{T}$ for various values of the back tick k, obtaining the dataset in feature space

$$\mathcal{S} = \left\{(\boldsymbol{x}_m, y_m) \in \mathbb{R}^D \times \mathbb{R}\right\}_{m=1}^{J-(K-1)-\hat{k}}, \tag{4.5}$$

where $\boldsymbol{x}_m = \left(\tilde{f}'_{1,k_1}(t_j + (K-1)\tau), \ldots, \tilde{f}'_{r,k_1}(t_j)\right)$, $\forall k$ are feature vectors and $y_m = y(t_j + (K-1)\tau)$ are the target values. The feature space dimensionality D is given thus as the product between the number of coupled time series used as input and the number of different back ticks k we employ for each time series.

Two more steps need to take place in order for this dataset to be a suitable input for our optimized regression algorithm. First of all, as with any data mining input data, one should try to minimize the number of outliers in the datasets. This is in general a non-trivial task, as the property of being an outlier is usually specific to the type of data one works with (currency exchange rates) and the properties of the algorithm it gets fed into (sparse grid regression). The previous work of Garcke et al. was not very specific in the procedure used, however we expect our approach, which we will describe here shortly, results in virtually the same outcome. We have noticed that the embedded data follows a pseudo-normal distribution aligned heavily along the axis, thus our outlier removal method was to remove those values lying in areas of low variance, so basically lying too far from the mean of the data along each of the dimensions. While we did this in our case manually, as the number of embedded datasets was small, this method can also use, e.g., a statistical approach to achieve qualitatively the same result, by means of a suitably bin-sized histogram in each dimension.

Lastly, once outliers are removed, the feature space needs to be transformed to the correspondingly-dimensioned unit hypercube, which is the input space required by our sparse grid algorithm. This is a quite simple procedure, done via a dimension-wise linear transformation of the data points from $[a_1, b_1] \times \cdots \times [a_D, b_d]$ to $[0, 1]^D$.

4.2.2 Processing the financial data

The work of Garcke et al. used as raw input data the *foreign exchange rate* between different currencies over a period of circa 4 years, from 01.08.2001 to 28.07.2005.

While in that study the data was acquired from a commercial data provider, where such historical data is available in a more processed and cleaned state, we had to rely only on open-source sources in order to obtain our data. In the following we will present the steps we took to clean and process the raw data in order to be suitable for the regression problem, pointing out along the way the differences and additions made to ensure that our numerical results could be compared to the ones in the work of Garcke et al. [24].

The data we had access to for the same period of time as in the existing work, comes from an online free foreign exchange database used by traders to run algorithms on[9]. The data is in the form of CSV files containing tick data on a precision of up to a millisecond, with each file containing a month's worth of data. Each line in the file contains the exact time of recording, two real values representing the bid and ask quotes, respectively, of the currency pair rate, and the trading volume. An example of such data would be:

```
20050703 170052000,1.194500,1.194100,0
20050703 170216000,1.194600,1.194200,0
20050703 170233000,1.194700,1.194300,0.
```

As we are not given the average bid value, which is the actual target needed for our purposes, we require to compute it. In order to efficiently process the large number of data points and files, we have written Python scripts that automate all these operations which we describe in this section. After computing the average, we need to only retain the values at specific tick intervals. In line with the work of Garcke et al., we are interested in intervals of $\tau = 3$ minutes. To maintain consistency across all currency pairs, which might have different initial tick data available, we compute de ticks starting with midnight in the first day of every month. For all such fixed ticks we take as our value the closest existing raw tick average bid, marking with zero (which is an invalid currency rate) the ticks for which no raw data exists. The reasons for such missing data are multiple: not all commercial and free data vendors have the same standards and reliability in reporting consistency of the values, during the weekends and public holidays there is no open market for which to report data, the daily trading schedule is also unequal during a regular work day due to differences in market volumes around the world. While missing data in itself is not that problematic for the regression problem, when we perform the transformation into the feature space we have to make sure values at all the data points involved exist, i.e., for tick t_j also the values at $t_j - k\tau$ and $t_j + \hat{k}\tau$ need to exist. Moreover, when considering as inputs more than one currency pair these ticks need to exist for all time series involved in order to obtain a valid feature data point. This process of obtaining the vectors $\boldsymbol{x}_m$ and targets y_m leads in any case to a reduction in the total number of data points in set $\mathcal{S}$ as compared to the size of $\mathcal{T}$. For our tests we used the currency pair rates between EU euro and US dollar (EUR/USD, denoted by €) and between British pound and US dollar (GBP/USD,

[9]The database can be found at: `http://www.histdata.com/`

Table 4.2: Statistics of the cleaned and preprocessed currency pairs tick datasets, before delay embedding is applied. Comparing to the similar statistics of the commercial datasets used by Garcke et al. [24], we can see that we have more missing tick data due to more gaps, some quite large, although the average gap sizes are smaller, meaning the gap size distribution is more heavily skewed. The existence of such gaps is to be expected and we also remain in our case with a significant portion of the ticks in order to obtain meaningful predictions from this data.

currency pair	total number of ticks	number of missing ticks	number of tick gaps	maximum tick gap	average tick gap
EUR/USD (€)	701280	274663	14378	5182	19.1
GBP/USD (£)	701280	288900	21277	5181	13.6

denoted by £), with the first one acting also as our prediction target, i.e., $y_m := {}_m$. Statistics for these two currency pairs tick datasets are presented in Table 4.2. We also maintain the strategy of splitting all embedded datasets into 90% training and 10% testing subsets, divided along the time axis.

To get an understanding of the underlying distribution of points in the feature space, Fig. 4.9 shows an example of how the foreign exchange rate data looks like for a single feature delay embedding (i.e., the 1D case).

4.2.3 Numerical results

One of the main purposes of our study was to investigate the suitability of using adaptive sparse grids in the context of regression-based time series prediction and as such we have not taken the task of recreating all the possible tests performed in the original study. Our results are restricted to show the main types of scenarios one would address in this context up to the point where one can draw conclusions on whether the sparse grid method produces beneficial outcomes or not, i.e., in the context of financial prediction, if we expect to obtain a profit or not. So, for example, we have excluded from our tests the more complex and very specific tradeable strategy using opening and closing thresholds for the trading signal, as it requires more significant changes to our otherwise generic sparse grid regression code we adapted. Secondly, we have also limited the number of currency pairs investigated due to the fact that existing results showed that an improvement, if it exists, should already be noticeable by using these particular two currency pairs. In general, we will specifically mention where and if we stray from the approach in the work of Garcke et al., and we refer those interested in more specific details and test scenarios to their study [24]. In particular, as we were not bound by the size of the dataset due to the fact that we used our fast, optimized code, we could forgo the need for any cross-validation step and perform all our tests directly on the full target dataset.

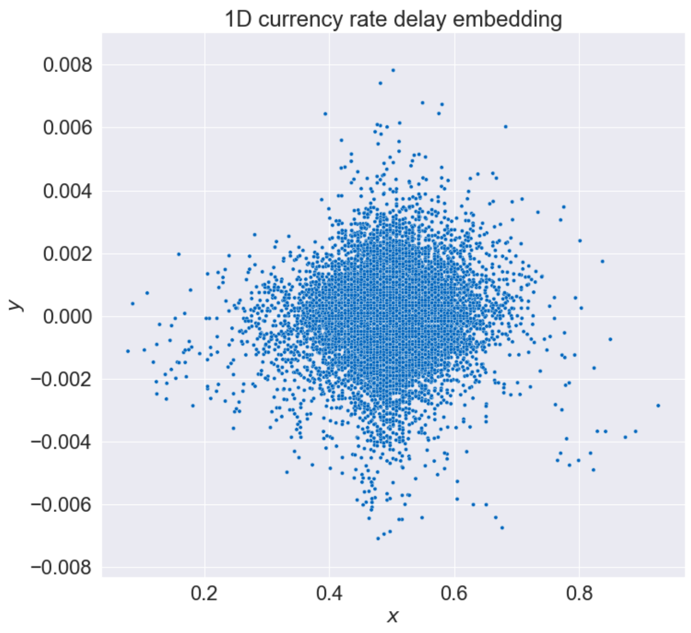


Figure 4.9: Example of a fully preprocessed test dataset for the 1D prediction task for the currency pair EUR/USD. This test dataset went through the steps of delay embedding, outliers removal, linear domain transformation to unit interval, and time-wise split (with the earlier 90% of the preprocessed feature space dataset being used as training data).

Quality measures. We employ the same prediction quality metrics as in existing works. They are, for a feature dataset of size M, a generic target currency rate f, and a learned sparse grid regression function u:

- The *cumulative profit* (cp), given by

$$cp := \sum_{m=1}^{M} \frac{\operatorname{sign}(u(\boldsymbol{x}_m)) \cdot (f(t_m + \hat{k}\tau) - f(t_m))}{f(t_m)} = \sum_{m=1}^{M} \operatorname{sign}(u(\boldsymbol{x}_m)) \cdot y(t_m), \tag{4.6}$$

  which denotes the net gain/loss predicted by our forecasting method.

- The *maximum cumulative profit* (mcp), given by

$$mcp := \sum_{m=1}^{M} \frac{|f(t_m + \hat{k}\tau) - f(t_m)|}{f(t_m)} = \sum_{m=1}^{M} |y(t_m)|, \tag{4.7}$$

  which is the actual overall gain one expects from a correct prediction.

- The *realized potential* (rp), computed as $rp := \frac{cp}{mcp}$.

- The *prediction accuracy* (*pa*), which is an otherwise general measure in forecasting tasks and which, in our context, is computed as:

$$pa := \frac{\#\{u(\boldsymbol{x}_m) \cdot (f(t_m + \hat{k}\tau) - f(t_m)) > 0\}_{m=1}^{M}}{\#\{u(\boldsymbol{x}_m) \cdot (f(t_m + \hat{k}\tau) - f(t_m)) \neq 0\}_{m=1}^{M}} = \frac{\#\{u(\boldsymbol{x}_m) \cdot y(t_m) > 0\}_{m=1}^{M}}{\#\{u(\boldsymbol{x}_m) \cdot y(t_m) \neq 0\}_{m=1}^{M}}. \tag{4.8}$$

The study of Garcke et al. states that favorable results, i.e., which would denote practical usefulness, are those for which one obtains a realized potential of at least 20% and a prediction accuracy of circa 55% or more. Thus, we also adopted these limit values as markers of successful predictions.

As the optimized sparse grid regression code we employ is a generic, non-application specific one, these quality measures had to be implemented in their own post-processing step. This was however quite a simple task due to the fact that, as can be noted from the given formulations, the quality measures can be rewritten in terms of evaluations of the resulting regression function at input points (i.e., $u(\boldsymbol{x}_m)$) and the target values of the regression problem (y_m), for which efficient vector operations were already present in our code.

Parameter search. The first step in our study is to determine suitable parameters for the back tick k and the future tick $\hat{k}$ for predicting the normalized difference of our target EUR/USD (€) currency pair. For this purpose we check different parameter combinations and asses the obtained realized potential on the test dataset in the one-dimensional case of using only one feature, i.e., $\boldsymbol{x}_m := x_m = (\tilde{€}'_k)_m$, running the code using linear basis functions (with boundary points) for our highest accuracy setting of level 4 grid and regularization parameter $\lambda = 0.0001$. Of course, we apply the strategy of choosing the parameters based on the strongest signals, i.e., the top 5% ticks with the highest absolute prediction value. The grid search results are shown in Fig. 4.10 and they are in line with the results on the commercial dataset [24]. While our data is quantitatively different, from the point of view of the prediction task it is qualitatively similar enough, with the same parameter pair $(k, \hat{k}) = (9, 15)$ giving the highest realized potential.

Non-adaptive single currency pair. With the prediction parameters set, we can start investigating the performance of the regression strategy when using a single currency pair. We ran tests on three different grid sizes and four λ values for regular sparse grids. We have additionally investigated also the results on modified basis functions, which, especially in higher dimensions, greatly reduce the amount of grid points one would need to invest by removing the need of boundary points. Results for the single feature scenario are compiled in Table 4.3. We obtain as best results: $cp = 0.22$, $rp = 0.87\%$, $pa = 50\%$. When we add the second feature for $k = 4$ (see [24] for further explanations on the reasons for this back tick choice), we observe, as expected, a steep improvement in the realized potential, but also a slightly improved prediction accuracy (Table 4.4).

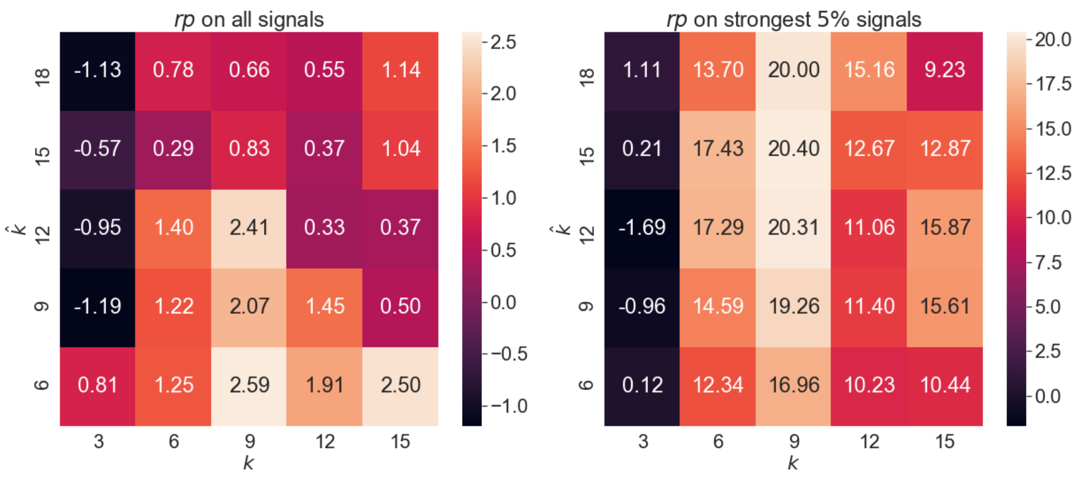


Figure 4.10: Realized potential (rp) obtained when predicting the currency pair EUR/USD using the normalized first derivative (i.e., the 1D case), for a level 4 grid and $\lambda = 0.0001$. While on all test ticks (left) values are higher for lower future ticks $\hat{k}$, we are interested in practical applicability, therefore we chose the best parameters by computing the rp values on the ticks that give the top 5% strongest predictions (right). This results in a parameter pair $(k, \hat{k}) = (9, 15)$.

In order to obtain practically useful results, we also consider the cases where we restrict our trading only on strong signals, which in this context represent predicted trading signals which are larger than 10^{-4} in absolute value. As can be seen in Table 4.5, this results in stark better values on our dataset, with the modified basis functions outperforming the linear ones. With realized potentials of over 31% at prediction accuracies of over 63%, this strategy of only trading on strong signals proves to be a very profitable option, irrespective of whether we learn on one or two features.

These numbers already provide some insight into our foreign exchange dataset. Compared to the values obtained by Garcke et al. on the commercially available tick data, in our case we observe a worse performance when considering all ticks, but improved results when only taking into account the strong signals. This means that in our case the concentration of valuable trading is far greater in the strong signals, while less trades actually take place.

Non-adaptive coupled currency pairs. We next explore the possibility of improvements brought on by the use of more than one currency pair. In the existing results we saw that this is a viable option, with different added currency pairs having larger or smaller influences. In our tests we chose to add the GBP/USD (£) data as it was shown previously to be one of the currency pairs that had a measurable beneficial influence in the commercial dataset.

Table 4.3: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ using the feature $\tilde{€}'_9$ (i.e., for back tick $k = 9$), on regular sparse grids of varying levels and regularization parameter values, for both linear (top) and modlinear (bottom) basis functions, when considering all ticks. Best level-λ combination is marked in each set of values, in terms of maximum realized potential values.

$€_9^{15}$ LBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	0.13	0.49	50.13	0.16	0.61	50.12	0.13	0.50	50.13	-0.48	-1.86	49.85
3	0.12	0.46	50.21	0.11	0.42	50.18	0.06	0.25	50.19	-0.40	-1.55	49.92
4	0.21	0.81	49.99	**0.21**	**0.84**	**49.99**	0.18	0.70	49.96	0.09	0.35	50.13

$€_9^{15}$ MBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	0.19	0.73	50.07	0.20	0.78	50.08	0.09	0.35	50.13	-0.46	-1.80	49.86
3	0.12	0.46	50.20	0.10	0.39	50.19	0.07	0.27	50.19	-0.45	-1.74	49.87
4	0.21	0.80	49.99	**0.22**	**0.87**	**50.00**	0.20	0.78	49.98	0.04	0.15	50.14

Table 4.4: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ using two features of the same currency: $\tilde{€}'_4$ and $\tilde{€}'_9$ (i.e., for back ticks $k = \{4, 9\}$), on regular sparse grids of varying levels and regularization parameter values, for both linear (top) and modlinear (bottom) basis functions, when considering all ticks. Best level-λ combination is marked in each set of values, in terms of maximum realized potential values. We observe a significant improvement in realized potential compared to using a single feature, with slightly better prediction accuracies (*pa*).

$€_{9,4}^{15}$ LBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	0.21	0.82	50.35	0.03	0.13	50.11	-0.06	-0.22	50.12	-0.40	-1.60	49.96
3	0.33	1.33	50.76	0.31	1.23	50.63	0.25	0.98	50.46	-0.14	-0.55	50.16
4	0.54	2.14	51.57	**0.58**	**2.31**	**51.50**	0.50	2.01	51.12	0.07	0.29	50.82

$€_{9,4}^{15}$ MBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	-0.13	-0.53	49.57	-0.15	-0.59	49.77	-0.05	-0.19	50.13	-0.42	-1.67	49.95
3	0.26	1.05	50.53	0.28	1.11	50.53	0.22	0.89	50.39	-0.29	-1.16	50.03
4	0.58	2.33	51.56	**0.57**	**2.27**	**51.45**	0.51	2.02	51.14	-0.06	-0.26	50.42

Table 4.5: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ using one (top) and two (bottom) features for the single currency pair case on regular sparse grids of varying levels and regularization parameter values, for both linear and modlinear basis, taking into account only the strong signals. Best results in terms of maximum realized potential for each set of values is marked in bold. We observe extremely favorable results in all 4 sets of values, well above those considered to be practically significant ($rp > 20\%$ at $pa > 55\%$).

€_{9}^{15} signal $> 10^{-4}$		$\lambda = 0.0001$			$\lambda = 0.001$		
basis	level	*cp*	*rp*%	*pa*%	*cp*	*rp*%	*pa*%
linear	2	0.07	20.68	63.30	0.05	16.01	62.19
linear	3	0.13	26.47	62.36	**0.13**	**27.57**	**64.11**
linear	4	0.24	23.10	58.39	0.19	20.71	58.01
modlinear	2	0.05	12.08	57.51	0.05	11.48	57.68
modlinear	3	**0.16**	**31.19**	**64.97**	0.14	29.45	64.74
modlinear	4	0.23	22.67	58.47	0.21	22.98	59.38

$\text{€}_{9,4}^{15}$ signal $> 10^{-4}$		$\lambda = 0.0001$			$\lambda = 0.001$		
basis	level	*cp*	*rp*%	*pa*%	*cp*	*rp*%	*pa*%
linear	2	0.12	30.04	64.03	0.05	20.61	60.53
linear	3	**0.20**	**36.66**	**64.42**	0.15	36.43	63.27
linear	4	0.27	28.74	61.22	0.24	27.67	60.81
modlinear	2	0.08	29.93	64.88	0.05	26.22	62.30
modlinear	3	**0.17**	**37.94**	**63.27**	0.12	28.87	60.49
modlinear	4	0.26	27.84	60.90	0.21	25.42	59.64

The first test scenario involves using the original best back and future tick pair $(k, \hat{k}) = (9, 15)$ for both datasets. For our free data source this did not result in a significant reduction of available feature space data points, as a similar number and distribution of gaps across the original tick data time span existed for both currency pairs. Again, we learn on the training data of 90% of data points in embedded space and test on the remainder of 10%. The results for this scenario when all ticks are considered can be seen in Table 4.6. We observe a significant improvement in the realized potential when we employ both basis functions, with *rp* values shy of 2% at a similar prediction accuracy rate of circa 50%. Next, we see whether this trend applies to using more than one back tick by again including also $k = 4$ in our embedding. This results in a 4-dimensional regression problem. The values in Table 4.7 show that indeed the realized potential improves in this scenario as well for both basis functions, reaching values of 2.7%.

Of course, these rates are not of any practical relevance, so we apply again the strategy of considering only the strong signals in our predictions ($> 10^{-4}$). These results are compiled in Table 4.8. We observe again that we obtain practically significant values, with $rp = 26.65\%$ at $pa = 58.72\%$ in the best scenario. In

Table 4.6: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ using the features $\tilde{€}'_9$ and $\tilde{£}'_9$ (i.e., for back tick $k = 9$ for coupled currency pairs), on regular sparse grids of varying levels and regularization parameter values, for both linear (top) and modlinear (bottom) basis functions, when considering all ticks. Best level-λ combination is marked in each set of values, in terms of maximum realized potential values. We observe an improvement of the *rp* rates at similar *pa* values compared to the corespondent single currency scenario.

$€^{15}_9$,$£^{15}_9$ LBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	0.21	0.91	49.64	0.18	0.75	49.61	0.16	0.68	49.55	0.02	0.08	49.85
3	0.30	1.26	49.88	0.28	1.18	49.79	0.24	1.00	49.74	0.11	0.44	49.79
4	0.42	1.79	50.05	**0.46**	**1.95**	**49.98**	0.44	1.87	49.87	0.35	1.46	49.90

$€^{15}_9$,$£^{15}_9$ MBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	0.04	0.19	49.32	0.08	0.32	49.38	0.08	0.32	49.41	-0.18	-0.76	49.85
3	0.26	1.08	49.82	0.29	1.24	49.82	0.22	0.92	49.71	0.09	0.40	49.86
4	0.42	1.76	50.00	**0.44**	**1.87**	**49.98**	0.44	1.84	49.92	0.22	0.93	49.87

Table 4.7: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ using two features of each of the two coupled currency pairs ($\tilde{€}'_4$, $\tilde{€}'_9$, $\tilde{£}'_4$, $\tilde{£}'_9$), on regular sparse grids of varying levels and regularization parameter values, for both linear (top) and modlinear (bottom) basis functions, when considering all ticks. Best level-λ combination is marked in each set of values, in terms of maximum realized potential values. We again observe an improvement of the *rp* rates at similar *pa* values compared to the corespondent single currency scenario.

$€^{15}_{9,4}$,$£^{15}_{9,4}$ LBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	0.27	1.18	50.23	0.19	0.85	50.04	0.08	0.37	49.82	-0.03	-0.15	49.93
3	0.44	1.92	50.65	0.45	1.96	50.60	0.40	1.75	50.45	0.36	1.56	50.13
4	0.56	2.47	51.14	0.50	2.19	51.09	**0.63**	**2.74**	**50.93**	0.45	1.96	50.47

$€^{15}_{9,4}$,$£^{15}_{9,4}$ MBF	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
level	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
2	-0.02	-0.07	49.42	0.02	0.09	49.55	0.05	0.20	49.72	-0.30	-1.30	49.82
3	0.45	1.97	50.47	0.41	1.81	50.47	0.34	1.50	50.18	0.27	1.20	50.08
4	0.49	2.13	50.95	0.52	2.30	50.95	**0.61**	**2.67**	**50.84**	0.43	1.87	50.49

Table 4.8: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ using one (top) and two (bottom) features for each of the two coupled currency pairs on regular sparse grids of varying levels and regularization parameter values, for both linear and modlinear basis, taking into account only the strong signals. Best results in terms of maximum realized potential for each set of values is marked in bold. While we still obtain practically significant results ($rp > 20\%$ at $pa > 55\%$), values are slightly lower than those for the single currency scenarios.

$€^{15}_{9}, £^{15}_{9}$ signal $> 10^{-4}$		$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$		
basis	level	*cp*	*rp*%	*pa*%	*cp*	*rp*%	*pa*%	*cp*	*rp*%	*pa*%
linear	2	0.12	15.78	56.71	0.09	13.56	57.14	0.02	5.87	54.49
linear	3	0.17	19.17	57.62	0.17	23.06	60.39	0.05	13.39	59.05
linear	4	0.25	21.86	57.90	**0.26**	**23.24**	**58.63**	0.13	16.62	56.62
modlinear	2	0.07	7.93	51.73	0.07	9.22	52.05	0.06	14.61	59.74
modlinear	3	0.11	17.69	57.80	**0.17**	**26.48**	**62.41**	0.11	22.87	63.57
modlinear	4	0.25	21.49	57.86	0.29	26.19	60.10	0.18	21.74	58.87

$€^{15}_{9,4}, £^{15}_{9,4}$ signal $> 10^{-4}$		$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$		
basis	level	*cp*	*rp*%	*pa*%	*cp*	*rp*%	*pa*%	*cp*	*rp*%	*pa*%
linear	2	0.21	23.53	57.95	0.12	17.47	57.32	0.02	12.82	52.00
linear	3	0.25	24.07	57.45	0.22	24.50	58.36	0.13	24.59	58.89
linear	4	0.22	18.32	55.58	0.27	23.69	56.86	**0.23**	**25.94**	**58.87**
modlinear	2	0.07	9.96	53.42	0.07	11.61	54.34	0.01	10.35	45.10
modlinear	3	0.17	19.54	55.89	0.20	26.33	58.16	0.11	21.60	58.24
modlinear	4	0.24	20.42	55.52	**0.29**	**26.65**	**58.72**	0.18	22.83	58.68

general, we again see that the modified basis functions obtain improved results over the linear one despite ignoring boundary points, which can be explained by the distribution of features in embedded space which favor in general sparse grids by being skewed alongside the dimension axes. However, compared to results for a single currency pair, it seems that the usage of our freely available tick data has finally shown its limitations, with our results showing a more than negligible influence of this second currency pair in the strong signals, where most of our profitable trading takes place. Of course, we cannot exclude as a reason for this result also the complex nature of exchange rates and market behavior, as there is no reason to expect always that an improvement in prediction on all ticks necessarily translates into one in the strong signals.

Adaptive single and coupled currency pairs. The original study of Garcke et al. did not touch on the aspect of grid adaptivity. By employing our optimized code we could fill in this gap, at least for the case of spatially adaptive grids, by running all our previous scenarios also in adaptive mode. To allow for a good comparison, our

Table 4.9: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ on adaptive sparse grids for varying regularization parameter values, for both linear and modlinear basis functions and all the single and coupled currency pairs scenarios, when considering all ticks. Runs use a level 2 starting grid and perform 3 refinement steps with 2^D refinement points per step, with D the dimensionality of the feature space. Best λ results are marked in each set of values, in terms of maximum realized potential.

$€^{15}_{9}$ ASG	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.12	0.46	50.20	0.22	0.85	50.00	**0.22**	**0.86**	**50.02**	-0.44	-1.72	49.88
modlinear	0.12	0.46	50.20	0.22	0.85	50.00	**0.27**	**1.07**	**50.18**	-0.45	-1.76	49.87

$€^{15}_{9,4}$ ASG	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.55	2.17	51.39	**0.59**	**2.34**	**51.07**	0.48	1.92	51.26	-0.14	-0.54	50.73
modlinear	0.52	2.07	51.27	0.33	1.30	50.88	**0.56**	**2.21**	**51.32**	-0.10	-0.42	50.34

$€^{15}_{9}, £^{15}_{9}$ ASG	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.11	0.47	49.74	0.20	0.84	49.66	**0.26**	**1.10**	**49.76**	0.12	0.49	49.60
modlinear	0.25	1.04	49.89	0.38	1.62	49.88	**0.53**	**2.21**	**50.08**	0.16	0.68	49.84

$€^{15}_{9,4}, £^{15}_{9,4}$ ASG	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$			$\lambda = 0.1$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.46	2.00	50.67	0.55	2.43	50.76	**0.59**	**2.58**	**50.93**	0.49	2.13	50.60
modlinear	0.52	2.29	50.86	0.44	1.91	50.90	**0.59**	**2.61**	**50.88**	0.52	2.30	50.59

strategy involves a starting grid level of 2 for all tests followed by 3 refinement steps using the strategy of surplus refinement, where grid points with highest absolute surpluses get priority, as this strategy has been seen to give good results in general. The number of grid points refined at each step is dependent on the dimensionality of the data and given by 2^D. This was chosen such that the resulting grids after the refinement would have as close as possible the number of grid points of a level 4 grid of that dimensionality. We ran our simulations for all values of λ as in previous tests and the results are shown in Table 4.9. When considering all ticks we see improvements on the single back tick results and similar values when 2 back ticks are considered, as compared to the non-adapting grid results. When moving to the actually practically relevant scenario of considering only the strong signals (Table 4.10), we obtain in almost all cases *rp* and *pa* values of practical relevance, with $rp = 30.72\%$ at $pa = 64.58\%$ and $rp = 25.8\%$ at $pa = 59.88\%$ being the best results for single, respectively coupled currency pairs.

Table 4.10: Results for the prediction of currency EUR/USD for $\hat{k} = 15$ on adaptive sparse grids for varying regularization parameter values, for both linear and modlinear basis functions and all the single and coupled currency pairs scenarios, when considering only strong signals. Best λ results are marked in each set of values, in terms of maximum realized potential. We again obtain practically significant results ($rp > 20\%$ at $pa > 55\%$).

$€^{15}_{9}$ ASG, signal $> 10^{-4}$	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	**0.15**	**29.41**	**63.17**	0.04	10.90	58.68	0.07	13.17	56.86
modlinear	**0.13**	**27.03**	**63.01**	0.07	16.85	60.51	-0.05	-59.79	20.59

$€^{15}_{9,4}$ ASG, signal $> 10^{-4}$	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.25	27.63	60.69	**0.17**	**30.72**	**64.58**	0.14	26.40	61.62
modlinear	**0.23**	**27.32**	**61.55**	0.14	26.30	59.51	0.13	24.44	59.63

$€^{15}_{9}, £^{15}_{9}$ ASG, signal $> 10^{-4}$	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.18	17.28	56.48	**0.16**	**18.59**	**58.08**	0.12	15.68	57.04
modlinear	0.21	19.43	57.62	**0.29**	**25.80**	**59.88**	0.20	22.35	58.70

$€^{15}_{9,4}, £^{15}_{9,4}$ ASG, signal $> 10^{-4}$	$\lambda = 0.0001$			$\lambda = 0.001$			$\lambda = 0.01$		
basis	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*	*cp*	*rp%*	*pa%*
linear	0.22	21.30	56.29	**0.24**	**25.36**	**58.17**	0.16	20.29	57.39
modlinear	0.27	23.33	56.89	0.25	22.98	55.66	**0.21**	**24.17**	**58.50**

Results summary and additional remarks. An overview of the best results across non-adaptive and adaptive grids for all feature space scenarios is provided in Table 4.11. Here we see more clearly that our smaller open-source tick data source still captures the same qualitative characteristics of the larger commercially available source dataset used in the study of Garcke et al. when employing non-adaptive sparse grids. While not investigated before in this financial time series prediction context, the modlinear basis functions prove that one can obtain just as good or even better results while not needing to invest in extra grid points on the domain boundary. This effect can be attributed mainly to the feature space data point distribution which favors sparse grids, something which can also explain why the adaptive grids could not bring any improvement on what regular grids could provide.

Now we want to address some of the aspects of the original study which were omitted by us here. From the beginning we have set our goals to investigate our scenarios up to the point where one can conclude or not that our additions to the original study could provide practical applicability. From this point of view, we

Table 4.11: Summary of the best results for the prediction of currency EUR/USD for $\hat{k} = 15$ on regular and adaptive sparse grids for all the single and coupled currency pairs scenarios. We distinguish between values when considering all test tick data and only the strongest signals. The modlinear basis functions clearly increase the performance of the prediction on our tick dataset, while using grid adaptivity did not bring any benefit in our test scenarios.

best non-adaptive	all ticks							
currencies	data points	basis	λ	level	trades	*cp*	*rp*%	*pa*%
€$_{9}^{15}$	348480	modlinear	0.001	4	38721	0.22	0.87	50.00
€$_{9,4}^{15}$	338201	modlinear	0.0001	4	37578	0.58	2.33	51.56
€$_{9}^{15}$, £$_{9}^{15}$	316185	linear	0.001	4	35132	0.46	1.95	49.98
€$_{9,4}^{15}$, £$_{9,4}^{15}$	299608	linear	0.01	4	33290	0.63	2.74	50.93
	signal $> 10^{-4}$							
currencies	data points	basis	λ	level	trades	*cp*	*rp*%	*pa*%
€$_{9}^{15}$	348480	modlinear	0.0001	3	380	0.16	31.19	64.97
€$_{9,4}^{15}$	338201	modlinear	0.0001	3	353	0.17	37.94	63.27
€$_{9}^{15}$, £$_{9}^{15}$	316185	modlinear	0.001	3	439	0.17	26.48	62.41
€$_{9,4}^{15}$, £$_{9,4}^{15}$	299608	modlinear	0.001	4	866	0.29	26.65	58.72

best adaptive	all ticks									
currencies	data points	basis	λ	level	num. of ref.	ref. pts.	trades	*cp*	*rp*%	*pa*%
€$_{9}^{15}$	348480	modlinear	0.01	2	3	2	38721	0.27	1.07	50.18
€$_{9,4}^{15}$	338201	linear	0.001	2	3	4	37578	0.59	2.34	51.07
€$_{9}^{15}$, £$_{9}^{15}$	316185	modlinear	0.01	2	3	4	35132	0.53	2.21	50.08
€$_{9,4}^{15}$, £$_{9,4}^{15}$	299608	modlinear	0.01	2	3	16	33290	0.59	2.61	50.88
	signal $> 10^{-4}$									
currencies	data points	basis	λ	level	num. of ref.	ref. pts.	trades	*cp*	*rp*%	*pa*%
€$_{9}^{15}$	348480	linear	0.0001	2	3	2	380	0.15	29.41	63.17
€$_{9,4}^{15}$	338201	linear	0.001	2	3	4	424	0.17	30.72	64.58
€$_{9}^{15}$, £$_{9}^{15}$	316185	modlinear	0.001	2	3	4	870	0.29	25.80	59.88
€$_{9,4}^{15}$, £$_{9,4}^{15}$	299608	linear	0.001	2	3	16	746	0.24	25.36	58.17

believe we have met this target. First of all, we have (re)validated the approach as a whole on this new dataset which was shown to possess similar qualitative properties as the commercially available data. Access to free good quality data is usually a problem for learning algorithms, so managing to replicate results on such datasets increases the confidence of future research and algorithms to be able to rely on sources like the one we have used in our work. Secondly, our additions to the approach of Garcke et al. have addressed two important questions left unanswered previously and which are of interest for any algorithm employing sparse grids, and with the ability of our optimized code to provide very fast time to solution, we were able to pursue these avenues by testing many scenarios in a fraction of the time it took in the original study. In terms of basis functions, when boundary points are considered one has to consider the possibility that their contribution could be insignificant. Our tests have shown that this is indeed the case, with the modlinear basis outperforming the linear one in most cases where trades are considered to be profitable, i.e., when strong signals are considered. The second sparse grid-specific aspect we have investigated is the use of adaptivity. Our tests showed that this does not seem to provide any benefit, and while the original study explored also higher-dimensional feature spaces, where adaptivity could be of more interest, the realized potential values obtained on the commercial tick data did peak in 4 to 6 dimensional feature spaces, and therefore our results can be seen as providing strong indications that adaptivity does not play a significant role. Of course, one cannot eliminate the possibility that a better, possibly application specific, refinement indicator could be of use in higher dimensions, however one has to also consider the added computational costs that grid refinement would impose.

Lastly, the work of Garcke et al. proposed also a more complex trading strategy, revolving around the value of cumulative profit per trade. While we have omitted this algorithm, we have enough results to make the educated assumption that on our dataset we can already obtain a significant profit margin even without this scheme. This is due to the fact that we already require fewer strong signals from our data, however at a cumulative profit per trade of between 3.3×10^{-4} and 4.8×10^{-4}, which is well above not only the desired margin of 8.5×10^{-5} (expected for the currency tick data we employ), but also previously reported values on the commercial dataset. Therefore, these strong signal trades we employ should already be enough to offset any trading costs without the need to further reduce the amount of capital invested, which was one of the main reasons given for the development of a more complex trading scheme.

4.3 Summary

In this chapter we have taken a closer HPC-oriented look at the tasks of classification and regression, which can be tackled in a unified way when using (spatially adaptive) sparse grids. While in the previous chapter we have dealt with mainly new algorithms, here we worked on existing codes and using established algorithms, with the intention to provide both new insights into newer hardware architectures, as well as leveraging optimized codes to fill in certain gaps in knowledge with respect to specific sparse grid-based data mining methods.

In Section 4.1 we have presented a code optimization study based on an existing legacy implementation on older CPU and GPU architectures. With multiple modes of operation, as well as its own set of hardware-specific intrinsic instructions, the target architecture of the KNL chip offered quite a large range of possibilities to cover and investigate in order to attain the fastest time to solution possible for our classification scenario. As such, we were able to obtain upward of $1.7\times$ speed-ups versus the previous Haswell CPU architecture at the node level. Moreover, we were able to also assess the best settings in which the KNL die runs most efficiently for datasets that cross beyond the high-bandwidth memory limit of 16GB, corroborating results and guidelines found in literature for similar applications.

With such a fast code for solving not only classification tasks, but also regression problems, we were able to provide in Section 4.2 a more in-depth analysis of the task of financial data prediction, previously only tackled on non-adaptive sparse grids using the Combination Technique. Going beyond the challenge of recreating the experimental setup using open-source as opposed to commercially available data and proving its viability, we have investigated two specific sparse grids aspects not yet tackled in this context. Our work showed that modified linear basis functions produce consistently superior results than the linear basis, and thus reducing the size of the required grids significantly by ignoring boundary points. On the other hand, adaptivity, in the form of surplus-based spatially-adaptive grids, was shown to not bring any significant improvements. Overall, we have reconfirmed the viability of the data mining method as a whole, added new insights in the possibilities offered by sparse grids, and the optimized code we used could open the door towards profitable financial trading strategies also in an online setting.

5 Contributions with a Usability Focus

The concern over the degree of usability of software is not something new, be it in generic open source projects [54] or numerical software in particular [56] (open-source or not). The development and usage of scientific software also takes a very significant part of the time of people working in science and engineering, but a lot of the knowledge acquired to do that actually falls mostly to self-study and information from peers [32]. Investing in the usability aspects of a software can thus also be seen not just as a way to improve current usage, but also improving the chances of a software being easily further developed and used after its initial developmental stages.

We have yet to actually properly define our terminology for this chapter. "Usability" has a very specific understanding in the software engineering and testing fields, where it acquired ISO standardization and where it is linked in practice almost exclusively to the user interface [87, 56]. While still rare, guidelines do exist for writing code with usability aspects in mind [44], especially for scientific software, which usually has its own special requirements [65]. Overall, what we will consider in this thesis as treating a numerical application from a usability perspective is the process of including non-functional, user-centered aspects in the design, development, and implementation of a numerical solution in order to better the experience of using, understanding, and further improving the code — in short, making life easier for both users and developers of the software.

In the previous chapters we have presented contributions driven mainly by aspects of algorithmic design and implementation and of high-performance computing, as they relate to some specific supervised learning tasks, where quality assessments of numerical results can be and usually are done exclusively by machine analysis, and therefore specifically user-oriented design and implementation requirements are seldom imposed or even considered. However, for unsupervised learning tasks, for example clustering, it is not only preferable, but almost obligatory in real-world applications for expert knowledge to be a part of the learning process, especially in the setup and analysis of the results. To this end, we consider particularly important for software solutions to unsupervised learning tasks to aid users in these critical aspects through user-oriented decision-making.

In this chapter, we will address the integration into and coupling with the existing SG++ data mining pipeline [69] of new sparse grid clustering approaches, with a focus on the degree of usability of the resulting software. Firstly, we will detail the integration of the original approach of Peherstorfer [61] for sparse grid density-based clustering, explaining how our design, development, and implementation choices were informed by a user-centered approach. Afterwards, two extensions to this standard method will be introduced, each with its distinct perspective on the issues of marrying new and existing code: full integration, in the case of our sparse grid hierarchical clustering algorithm, or development as add-on software, in the case of our sparse grid uncertain clustering method. For both extensions we will present various results on artificial and real datasets, showcasing the practicality of our approaches.

5.1 Clustering integration into the SG++ data mining pipeline

Our variant of the density-based clustering with sparse grids algorithm is intended to extend the current range of applications of the SG++ data mining pipeline of Röhner [69], taking full advantage of the existing functionalities, efficient algorithmic implementations, and a high degree of usability this piece of software possesses. In the following we will detail the main improvements and additions we have brought to the standard sparse grid clustering algorithm of Peherstorfer as we integrated this machine learning task into the SG++ data mining pipeline.

We note that the initial implementation of the following algorithms and integration into the pipeline was a result of a student collaboration [2], with several iterations of subsequent additions and improvements since.

5.1.1 The nearest neighbors algorithm

As presented, the naive approach has certainly some benefits when it comes to being parallelized. However, their quadratic complexity in terms of the number of data points ($\mathcal{O}(M^2)$) still becomes prohibitively expensive as the datasets increase in size. Space-partitioning methods can mitigate the data dependency aspect for lower dimensions, giving up to quasilinear ($\mathcal{O}(M \log M)$) or even linear ($\mathcal{O}(M)$) for low to moderate dimensions, but start to break down as the dimensionality reaches even moderate values like 10 [86]. Thankfully, there are alternatives. Hash- and tree-based methods seem to be more suited not only to provide very good quasilinear or linear behaviors in nearest neighbor problems, they can also be efficiently parallelized to scale well both in terms of dataset size and dimensionality [88].

Vantage-point trees. Our work on integrating clustering into the data mining pipeline, while not intended to produce at this stage the most efficient and parallelized code possible, was still done such that further optimizations could be subsequently integrated in future contributions, staying true to the high usability principles that guided the original development of this code. Thus, we have opted for a *vantage-point tree* (VP tree, for short) implementation.

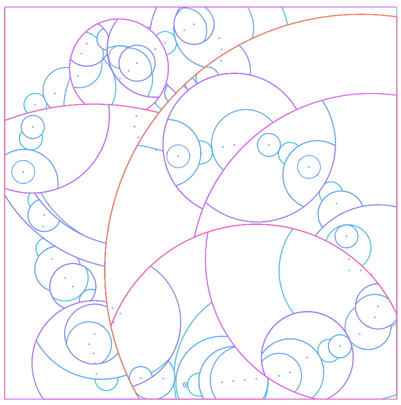


Figure 5.1: An example of 2D space partitioning based on a VP tree. Image from [20].

VP trees have been around for almost two decades already, since their introduction by Yianilos [90], and have become extremely used in various research areas like computational biology [14], image processing [52], computer vision [47], and algorithmic optimization [85], to name a few. The data structure falls in the category of metric trees, i.e., a structure that partitions data lying in (usually high dimensional) metric spaces. This allows them to be not only extremely efficient at accessing the data points, but also very flexible, as there is no innate restriction on the metrics we can use. In our case however we only require the use of the Euclidean distance. An example of a VP tree-based space partitioning is given in Fig. 5.1.

Building the tree. As most tree-based algorithms, the creation of the underlying data structure is done via a recursive algorithm. What makes it easier is the fact that a VP tree is a type of binary tree, thus we have always a fixed number (two) of children, or subtrees, to descend into.

The algorithm works as follows: at each recursion level we select one of the data points available there as a so-called “vantage-point”, storing it in the tree node along with the median distance from it to all the data points processed in the tree so far. Then the algorithm goes on to separate the remaining unprocessed data points into two subsets: those inside, respectively outside, a hypersphere, centered at the vantage-point, with a radius chosen such that these two subsets have approximately equal size. Lastly, we descend recursively into the two subsets, repeating this process until we have processed all points in the original dataset.

One aspects needs to be detailed further: the choice of vantage-point at each recursive step can be done at random (which leads to the overall $\mathcal{O}(M \log M)$ complexity of the tree construction), or using some data-based heuristic, which can lead sometimes to better performance. In our implementation we chose the random selection strategy, leaving it open though to be extended with possible heuristics as a future contribution if it will prove beneficial.

Algorithm 1: VP tree recursive builder

```
Input:
    dataMatrix – matrix storing all data points row-wise, with columns
    representing coordinates; indexing goes from 0 to $dataMatrix.size-1$
Result:
    root – root node of the newly created VP tree
-----------------------------------------------------------------
Procedure VpTree(dataMatrix):
    $root \leftarrow$ VpTreeBuildRecursive(0, dataMatrix.size)      /* starting
        the recursive algorithm */
Function VpTreeBuildRecursive(startIndex, endIndex):
    if $endIndex = startIndex$ then
        return NULL                          /* recursion end condition */
    $startPoint \leftarrow dataMatrix[startIndex]$
    $node \leftarrow$ CreateEmptyTreeNode()    /* alocate new empty node for
        this vantage-point */
    $node.index \leftarrow startIndex$          /* save the reference to the
        corresponding data point */
    if $endIndex - startIndex > 1$ then
        SortByDistance(startIndex, endIndex)   /* maintain points in
            sorted order by distance */
        $avgIndex \leftarrow \frac{1}{2}(startIndex + endIndex)$   /* get position where
            split will happen */
        $endPoint \leftarrow dataMatrix[avgIndex]$
        $node.threshold \leftarrow \|endPoint - startPoint\|_{L^2}$       /* store split
            Euclidean distance */
        $node.left \leftarrow$ VpTreeBuildRecursive(startIndex + 1, avgIndex)
            /* recursive call; left child contains points closer
            than node.threshold */
        $node.left \leftarrow$ VpTreeBuildRecursive(avgIndex, endIndex)
            /* recursive call; left child contains points farther
            than node.threshold */
    return node
```

In terms of actual code implementation strategies in the data mining pipeline, in order to maintain constant and unique references to data points throughout the clustering process, the tree nodes store the vantage points as index references to the corresponding data point coordinates stored in matrix format, therefore maintaining a constant complexity with respect to the dataset dimensionality. Additionally, the VP tree recursive builder function works not with the data directly, but with index intervals in a maintained sorted order based on distances between vantage-points and previously processed data points, such that we can easily define the subset splits described in the tree generating algorithm. All these steps are covered in the minimum pseudocode implementation of Algorithm 1.

Searching in the tree. Having the VP tree created, we can now look at how we can perform efficient searches of nearest neighbors. This is quite simple to perform recursively, considering we have introduced the convention that for each tree node its left subtree contains the near points. As the resulting structure is an almost balanced binary tree, the running time for any single tree query is in $\mathcal{O}(\log M)$, so finding the n-nearest neighbors is in $\mathcal{O}(n \log M)$.

The algorithm works again in a recursive manner. Starting with the root, we descend in a specific way down the tree in order to avoid the exhaustive search, ensuring the logarithmic complexity. This is done via a running maximum variable τ against which we compare the distance from the target point to each node's vantage-point. If smaller than τ, we add the point to the nearest neighbor list (from which we kick out the largest value in case we exceed by this the limit of n neighbors). We then proceed to go down the VP tree hierarchy if that is something possible or reasonable to do. Based on the relationships between the node's stored threshold, running maximum distance and the target-to-vantage-point distance, it can be that we descend into both subtrees, only one of them, or none, and with a preference to search either the left or right subtrees first. These aspects are made more clear in the minimum pseudocode implementation of Algorithm 2.

In order to improve the algorithmic complexity of the nearest neighbors search algorithm, a priority queue was used instead of a generic list, which ensures deletion of elements in $\mathcal{O}(1)$ instead of $\mathcal{O}(M)$, while insertions happen in $\mathcal{O}(\log M)$ with the elements being kept in order. The queue stores objects that point to a single VP tree node and allow comparison directly via the stored distance values computed during the search.

5.1.2 The graph operations

While the VP tree is a data structure used to efficiently query nearest neighbors, we still require to somehow store the resulting graph. Graphs are quite fundamental data structures and efficient implementations exist that allow all the normal operations one would require. While for the VP tree, which is a very specific type of graph, we required to have our own implementation, for the generic nearest neighbor graph needed to store the spatial network of data points we have opted to use the Boost Graph Library (BGL)[1] [75], which not only provides various algorithms that we need (e.g., finding connected components), but also allows customizations of the node and vertex objects. There is also a distributed extension to allow parallelizations of many of the algorithms[2], which could be considered for a future extension to our implementation.

[1]`https://www.boost.org/doc/libs/1_76_0/libs/graph/doc/index.html`
[2]`https://www.boost.org/doc/libs/1_76_0/libs/graph_parallel/doc/html/index.html`

Algorithm 2: VP tree recursive nearest neighbors search

Input:

targetPoint – point for which we want to find the n-nearest neighbors

n – number of nearest neighbors to search for in the VP tree

Output:

nnQueue – priority queue in which the nearest neighbors will be stored; the queue elements store the index referencing a data point, as well as its distance to *targetPoint*; it is updated in place

Data:

root – root node of the VP tree

```
Function VpTreeNearestNeighbors(targetPoint, n):
    $nnQueue \leftarrow$ CreateEmptyPriorityQueue() /* initialize queue */
    $\tau \leftarrow \infty$ /* initialize running maximum distance with infinity;
        in practice: maximum representable float value */
    VpTreeSearchRecursive(root, targetPoint, n, nnQueue, $\tau$)
        /* call recursive algorithm from the root node of the VP
        tree with an initially empty nearest neighbor queue */

Procedure VpTreeSearchRecursive(node, targetPoint, n, nnQueue, $\tau$):
    if $node = \text{NULL}$ then return /* recursion end condition */
    $dist \leftarrow \|dataMatrix[node.index] - targetPoint\|_{L^2}$ /* get distance
        from current vantage-point to target */
    if $0 < dist < \tau$ then
        /* if vantage-point is close to target, but distinct */
        if $nnQueue.size = n$ then
            /* if queue full, make room first by removing the
                farthest neighbor */
            nnQueue.pop()
        nnQueue.push(CreateNewQueueElement(nonde.index, dist))
            /* add new element to the queue */
    if $dist \leq node.threshold$ then
        /* if inside the vantage-point hypersphere, descend into
            close subset (i.e., left child) first */
        if $dist - \tau \leq node.threshold$ then
            VpTreeSearchRecursive(node.left,targetPoint,n,nnQueue,$\tau$)
        if $dist + \tau \geq node.threshold$ then
            VpTreeSearchRecursive(node.right,targetPoint,n,nnQueue,$\tau$)
    else
        /* if outside the vantage-point hypersphere, descend into
            far subset (i.e., right child) first */
        if $dist + \tau \geq node.threshold$ then
            VpTreeSearchRecursive(node.right,targetPoint,n,nnQueue,$\tau$)
        if $dist - \tau \leq node.threshold$ then
            VpTreeSearchRecursive(node.left,targetPoint,n,nnQueue,$\tau$)
```

Integration into the data mining pipeline required mostly the creation of wrappers for the classes and methods of the BGL in order to integrate into the design of the data mining pipeline. The more important aspect though was to decide the correct underlying containers that would store the data structures of the graph. For both vertices and edges we could have picked between vectors, lists, and sets. The choice made, of using a list to store vertices and a set to store edges, was based on the needs of our implementation and for higher code efficiency:

- Regarding vertices, our clustering approach requires mostly deletions, as we perform a graph pruning for each given density threshold. Lists offer the best performance for these operations compared to vectors, with a complexity in $\mathcal{O}(|V| + |E|)$, where with V and E we denote the sets of vertices and edges in the graph, respectively, and $|\cdot|$ is the set cardinality function. Also, compared to both sets and vectors, iterators in list containers are not invalidated after deletions.
- For edges, although we also perform mostly deletions, we were more concerned with spatial requirements, therefore we have opted for a set container, which better handles the case of undirected edges when constructing the nearest neighbor graph by automatically handling repeated entries.

To strike a better balance between computational effort and memory consumption, we have opted for additional mapping structures that would ease the burden of some of the traversal algorithms needed for accessing related graph elements. This was accomplished using two maps, a pointer-to-index and an index-to-pointer, which allows random access of vertices from edges, and vice versa, when required.

5.1.3 Cluster quality metrics

A main issue to be solved when dealing with the task of clustering is to assess the quality of the obtained clusters. The same way as there is little shortage of algorithms to perform clustering, there is also no definitive answer as to what makes a cluster quality metric a good one in the general sense. Of course, for very narrow and well-defined scenarios one would have a more clear understanding of what a "good" clustering should return, however such scenarios are rarely interesting from an algorithmic perspective. Widening the field, by, e.g., generalizing the type of input data one uses, and very quickly consensus on how to evaluate the results fades away. Work on creating a well-defined theoretical framework has had some limited success (e.g., through defining a system of axioms [7]), however this direction of study did not seem to have been pursued further since. The behavior of cluster metrics can also be influenced by the algorithms and the datasets themselves, as consistency with respect to variables like data dimensionality or number and structure of clusters is not really guaranteed [82].

Most researchers distinguish between two main types of clustering quality metrics [30]. Those based on *external* criteria require the comparison against a predefined labeling considered to be definitive (e.g., created by an expert through manual labeling). There are of course obvious advantages to such a metric, as it allows clear comparison of clustering algorithms on predefined datasets. However, there is also no guarantee that a good behavior on some predefined, labeled dataset translates into an overall good performance on new, even when similar, data. This is also done under the assumption that the labeling we consider as definitive is neither prone to human error nor too specific to the data context to be unhelpful for generalization. These metrics contrast thus to the ones using an *internal* validity criterion. In this case we exclude the direct human contribution, with the quality evaluation using only values intrinsic to the input dataset. Some downsides of such methods are that they do not necessarily correlate with the amount of information one could obtain from the data (the concept of information retrieval [51]) and that they could be biased towards specific types of clustering algorithms.

A third type of clustering quality metric has been put forward. Using a *relative* criterion [31], one compares the results of a clustering with those of other algorithms, finding the one that satisfies best some predefined criterion.

With all these considerations in mind, the approach in our implementation was two-fold: on the one hand, creating the framework to allow for the implementation of any number of future metrics, and on the other hand, providing already a nice array of options that a user could choose from such that a better analysis of the clustering results could be performed. As we have basically one type of algorithm, using relative metrics is not possible. We have thus implemented instead two internal and two external quality measures proposed in literature, which will be detailed in the following.

V-Measure. [66] As stated in the paper introducing it, the *V-Measure* (VM) is an external "entropy-based" metric [66]. What is meant by this is that, similarly to the concept in physics and cosmology, it measures some form of disorder in the system, or, in our case, in the clustering. This is accomplished by measuring two aspects: *homogeneity* and *completeness*. The first gives the degree to which each cluster has data points belonging to a single class label, while the second measures to what extent all points of a certain class label get clustered in one cluster. At the extremes, a trivially homogeneous clustering would assign each data point to its own cluster, and a trivially complete clustering would return a single cluster containing the entire dataset. The VM is an approach to mediate these two opposing tendencies.

In order to define mathematically the metric, we require to introduce some notations, which will be similar to the ones in its initial publication [66]. Let $\mathcal{C}$ be our clustering (i.e., the set of obtained clusters in our algorithm), $\mathcal{L}$ the set of true class labels, and M the size of the dataset. Then we can define by $a_{l,c}$ the number of data points with true label $l \in \mathcal{L}$ that are in cluster $c \in \mathcal{C}$. Then:

- The *maximum entropy reduction* of the clustering is

$$\mathrm{H}(\mathcal{L}, \mathcal{C}) = -\sum_{l\in\mathcal{L}} \sum_{c\in\mathcal{C}} \left(\frac{a_{l,c}}{M} \log \frac{a_{l,c}}{M} \right). \tag{5.1}$$

- The *homogeneity score* is given by

$$\mathrm{HS}(\mathcal{L},\mathcal{C}) = \begin{cases} 1, & \text{if } \mathrm{H}(\mathcal{L},\mathcal{C}) = 0 \\ 1 - \dfrac{\mathrm{H}(\mathcal{L}|\mathcal{C})}{\mathrm{H}(\mathcal{L},\mathcal{C})}, & \text{otherwise} \end{cases} \tag{5.2}$$

  where

$$\mathrm{H}(\mathcal{L}|\mathcal{C}) = -\sum_{c\in\mathcal{C}}\sum_{l\in\mathcal{L}}\left(\frac{a_{l,c}}{M}\log\frac{a_{l,c}}{\sum_{l'\in\mathcal{L}} a_{l',c}}\right). \tag{5.3}$$

- The *completeness score* is given by

$$\mathrm{CS}(\mathcal{L},\mathcal{C}) = \begin{cases} 1, & \text{if } \mathrm{H}(\mathcal{L},\mathcal{C}) = 0 \\ 1 - \dfrac{\mathrm{H}(\mathcal{C}|\mathcal{L})}{\mathrm{H}(\mathcal{L},\mathcal{C})}, & \text{otherwise} \end{cases} \tag{5.4}$$

  where

$$\mathrm{H}(\mathcal{C}|\mathcal{L}) = -\sum_{l\in\mathcal{L}}\sum_{c\in\mathcal{C}}\left(\frac{a_{l,c}}{M}\log\frac{a_{l,c}}{\sum_{c'\in\mathcal{C}} a_{l,c'}}\right). \tag{5.5}$$

- The V-Measure (VM) value is then computed as

$$\mathrm{VM}_\beta(\mathcal{L},\mathcal{C}) = \frac{(1+\beta)\cdot\mathrm{HS}(\mathcal{L},\mathcal{C})\cdot\mathrm{CS}(\mathcal{L},\mathcal{C})}{(\beta\cdot\mathrm{HS}(\mathcal{L},\mathcal{C}))+\mathrm{CS}(\mathcal{L},\mathcal{C})}, \tag{5.6}$$

  with the trade-off between homogeneity and completeness being controlled with the parameter β. For our purposes we have chosen a value of $\beta = 1$, meaning an equal weighting for the two components, resulting in the harmonic mean of HS and CS. The VM is a real number in the range $[0, 1]$, with higher values signaling a better clustering.

Fowlkes-Mallows Index. The *Fowlkes-Mallows index* (FMI) [19] can actually be considered from a relative validity perspective, comparing two (hierarchical) clusterings, but we will use it as an external metric, comparing instead our algorithm's results with a pre-defined set of labels. Using the same notations as introduced for the V-Measure, the FMI is computed as:

$$\mathrm{FMI}(\mathcal{L},\mathcal{C}) = \frac{\mathrm{T}(\mathcal{L},\mathcal{C})}{\sqrt{\mathrm{P}(\mathcal{L},\mathcal{C})\cdot\mathrm{Q}(\mathcal{L},\mathcal{C})}}, \tag{5.7}$$

where

$$\mathrm{T}(\mathcal{L},\mathcal{C}) = \sum_{lin\mathcal{L}}\sum_{cin\mathcal{C}} a_{l,c}^2 - M, \tag{5.8}$$

$$\mathrm{P}(\mathcal{L},\mathcal{C}) = \sum_{lin\mathcal{L}}\left(\sum_{cin\mathcal{C}} a_{l,c}\right)^2 - M, \tag{5.9}$$

$$\mathrm{Q}(\mathcal{L}, \mathcal{C}) = \sum_{cin\mathcal{C}} \left(\sum_{lin\mathcal{L}} a_{l,c} \right)^2 - M. \tag{5.10}$$

The value of the metric is a real value in the range $[0, 1]$, again with a higher value indicating a better match to the reference clustering.

Davies-Bouldin Index. The first internal metric we have implemented is a centroid-based approach. The *Davies-Bouldin index* (DBI) [13] uses cluster scattering to determine how similar clusters are to one another, with an ideal clustering having as dissimilar clusters as possible. In this context, the measure of scatter in a cluster $\mathcal{C}_i$, $i = 1, \ldots, |\mathcal{C}|$ is computed as

$$\mathrm{S}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\boldsymbol{x} \in \mathcal{C}_i} \|\boldsymbol{x} - \boldsymbol{z}_i\|, \tag{5.11}$$

where $\boldsymbol{z}_i$ is the centroid (i.e., geometric mean) of the data points $\boldsymbol{x} \in \mathcal{C}_i$. Denoting by $d_{i,j} = \|\boldsymbol{z}_i - \boldsymbol{z}_j\|$ the distance between the centroids $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ of two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$, respectively, the DBI is given by

$$\mathrm{DBI} = \frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} \max_{\substack{j=1,\ldots,|\mathcal{C}| \\ j \neq i}} \frac{\mathrm{S}_i + \mathrm{S}_j}{d_{i,j}}. \tag{5.12}$$

This metric is a real value lying in the range $[0, \infty)$, where lower values denote a better clustering.

Caliński-Harabasz Index. Known also as the variance ratio criterion, the *Caliński-Harabasz index* (CHI) [12] is an internal metric that measures the ratio between inter-cluster separation and and intra-cluster cohesion. Keeping with the notations introduced so far, for a dataset $\boldsymbol{x}_{j\,j=1,\ldots,M}$ split into clusters $\mathcal{C}_i$, $i = 1, \ldots, |\mathcal{C}|$ by our clustering $\mathcal{C}$, the CHI is given by

$$\mathrm{CHI} = \left(\frac{1}{|\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} |\mathcal{C}_i| \|\boldsymbol{z}_i - \boldsymbol{z}\|^2 \right) \Big/ \left(\frac{1}{M - |\mathcal{C}|} \sum_{i=1}^{|\mathcal{C}|} \sum_{j=1}^{|\mathcal{C}_i|} \|\boldsymbol{x}_j - \boldsymbol{z}_i\| \right), \tag{5.13}$$

where again by $\boldsymbol{z}_i$ we denote the centroid of the cluster $\mathcal{C}_i$, and by $\boldsymbol{z}$ we represent the centroid of the whole input dataset. The numerator of CHI is the weighted between group sum of squares, which is also the trace of the weighted covariance matrix of cluster centroids as a measure of separation, while the denominator is the weighted withing group sum of squares, or the trace of the weighted sum of each cluster's covariance matrix.

The CHI is a real value in the range of $[0, \infty)$, with higher values indicating a better clustering. Note then that, as opposed to the previous measures, this index does not provide an exact value that would constitute, in its view, an ideal clustering. Instead, the CHI is used in practice to compare different clusterings against each other. In the context of our density-based sparse grid clustering, this metric could therefore be a good option as a quality measure for the choice of the density threshold that decides the cut into clusters.

5.1.4 Visualization options

Another addition coming from the integration of the clustering algorithm into the data mining pipeline is the possibility to obtain a visualization of the results, part of which was already existing, implemented in previous work [1]. With the actual clustering happening subsequent to the density estimation, we have moved the visualization calls to the post-processing class created to handle operations (like the graph-based ones) that are not directly sparse grid-based. Additionally, we have adapted the existing scatter plotting capabilities of the pipeline to handle data represented in graphs, with color-coded connected components representing the different clusters. The treatment of higher-dimensional data is done, as in the case of scatter plots in density estimation, using first a *t-distributed stochastic neighbor embedding* (t-SNE) [35] algorithm to reduce the dataset to a 2D representation that can be easily visualized. Of course, some information is being lost with this embedding, however we also store all the results in *JavaScript Object Notation* (JSON) format[3], allowing it to be easily imported and visualized with another dedicated software when needed.

We have to make two remarks regarding our use of the t-SNE algorithm. First of all, it is not the only option possible to allow a lower-dimensional embedding for visualization purposes. The *principal component analysis* (PCA) algorithm is also an extensively used approach to achieve similar results. However, as a variant of t-SNE already existed in the pipeline [1], we have opted to reuse it for this purpose as well. Secondly, the current implementation in the pipeline is far from ideal in terms of performance. However, the fact that the pipeline allows for these kind of algorithms to be treated as plug-ins, future contributions can be directed also exclusively for speeding-up this part of the visualization process. For the cases we will look at, with moderately high dimensions and not very large datasets, the current implementation suffices.

5.2 Hierarchical Clustering

The first, and most important, contribution to the regular clustering algorithm was motivated by the major parameter and algorithmic bottleneck of the density-based approach, which is the choice of a suitable density threshold. Already from its introduction, this parameter was shown to influence significantly the clusters we obtain [61]. One option would be to automatically detect a threshold by means of cross-validation using some predefined quality metric. We have opted instead

[3]`https://www.json.org/json-en.html`

to give the user more freedom in choosing some good thresholds themselves. This was done for two reasons. First of all, it is not clear for a dataset what would constitute a good quality measure. Of course, implementing such an automatic threshold choosing approach is not complicated once a metric is given (e.g., splitting the dataset close to a specific number of clusters, or optimizing for a certain fixed quality metric from the ones presented already). The pipeline does allow such a method to be implemented subsequently, as a future contribution, if one so chooses. Secondly, we wanted to facilitate the user to visualize a higher number of possible clusterings to choose from before making a more in-depth analysis for certain thresholds, instead of focusing only on a specific option, allowing for an overall more informed decision process. Once the user decides on a specific threshold for example, a follow-up analysis with the usual clustering approach can always be performed.

Again we note that the following shows the current state of the algorithms in the data mining pipeline, with an initial implementation being the result of a student collaboration [2].

5.2.1 Theory and implementation

The idea. Originally proposed in a separate implementation of the sparse grid clustering algorithm [18], the idea is to expand the regular clustering algorithm to use not just one density threshold, but multiple ones. For each we perform the usual operations of removing vertices and edges from the nearest neighbor graph with density values below the respective threshold and finding clusters as connected components. However, the clusters obtained for each threshold can be linked in parent-child relationships to those found with previous threshold values, forming a tree of clusters.

Thus, our main contribution lies in the implementation of such a sparse grid density-based *hierarchical* clustering method in the data mining pipeline, which extends significantly the capabilities of the regular clustering approach. In the following we will look at the various components of this algorithm.

The clustering tree. In the hierarchical clustering method each found cluster is treated as a node in a tree structure, with the root node containing the whole dataset as an initial cluster. The first (non-zero) density threshold gives by the usual approach a certain number of clusters, which in our algorithm become child nodes of the root. Each subsequent density threshold creates new clusters which are then linked to their corresponding parents in the previous density level, which can be done easily due to the fact that for density-based approaches clusters at higher thresholds are subsets of those at lower thresholds, as can be seen from our example in Fig. 5.2a.

For each node we store, first of all, the cluster information, formed by the indices of the data points that are part of the cluster, along with the label assigned to it

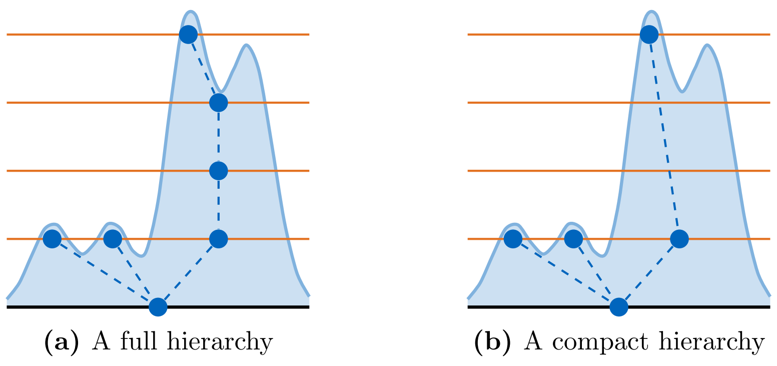
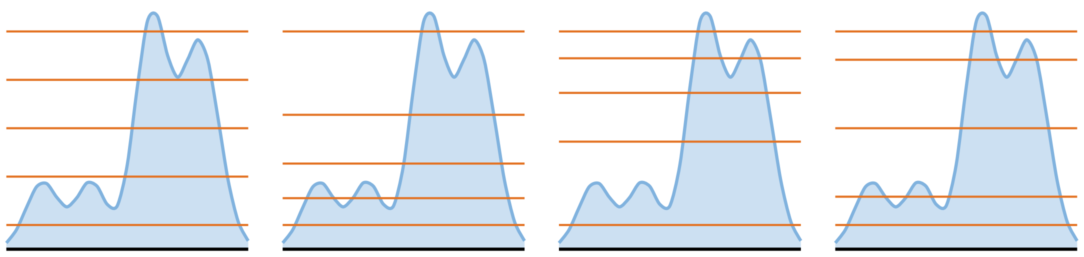


Figure 5.2: Examples of resulting (upward) tree structures from the use of 4 density thresholds in our hierarchical clustering method. To ease understanding we ignore the noise cluster. As can be seen easily from this 1D example, clusters at higher threshold values become smaller based on density width, (or diameter, in higher dimensions) and are subsets of clusters obtained at lower threshold values. In our approach, the root cluster contains the initial dataset. To produce a more robust hierarchy **(b)**, our method by default skips single-child clusters from the final result, obtaining a less deep tree structure. The full hierarchy **(a)** can however still be recovered in our implementation if needed.

and the density threshold at which it can be found. Additionally, we store tree-related information, like the hierarchical level in the tree and pointers to the direct parent cluster and to the children clusters (if any). These latter pointers are filled in retrospectively as soon as new clusters are found and their hierarchical parents are determined. In terms of storage, the overhead of this hierarchical structure is in $\mathcal{O}(1)$ with respect to the size of the dataset, making it negligible.

The density thresholds. As mentioned, the whole idea of a hierarchical clustering relies on the usage of more than one density threshold. These are provided by the user indirectly by means of parameters. First of all, we require a minimum and a maximum density to set the range of values we want to investigate. As we cannot know a priori the range of amplitudes of densities obtained using SGDE, the values are in $[0, 1]$ as relative to the maximum density of the dataset. Additionally, we require a number of steps to take between the minimum and maximum thresholds, as well as a thresholding distribution. Fig. 5.3 shows how these distributions we provided allow to better investigate different regions of the estimated density.

Handling the nearest neighbor graph. To strike a balance between storage and performance, we maintain at all times two nearest neighbor graphs, one with the state before the current processed density threshold, and one with the state after the current pruning step. This is done to facilitate the identification of hierarchical parents of the newly detected connected components. However we again work with



(a) Linear **(b)** Exponential **(c)** Logarithmic **(d)** Sigmoid

Figure 5.3: Exemplification of the different thresholding distributions implemented for hierarchical clustering. The linear thresholding is simplest, but it might be insufficient for certain scenarios, and investigating closer together thresholds would require an unnecessary increase in total number of steps, thus time to solution. Especially in high dimensions, it was noted that SGDE performs better for clustering purposes when we allow less regularization, which means more peaks and valleys, therefore focusing on one of those (peaks for mode finding, valleys for higher granularity and more clusters) could be helpful, especially in cases where we strive to find clusters of different homogeneity. The sigmoid thresholding tries to marry the exponential and logarithmic cases, by focusing more thresholds in the extremes of the thresholding interval.

indices referencing data points, therefore the added complexity is only in terms of the number of data points, not also in terms of the dataset dimensionality.

The split threshold and a compact hierarchy. A second idea stemming from the work in [18] is that of a splitting threshold that would decide whether we want to keep a newly found cluster at a certain density threshold in the clustering tree, or have it be replaced by its children instead. The motivation for such an endeavor would be to provide better insight into the datasets, as such an approach would help with narrowing down the positions of modes in the data, at the same time reducing the complexity and depth of the resulting clustering tree, implicitly reducing the number of total clusters identified. Based on a metric we compute from the nearest neighbor graph (which we will introduce in the following), we assess for each cluster from the previous step that has multiple children whether the split condition is fulfilled or not with respect to any of its children. If yes for at least one child cluster, then the split takes place and the parent cluster is replaced in the hierarchy, meaning that its children now belong to its parent instead. For clusters from the previous step that have only one child, we do nothing, such that the resulting hierarchy is more compact and the retained clusters are more meaningful (Fig. 5.2b).

Our version of the metric used to determine whether such a split happens or not is given by a ratio between two connectivity coefficients, one relating how connected a child cluster is to its parent, and the other how well-connected the parent cluster is with itself. While in [18] this metric is geometry-based, defining a connectivity between populated spatial cells, as distances start loosing meaning with increasing the dimensionality of the data, we opted for a graph-based approach instead. Our version of the criterion, which is shown in the minimum pseudocode implementation Algorithm 3, is relatively simple to understand and also behaves in a manner suitable to the clustering task at hand, which is revealed by a closer inspection of the formula we employ.

For a fixed size of the child cluster, our metric requires more connectivity as the number of parent cluster vertices increases. After a while, the only way the split happens is if the child-parent connectivity is extremely loose and the size of the child cluster is also significant. This is actually what we would like: the split is supposed to signal that the parent cluster has no significance by itself (e.g., as a low density cluster), but is just the result of an overlap of child clusters, so it can be safely removed from the hierarchy. Moreover, especially with the increase of the dataset dimensionality, the actual connectivity in an n-nearest neighbor graph would tend less and less to the maximum possible value with the growing size of a cluster, which would lead naturally to a higher splitting metric and thus by itself a lower chance of splits taking place. This is again a wanted effect, as it is known that in higher dimensions distances and local densities become less discernible and thus allowing splits to take place could lead to removing otherwise interesting, lower density, clusters from the hierarchy. While there are results that show that we can still recover a good clustering by means of purposeful overfitting by allowing the regularization value *lambda* in the SGDE estimation to take lower values [62] than we would normally for lower-dimensional data, we assess that it is less likely that a good argument for removing such clusters from the hierarchy can be made, therefore we would like to keep as much of the hierarchy as possible.

On the other hand, the fact that our metric allows a split to take place technically for any size of clusters or dimensionality of data does represent a minor, yet possible, issue. This is especially true for the case where we have a very small child cluster that is extremely loosely connected to, up to completely disconnected from, its parent cluster. While there are some other fixes that can be made, e.g., adjusting the formula itself to account for such scenarios, we have opted for a higher-level approach to address this problem. In general, for our sparse grid-based approach to clustering, obtaining a very small cluster is more often than not an unwanted case, as it can easily be due to some local artifact of the density estimation algorithm. Therefore our fix is simply to add another parameter to the clustering process that sets a minimum cluster size we want to detect with our algorithm, which we argue that doesn't even needs to be modified by the user for most scenarios from the default value.

The visualization. Now that we have presented also the hierarchical variant of the sparse grid-based clustering algorithm, we will delve in more details into the visualizations we can provide with our implementation. As mentioned, the SGDE

Algorithm 3: Cluster splitting algorithm

Input:

cluster – a certain cluster (i.e., node in the clustering tree) for which we check the splitting condition

$G = (V, E)$ – nearest neighbor graph, defined by the set of vertices V and the set of edges E

splitThresh – splitting threshold

Output:

boolean value that tells whether we should split the given cluster

```
Function SplitCluster($cluster$, $G$, $splitThresh$):
    for $childCluster \in cluster.children$ do
        if SplitChild($cluster$, $childCluster$, $G$, $splitThresh$) then
            return True
    return False
Function SplitChild($parentCluster$, $childCluster$, $G$, $splitThresh$):
    $V_p \leftarrow parentCluster.nodes$        /* the set of vertices of parent cluster */
    $V_c \leftarrow childCluster.nodes$        /* the set of vertices of child cluster */
    $conn_p \leftarrow |\{e = \{i, j\} \in E \,|\, i, j \in V_p,\, i \neq j\}|$        /* parent cluster connectivity */
    $conn_p^{max} = \frac{1}{2}|V_p|\,(|V_p| - 1)$        /* maximum parent cluster connectivity */
    $connCoef_p = \frac{conn_p}{conn_p^{max}}$        /* parent cluster connectivity coefficient */
    $conn_{c-p} \leftarrow |\{e = i, j \in E \,|\, i \in V_c,\, j \in V_p\}|$        /* child-parent cluster connectivity */
    $conn_{c-p}^{max} = |V_c|\,(|V_p| - |V_c|)$        /* maximum child-parent cluster connectivity */
    $connCoef_{c-p} = \frac{conn_p}{conn_p^{max}}$        /* child-parent cluster connectivity coefficient */
    return $\frac{connCoef_{c-p}}{connCoef_p} < splitThresh$        /* we split if the connectivity coefficient ratio is below the threshold */
```

part of the clustering algorithm is separate from the subsequent graph-based operations that actually detect the clusters, therefore the existing visualizations, like heatmaps and linear cuts, are kept in their separate function call. The rest of the visualization is called in the new post-processing submodule after all the hierarchy is made available. Similarly to the existing visualization module, the visuals are not actually integrated into the framework, which is written in C++, but instead we generate formatted output files that can then be directly visualized with the free and open-source Plotly library[4]. (For more details, see the initial implementation of the visualization module in the SG++ data mining pipeline in [1].)

We offer the user multiple options of what and how to visualize the data, with flags set in the input JSON file that contains all the parameters for a complete pipeline run. The first kind of outputs we provide are not per se for visualization purposes, but they do help with understanding the clustering results. We can output a file in the well-known CSV format containing all the density evaluations at data points of either the original dataset or its t-SNE embedding (kept in input order), helpful for further post-processing and external visualizations. A second CSV formatted file will contain the corresponding labels of each data point as found by our clustering algorithm. For the user to then be able to understanding the relationships between these labels, the hierarchical information of the clustering tree is stored in a JSON file.

In terms of actual visualization, we generate two JSON files that can be directly used by the Plotly library to generate HTML files that can be visualized in any web-browser. The first file contains the initial nearest neighbor graph of the embedded dataset, with the vertices color-coded based on the value of their densities as computed with the SGDE algorithm. The second file is the most important one, as it contains the whole clustering hierarchy. The visualization is achieved by storing in one animated plot the clusters at each level in the hierarchy tree, with a slider allowing the transition from one level to the next. At each level we show the current clusters as graphs (i.e., the connected components), with color-coded vertices to distinguish between cluster labels. Additionally, we also show the remaining vertices of all the direct parent clusters, with no edges (to avoid confusions), but keeping their color-coding from the previous hierarchy level. For the first level, these points form the noise cluster, considered also as having label -1. Moreover, to aid the user, the legend contains not only the cluster labels, but also the density threshold with which the clusters were found, as, due to the splitting algorithm, clusters found at different density thresholds can end up on the same hierarchical level in the clustering tree. Finally, we used the fact that Plotly is an interactive visualization library to allow the user to get information like coordinates (in embedded space), density values or cluster labels in a pop-up when hovering over a data point in the resulting web-browser plots. Figs. 5.4 and 5.5 show the existing and the newly introduced visualization options, respectively.

Performance aspects. The hierarchical clustering approach presented is currently not implemented to account for the possible parallelization possibilities in order to boost performance. However, we can present some of the options available that can be implemented as a future contribution. The SGDE part has already been covered, including the fact that the density estimation and the nearest neighbor

[4] `https://plotly.com/python/`

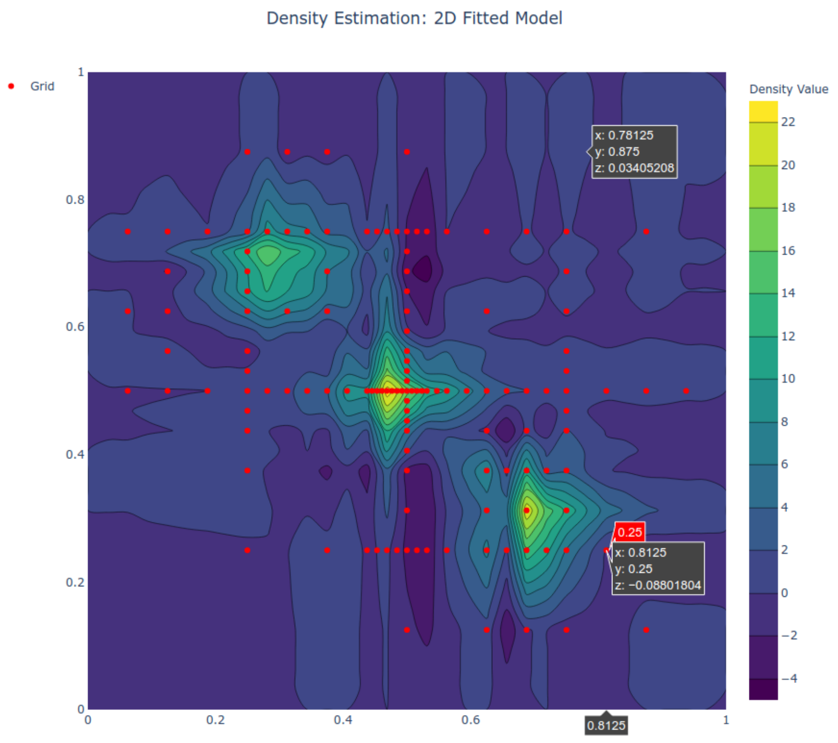


Figure 5.4: Example of the existing data mining pipeline heatmap visualization option, including the Plotly pop-up information available by mouse hover.

graph generation can be done independently in parallel, therefore we focus only on the subsequent steps of the hierarchical clustering algorithm.

In its current form, the loop over the list of density thresholds cannot be directly parallelized, as the modifications on the nearest neighbor graph happen sequentially. One could however separate the graph pruning and connected components processes, which can be done independently per density threshold as they require only concurrent reading of the nearest neighbor graph, from the clustering tree hierarchy building, which includes the process of tracking the parent clusters. This would significantly reduce the amount of total computation required, however some more logic, and possibly some changes to the data structures, would be required.

For a parallel version of the graph operations, one option would be to try a distributed implementation of the VP tree algorithms. Another option, as mentioned also in [62], would be to use instead an alternative that allows easier parallelization, like *locality-sensitive hashing* (LSH). In terms of the splitting algorithm, this can also be parallelized across children clusters, the search ending either when the first child returns a value of 'True' (as then we can just asynchronously interrupt all other searches), or when all child calls return with 'False'.

Regarding the visualization part, one level of parallelism has been accounted for, with the current implementation allowing the linear cuts and the heatmaps to be generated independently.

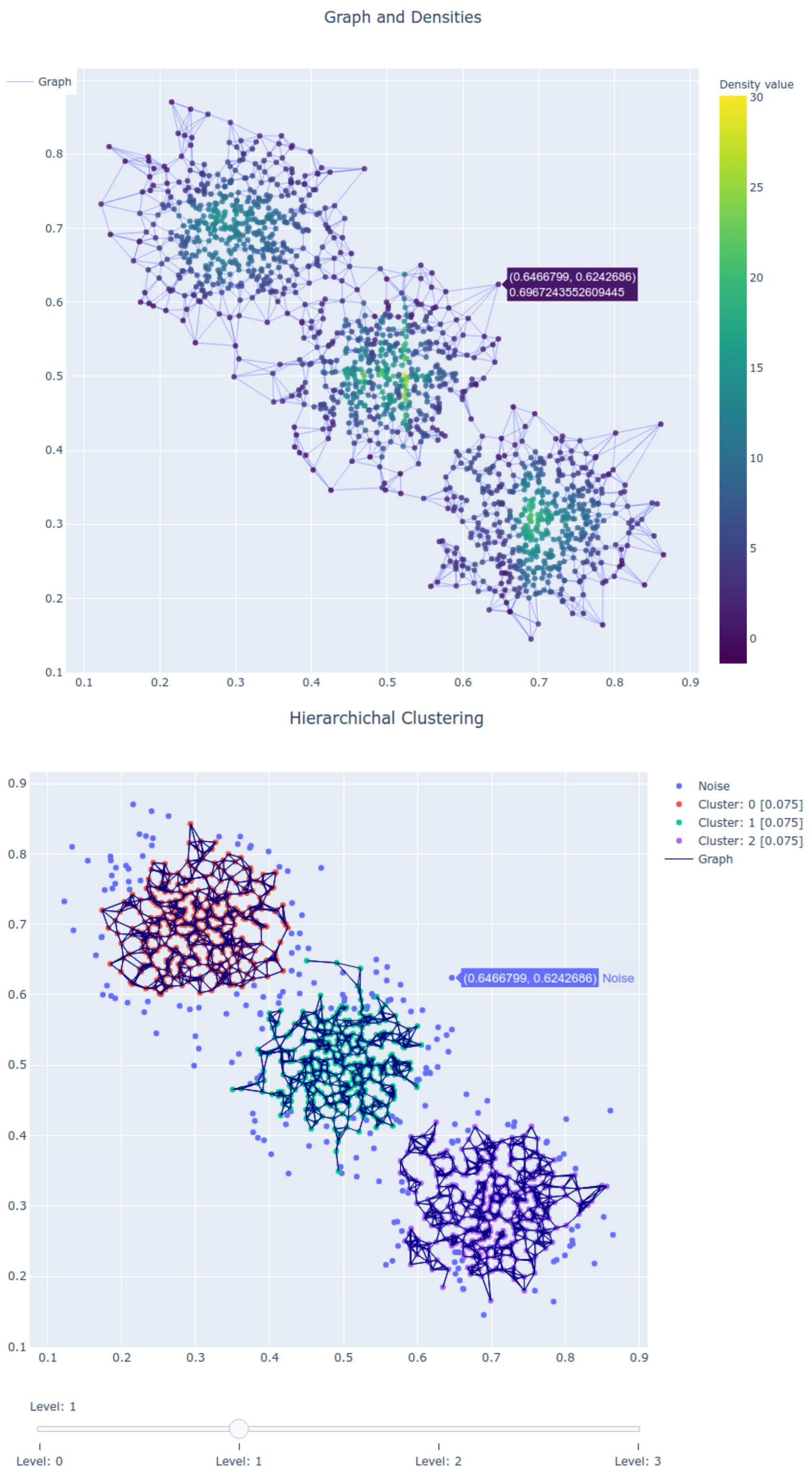


Figure 5.5: Example of the new data mining pipeline clustering visualization options, including the Plotly pop-up information available by mouse hover. Two types of visualization options exist: graph densities (*top*) and clustering hierarchy.

Table 5.1: Description of the hierarchical clustering test datasets.

Dataset name	Dimensionality	Number of samples	Number of real clusters
3Gauss	2	1500	3
2Moons	2	180	2
Iris	4	150	3
HTRU2	8	17898	2

5.2.2 Numerical results

Goals. The purpose of the numerical tests carried out using our newly introduced hierarchical clustering algorithm was to assess the diverse user-oriented functionalities and algorithms introduced with our method and integrated into the data mining pipeline. We have thus not focused in particular on the strict clustering performance aspects (i.e., how well we recover the true clusters from datasets), where existing literature already shows the capabilities of the density-based sparse grid clustering approach [61, 62], but we rather centered our tests on demonstrating the benefits and extent to which our approach, from design to implementation, and the accompanying functionalities added to the data mining pipeline, can be beneficial to users performing clustering tasks.

The datasets. In order to reach the goals of our numerical tests we have opted to work on 4 datasets of different size and dimensionality, which can showcase varied aspects of the hierarchical clustering algorithm. We will next expand on our choice of datasets, with a summary of their main characteristics being presented in Table 5.1.

The *3Gauss* dataset consist of 1500 data points sampled equally from three 2D normal distributions $\mathcal{N}_2(\boldsymbol{\mu}_i, \boldsymbol{\Sigma} = \text{diag}[0.0035, 0.0035])$, $i = \{1, 2, 3\}$ centered at $\boldsymbol{\mu}_1 = [0.3, 0.7]$, $\boldsymbol{\mu}_2 = [0.5, 0.5]$, and $\boldsymbol{\mu}_3 = [0.7, 0.3]$, respectively. These parameters ensure that the dataset has a negligible amount of points closer than 0.1 to any of the 4 domain boundaries, allowing the use of linear basis functions with no ill-effects on the SGDE approximation, while also producing lightly-coupled clusters which cannot be trivially discerned and thus are a challenging enough scenario for any clustering algorithm.

The *2Moons* artificial dataset is part of the datasets provided with the SG++ library and consists of a well-known pattern of two non-linearly-separable clusters of points, distributed in the shape of half circles, in two dimensions. While for this dataset the clustering problem is easily solvable by many algorithms, including the standard sparse grid clustering method, we wanted to explore what other interesting aspects could be revealed by our hierarchical approach.

The *Iris* dataset is a well-known test case in the clustering community, part of the UCI Machine Learning Repository [15]. It consists of 150 real-life measurements

Table 5.2: Sparse grid parameters for the clustering tests.

Dataset name	Grid level	Batch size	λ	Number of adaptivity steps	Refined/ coarsened points	Refinement/ coarsening thresholds	Adaptivity period
3Gauss	3	300	10^{-4}	5	10 / 10	0.0001 / 0.9	300
2Moons	3	60	10^{-3}	3	10 / 10	0.0001 / 0.9	60
Iris	3	50	10^{-4}	2	10 / 10	0.0001 / 0.9	50
HTRU2	3	2000	10^{-5}	5	10 / 10	0.0001 / 0.9	3000

Table 5.3: Hierarchical clustering (*top*) and visualization parameters of the data mining pipeline for our clustering tests.

Nearest neighbors k value	Minimum density threshold	Maximum density threshold	Number of threshold steps	Thresholding distribution	Minimum cluster size
5	0.0	0.75	10	linear	10

Embedding algorithm	Perplexity	Theta parameter	Random seed	Maximum number of iterations
t-SNE	30	0.5	150	1000

of 4 flower attributes (sepal and petal lengths and widths) for three types of Iris flowers, resulting in three equally-sized classes of 4-dimensional data points. While one of these clusters is linearly (and thus trivially) separable from the rest, the other two clusters are not, resulting in a challenging clustering problem overall.

Lastly, we have the *HTRU2* dataset [49], which is a larger and higher-dimensional test case, sourced as well from the UCI Machine Learning Repository [15]. This 8-dimensional dataset consists of samples of astronomical data describing attributes of pulsars (which are a type of radio wave-emitting stars), consisting of two imbalanced classes of 1639 positive examples (true pulsars) and 16259 negative examples (false pulsars).

To allow the use of sparse grids with linear basis functions, the *Iris* and *HTRU2* datasets were linearly transformed such that in each dimension data points lie in the interval $[0.1, 0.9]$. The *3Gauss* dataset did not require such a transformation due to the carefully selected parameters of the sampled Gaussian distributions. However, for all of these three mentioned datasets we have performed an order shuffling to ensure random positions of each cluster data points. The *2Moons* dataset, part of the SG++ library, was already in a sparse grid-suitable state and required no such preprocessing steps.

The test setup and parameters. The main data mining pipeline parameters used in the JSON files when running our tests are shown in Tables 5.2 and 5.3.

Table 5.4: Hierarchical clustering results for the *3Gauss* dataset with split threshold 0.

Level	Number of clusters	Number of points in level	Density thresholds	Average number of points per cluster
0	unclustered	1500	—	—
1	3	1311	0.075	437
2	5	306	0.3, 0.375	61.2
3	2	41	0.45	20.5

In general, we use dataset batching and grid adaptivity using the surplus-value refinement/coarsening strategy for our sparse grids to allow for good SGDE results, which a prerequisite for any good sparse grid-based clustering.

Hierarchical clustering metrics tests. Our main set of tests were meant to showcase the type of results one can expect from our hierarchical clustering algorithm, as well as to analyze the quality of our proposed clustering metrics, as described in Section 5.1. To this end, for each dataset we ran our pipeline to produce two solutions: a *compact hierarchical clustering*, where we considered all the threshold values as described by the parameters in Table 5.2 and applied a split threshold of 0 for creating the cluster hierarchy, and a *flat clustering*, where we performed in effect the standard approach of Peherstorfer [61] by fixing the minimum and maximum density thresholds to a same given value and setting the number of threshold steps to 0.

Two remarks need to be made. Firstly, to reduce the variability of the results and to ease the analysis of the clustering metrics we have opted for this batch of tests to use a split threshold of 0, as mentioned, for creating the cluster hierarchy. Non-zero split thresholds and their effect will be covered subsequently in this section. Secondly, we have to mention that a third type of clustering can be done with our current implementation in the data mining pipeline, namely a *full hierarchical clustering*, where we intentionally retain all clusters at each considered density threshold level, however such a clustering has limited practical value, as it results in many unnecessary clusters and a deep hierarchy (cf. Fig. 5.2). We recommend this option to be left mainly for debug purposes in future extensions of the implementation.

We will begin the showcase of the simulation results with the *3Gauss* dataset. The hierarchical clustering resulted here in a total of 11 clusters, including the noise cluster, distributed in a hierarchy of depth 3, as summarized in Table 5.4. Due to the fact that at level 1 of the hierarchy we obtained already the desired number of clusters, we ran the corresponding flat clustering with the respective density threshold (in this case, 0.075). As mentioned previously, these density values are not absolute, but relative to the maximum SGDE density value as evaluated at the data points. Fig. 5.6 shows all the clusters obtained, as visualized with the new

Table 5.5: Clustering scores for the compact hierarchical (*top*) and flat clusterings, respectively, for the *3Gauss* dataset.

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI))	Davies-Bouldin (DBI)
189	0.690975	0.619456	525.935	6.66517

Homogeneity	Completeness
0.849882	0.487328

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI)	Davies-Bouldin (DBI)
189	0.856915	0.765207	1732.51	5.29374

Homogeneity	Completeness
0.848792	0.696608

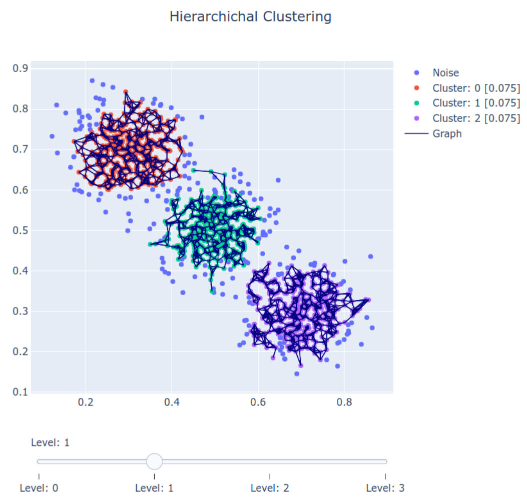
Table 5.6: Hierarchical clustering results for the *2Moons* dataset with split threshold 0.

Level	Number of clusters	Number of points in level	Density thresholds	Average number of points per cluster
0	unclustered	180	—	—
1	2	179	0.0	89.5
2	4	149	0.15, 0.225	74.5
3	2	53	0.3	26.5
4	2	37	0.375	18.5

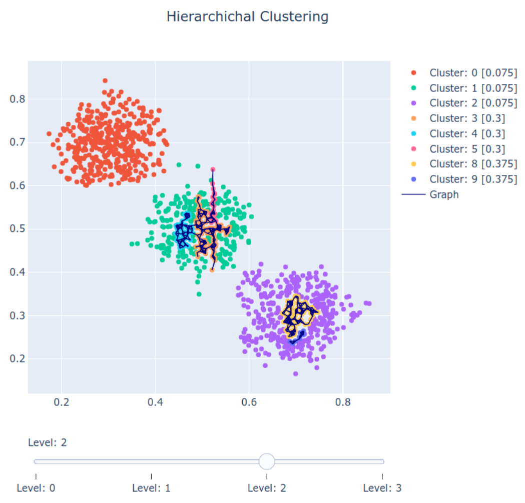
capabilities of the data mining pipeline. We observe that our hierarchical approach can indeed capture the important aspects of the computed density estimation, identifying all density thresholds that produce meaningful sub-clusters.

Table 5.5 summarizes the computed values of the 4 proposed metrics, including the values for homogeneity and completeness used to compute the V-measure. The FMI and VM values, which compare against the true labels, showcase that the flat clustering results are indeed better, as expected, as it does not create additional clusters at higher levels, only those we have obtained at level 1 in the hierarchy, which produces a relatively accurate result. The CHI and DBI values also paint the same picture, with the corresponding CHI value being higher and the DBI value smaller than the corresponding ones obtained for the compact hierarchical clustering.

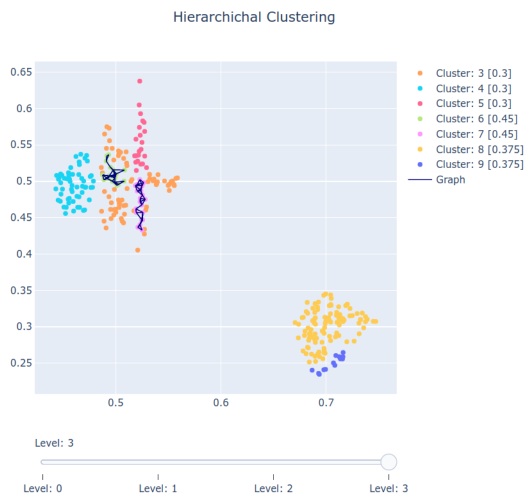
Similar results were obtained for the *2Moons* dataset, where we got also 11 clusters, however across one more level, as presented in Table 5.6. We observe that in this case a density threshold of 0 was enough to identify two distinct classes, which reflects how well-separated the two clusters are in the input dataset and how



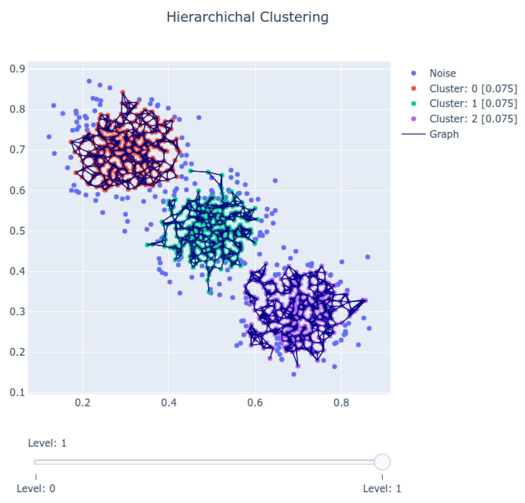
(a) Compact hierarchical clustering - level 1



(b) Compact hierarchical clustering - level 2

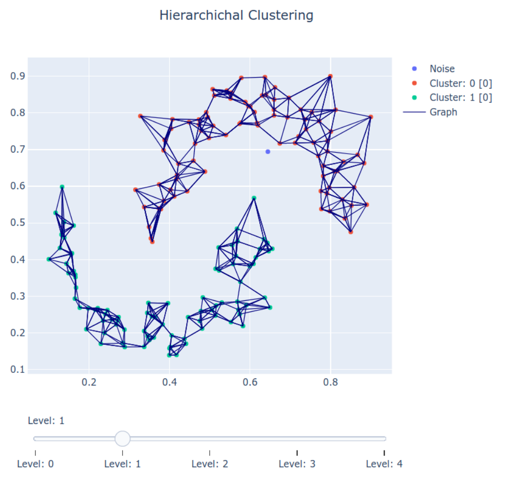


(c) Compact hierarchical clustering - level 3

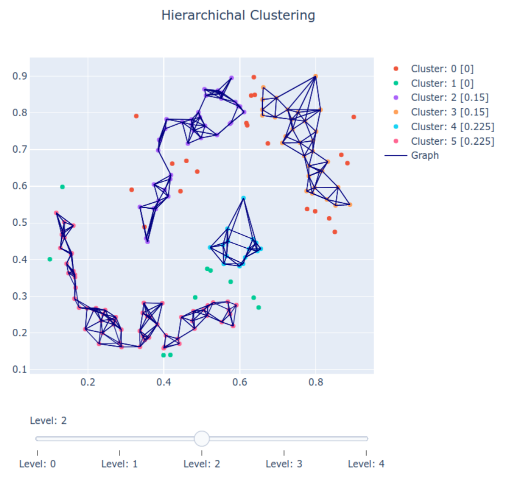


(d) Flat clustering

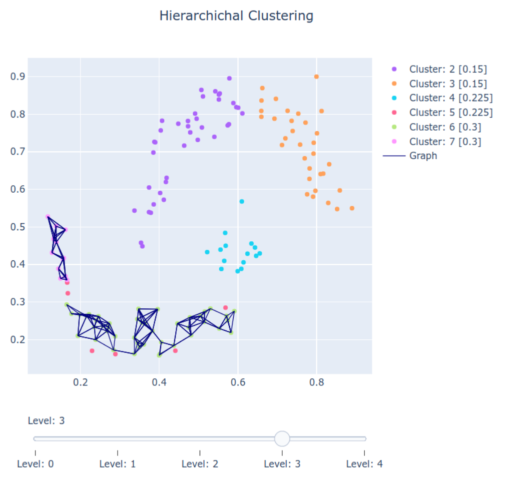
Figure 5.6: Hierarchical and flat clustering results for the *3Gauss* dataset, using our hierarchical clustering pipeline. Level 1 of the hierarchy already produces the same clusters as one would obtain with the standard sparse grid clustering method.



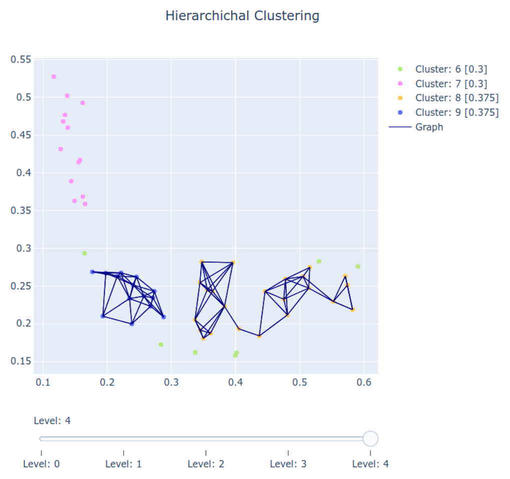
(a) Compact hierarchical clustering - level 1



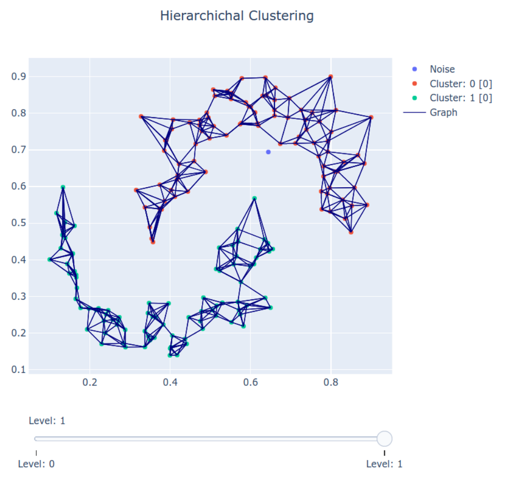
(b) Compact hierarchical clustering - level 2



(c) Compact hierarchical clustering - level 3



(d) Compact hierarchical clustering - level 4



(e) Flat clustering

Figure 5.7: Hierarchical and flat clustering results for the *2Moons* dataset, using our hierarchical clustering pipeline. Level 1 of the hierarchy already produces the same clusters as one would obtain with the standard sparse grid clustering method.

Table 5.7: Clustering scores for the compact hierarchical (*top*) and flat clusterings, respectively, for the *2Moons* dataset.

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI)	Davies-Bouldin (DBI)
1	0.503915	0.484046	69.1593	0.843039

Homogeneity	Completeness
1	0.319301

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI)	Davies-Bouldin (DBI)
1	0.994306	0.978364	113.01	3.65059

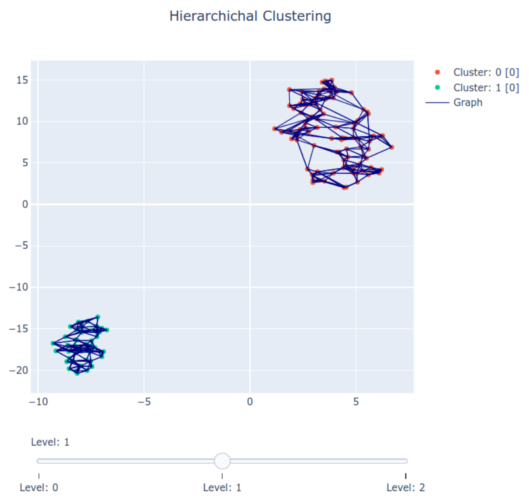
Homogeneity	Completeness
1	0.957644

Table 5.8: Hierarchical clustering results for the *Iris* dataset with split threshold 0.

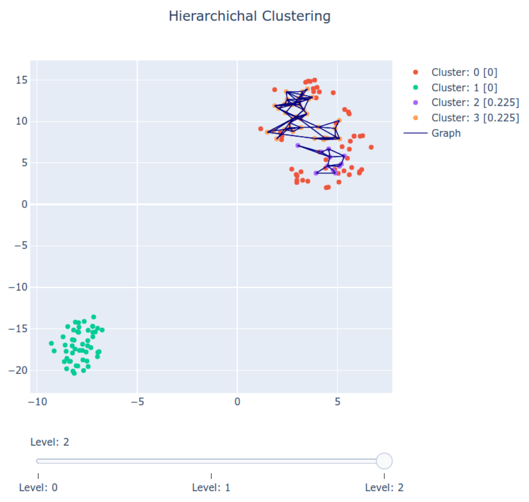
Level	Number of clusters	Number of points in level	Density thresholds	Average number of points per cluster
0	unclustered	150	—	—
1	2	150	0.0	75
2	2	50	0.3, 0.225	25

easy it was for the sparse grid approach to identify this separation, even with a low level grid. The corresponding flat clustering was obtained using this density threshold of 0, resulting in the same clusters as those at level 1 of the hierarchy (Fig. 5.7). The FMI and VM values in Table 5.7 confirm that this is an almost perfect clustering result, with just one data point incorrectly assigned as noise. It is to be expected that a small increase in grid points via a higher initial level or additional refinement steps would produce a grid able to fix this, obtaining thus a perfect clustering, however that is not our focus in these tests. What has to be noted though here is that, although the CHI value is in line with expectation, the DBI produces the opposite results. This can definitely happen when considering internal metrics, like DBI, therefore using multiple such metrics gives a better chance of correctly distinguishing a good clustering from a poor one and providing the user relevant insight.

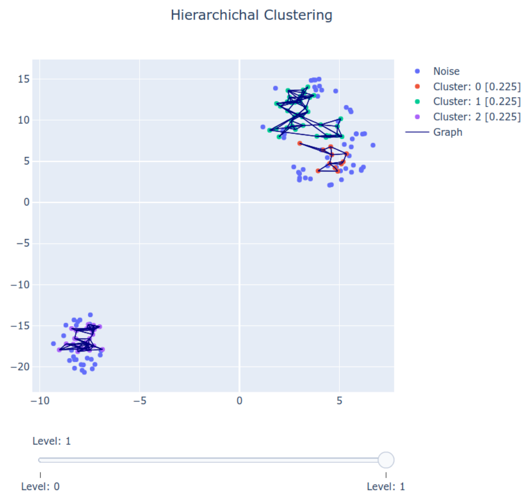
For the *Iris* dataset Table 5.8 shows that we obtain with our approach two levels of 2 clusters each, with no points marked as noise. Fig. 5.8 allows the visual inspection of the resulting clusters, together with the those obtained by the corresponding flat clustering using a threshold value of 0.225, with all data points embedded in 2D space via the t-SNE algorithm of the data mining pipeline. We can observe clearly



(a) Compact hierarchical clustering - level 1



(b) Compact hierarchical clustering - level 2



(c) Flat clustering

Figure 5.8: Hierarchical and flat clustering results for the *Iris* dataset, using our hierarchical clustering pipeline. Here we can see that the hierarchical clustering can produce if not the same then better clusters than the standard sparse grid clustering method, with the linearly separable class being correctly and fully identified (Cluster 1 in **(a) and (b)**).

Table 5.9: Clustering scores for the compact hierarchical (*top*) and flat clusterings, respectively, for the *Iris* dataset.

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI)	Davies-Bouldin (DBI)
0	0.722094	0.636946	141.317	1.68952

Homogeneity	Completeness
0.68786	0.593049

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI)	Davies-Bouldin (DBI)
75	0.495529	0.382766	24.3257	0.989091

Homogeneity	Completeness
0.39817	0.368509

Table 5.10: Hierarchical clustering results for the *HTRU2* dataset with split threshold 0.

Level	Number of clusters	Number of points in level	Density thresholds	Average number of points per cluster
0	unclustered	17898	—	—

that in this scenario our hierarchical approach produces a net better labeling of the data points into meaningful clusters by the fact that the linearly separable class is correctly and fully identified as such, while the flat clustering fails to do so due to its intrinsic simplicity in approach. This visual assessment is confirmed also by the computed metrics presented in Table 5.9. One has to note however that here, similarly to the *2Moons* scenario, the DBI metric fails again to correctly assess the better clustering result as such.

The *HTRU2* dataset provided a different type of result. Here our hierarchical clustering did not return any clusters (Table 5.10), which means that at no density threshold more than one connected component could be identified from the sparse grid-based estimated density. Thus, for the corresponding flat clustering we had no practical information and therefore we chose the first density threshold capable of creating a separation of clusters, i.e., 0.075. This results in a decent identification of the real clusters, based on our metrics in Table 5.11. For completeness, Fig. 5.9 shows the 2D t-SNE embedding of these identified clusters, and Table 5.12 shows the resulting confusion matrix, proving that, even though we did not tailor in any way our method's parameters to this more challenging dataset, the sparse grid clustering is still capable of identifying a large portion of the minority class (i.e., the true pulsars).

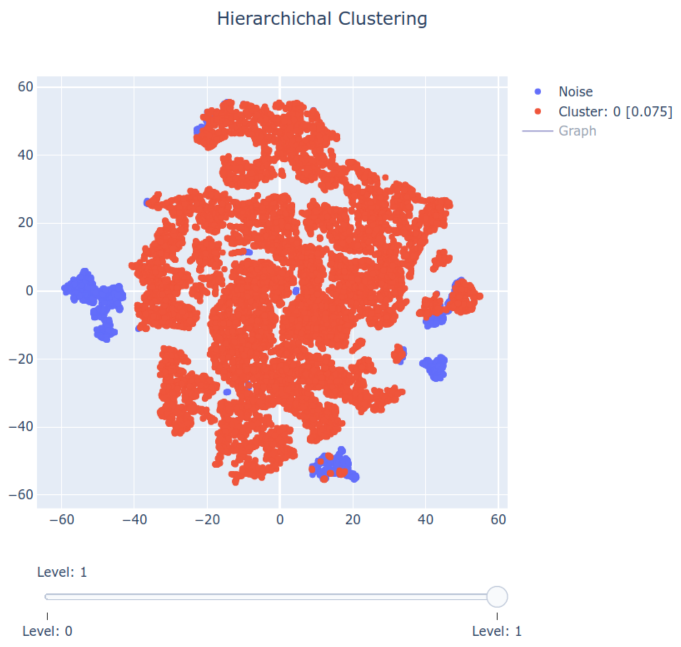


Figure 5.9: Flat clustering results for the *HTRU2* dataset, using our hierarchical clustering pipeline. The minority class (true pulsars) data points are identified as noise.

Table 5.11: Clustering scores for the flat clusterings for the *HTRU2* dataset. The hierarchical clustering did not return any clusters, therefore the value of the metrics int hat case have no practical meaning.

Number of noise points	Fowlkes-Mallows (FMI)	V-measure (VM)	Caliński-Harabasz (CHI)	Davies-Bouldin (DBI)
1374	0.937327	0.384065	9706.77	0.889982

Homogeneity	Completeness
0.361886	0.40914

Table 5.12: Confusion matrix for the flat clustering of the *HTRU2* dataset. For the true pulsars (positive examples) we obtain a precision of 0.73 and a recall value of 0.61.

Labels \ Clusters	Noise cluster	Cluster 0
negative examples (false pulsars)	371	15888
positive examples (true pulsars)	1003	636

Split threshold tests. So far we have only considered the case when we use a split threshold of 0, which results in the most complex compact hierarchy by retaining all cluster parents with more than one children. In the following we will show the consistency of the hierarchical cluster splitting metric, as well as its practical applicability.

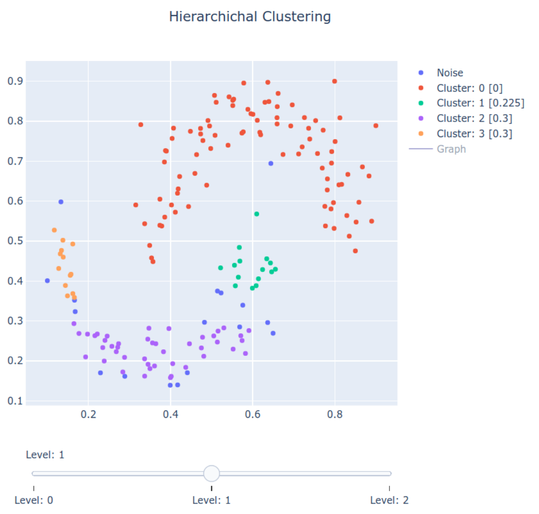
Firstly, Table 5.13 presents the split scores computed for each set of multiple cluster children across all hierarchy levels for the 4 considered datasets, as resulting from the compact hierarchical clusterings presented previously. As can be observed, across all datasets where a cluster hierarchy could be established the magnitudes of the split scores vary in relatively same manners, showing that our strategy for computing this metric is stable and consistent across different parameters of the datasets and clusters (i.e., number of data points, size of clusters, dimensionality of data points).

To assess the other point of interest, i.e., the usefulness of our approach, we considered the particular case of the *2Moons* scenario and ran additional simulations with split thresholds 1.5 and 2.0. These values were chosen based on the tabulated split metric values in Table 5.13, in order to allow various merges of clusters at different hierarchical levels. In Table 5.14 we summarize the obtained cluster hierarchies. An increase in the split threshold results in a flatter hierarchy by condensing clusters at previous diverse levels, which are to a higher or lesser degree connected in a graph sense to one another, as can be seen in Fig. 5.10.

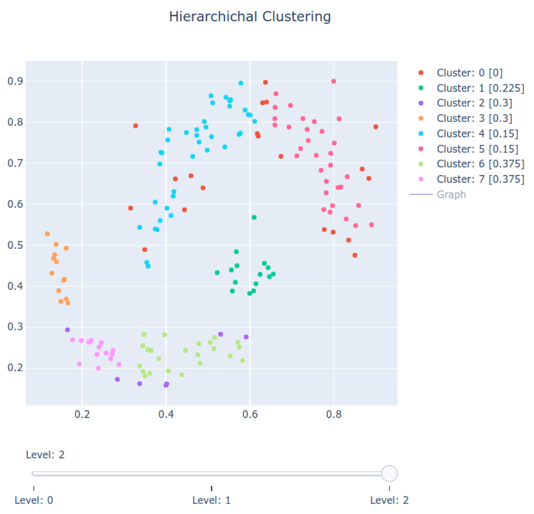
Summary of results. Through our tests we have demonstrated the new functionality of the data mining pipeline through our hierarchical clustering algorithm, which extends the capabilities of the standard approach, giving the user the possibility to gain new insights into the characteristics of the input datasets by means of additional sub-clusters (*3Gauss*, *2Moons*), better quality clusters (*Iris*), or even by means of negative clustering results (*HTRU2*). Besides improving the SGDE approximation, our clustering approach now has parameters that can improve the result of the clustering process which relate to the nearest neighbor graph portion of the algorithm, like our split threshold, providing thus a more powerful clustering library overall.

Table 5.13: Computed split metric values obtained for the considered datasets. The values are grouped by the level at which they appear, each value representing one of multiple child clusters.

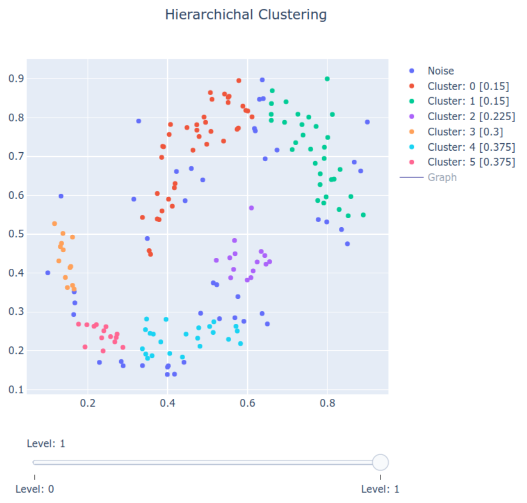
Dataset	Split metric values
3Gauss	(2.40794, 2.17093, 1.37411), (3.34131, 2.48213), (2.1738, 1.6241)
2Moons	(1.81808, 1.67035), (1.23619, 3.58565), (3.31704, 1.21181), (2.07478, 1.64157)
Iris	(1.51934, 2.48045)
HTRU2	—



(a) Compact hierarchical clustering - level 1; split threshold 1.5



(b) Compact hierarchical clustering - level 2; split threshold 1.5



(c) Compact hierarchical clustering - level 1; split threshold 2.0

Figure 5.10: Hierarchical clustering results for the *2Moons* dataset, using our hierarchical clustering pipeline, with split thresholds 1.5 (**a** and **b**) and 2.0 (**c**), respectively.

Table 5.14: Hierarchical clustering results for the *2Moons* dataset for split thresholds 1.5 and 2.0, respectively.

Level	Number of clusters	Number of points in level	Density thresholds	Average number of points per cluster
0	unclustered	180	—	—
1	4	163	0, 0.225, 0.3	40.75
2	4	108	0.15, 0.375	27

Level	Number of clusters	Number of points in level	Density thresholds	Average number of points per cluster
0	unclustered	1500	—	—
1	6	136	0.15, 0.225, 0.3, 0.375	22.67

5.3 Uncertain Clustering

A second extension to the clustering task that was addressed in our work was in the handling of uncertain datasets. This encompasses the situations where the actual coordinates of our input data points are not known exactly. The simplest option to model such a dataset is to consider that each data point is described by its expectation (as a d-dimensional vector) and a variance under the assumption that the uncertainty is of a known, simple distribution (e.g., Gaussian). A set of samples from these distributions, one for each uncertain data point, forms a so-called *possible world* or instance of the uncertain dataset. The same way as the results of clustering a deterministic dataset can be measured by some metrics, clustering an uncertain dataset adds an additional complexity layer, usually in the form of some confidence measure, as what makes a 'good' clustering now depends also on the parameters of the uncertainty model considered.

Keeping in the spirit of our high usability goal, we took a method-independent approach for handling uncertainty in the context of clustering and applied it to our sparse grid-based algorithm. Therefore, while the approach to model the uncertainties and extract meaningful information from the resulting clusters is not new, our usage of the sparse grid clustering algorithm in this context is, to our knowledge, the first study of any kind of sparse grid-based uncertain clustering.

In the following, we will describe first of all the theoretical aspects of the uncertainty model. Next we will delve into certain details regarding our implementation and how the integration with our regular sparse grid-based clustering algorithm takes place. Lastly, we will present some numerical results.

5.3.1 Theory and implementation

The approach. The idea of Züfle et al. [92] of *representative clustering*, which we employ, is to find for a set of possible worlds sampled from our unknown dataset a subset for which applying the deterministic clustering algorithm produces results that are representative (i.e., close under some measure), with some degree of certainty, to what it would produce on the actual, unknown deterministic dataset. This method gives mathematical guarantees regarding the degree of confidence in the output and requires far less clusterings to be done on possible worlds than would, for example, a naive Monte Carlo approach.

The steps of the uncertain clustering using representatives in the context of our sparse grid-based implementation follows for the most part the ones described in [92]:

Step 1: Construct the set $X = \{X_0, \ldots, X_{|X|-1}\}$ of possible worlds by sampling from the uncertain input dataset.

Step 2: For each such world $X_i \in X, i = 0, \ldots, |X| - 1$, apply the sparse grid clustering (which we will denote in this context by $\mathcal{C}$), resulting in labelings (or *clusterings*) $\mathcal{C}(X_i)$. This results in a set of distinct clusterings, called the *possible clusterings set PC*. For each entry, the number of apparitions in the set $\{\mathcal{C}(X_i)\}$ gives its *support*.

Step 3: Using a distance measure $d_{\mathcal{C}}$ that describes how similar two clusterings are to each other, cluster by any given distance-based method $\mathcal{C}'$ the elements of PC.

Step 4: For each such *metacluster* returned by $\mathcal{C}'$, select, in a specific manner, one of its member elements to be its *representative*. The set of these metacluster representatives is the output of the uncertain clustering.

Sampling possible worlds. The way we construct the set X in step 1 of our algorithm as already mentioned by taking a sample from each of the distributions describing each data point of our uncertain input dataset. To reduce the computational demand, we can reuse samples in different combinations as the field of generated possible worlds from which to choose the number we require. For example, one can generate just $j = 10$ samples per data point for an uncertain dataset of size N and then randomly choose as possible worlds $|X| = 100$ of the resulting $j^N = 10^{100}$ combinations. (In practice, of course, one does not even effectively generate j^N items, with $j * N$ being enough for a simple, yet efficient, implementation.) This approach reduces both the space and time requirements for the sampling procedure.

While there is no upper bound to the number of possible worlds one would use, previous results showed that moderate values are enough in order to obtain meaningful results. The only restriction on $|X|$ is a lower bound given by the central limit theorem (used in the calculation of the confidence degree we can obtain in the representative clustering approach), leading to a generic rule of thumb of $|X| > 30$.

A more detailed explanation of the implementation of such a sampling can be found in the source material [92].

Integration with the pipeline. The second step of the algorithm is where the sparse grid clustering comes into play, as we need to cluster each possible world in X. This step is also the most computationally demanding, however it can be straight-forward parallelized, as the clusterings are independent of one another. Integrating the representative clustering approach into the data mining pipeline was however considered to be unnecessary because, on one hand, it would require extensive changes to the whole pipeline to allow multiple clusterings to be done in one overarching run (although it can be considered in future contributions) and, on the other hand, the other steps of the uncertain clustering algorithm can more easily be done independently of the sparse grid-related methods. As such, our implementation encapsulates the calls to the C++-based pipeline clustering for each world X_i into a larger Python-based code, with scripts written to automatize almost the whole uncertain clustering algorithm, maintaining the overall high degree of usability of our implementation. Thus, for example, we can automatically create the proper JSON input file associated to each dataset X_i needed for the sparse grid clustering call and, afterwards, we can automatically retrieve the labelings stored in the output files of the data mining pipeline needed for the subsequent steps 3 and 4 of the representative clustering algorithm.

In order for the uncertain clustering to return meaningful results, an important aspect is the setup of the parameters for $\mathcal{C}$. Therefore, the template JSON file used to run all clusterings for each uncertain dataset would ideally have to contain parameters that would make sense for a similar deterministic input. While one can think of more complex options, considering the type of uncertainty model we work with, we have opted to choose the parameters in the template JSON file such that they would return good results if used by the deterministic sparse grid clustering using as input the set of expectation vectors for each uncertain data point.

The metaclustering. For the third step of the algorithm we need to cluster the $|X|$ clusterings in order to distinguish subsets of labelings that are closely related, removing thus part of the variance introduced by the uncertainties in the original input dataset. For our implementation we have kept in line with the approach of the original paper and also used $\mathcal{C}' = \text{PAM}$ (*partition around medoids*), available as a Python package [73], and using the reciprocal ARI (*adjusted Rand index*) as a metric between clusterings (i.e., $d_{\mathcal{C}} = 1 - \text{ARI}$) due to its invariance to label value shuffling.

Obtaining the representatives. Our implementation returns three results. The first one is the *sample medoid*, which is simply the median of all clusterings in PC, weighted by their support. This is computed purely for comparison reasons, as it represents a naive solution to the uncertain clustering problem.

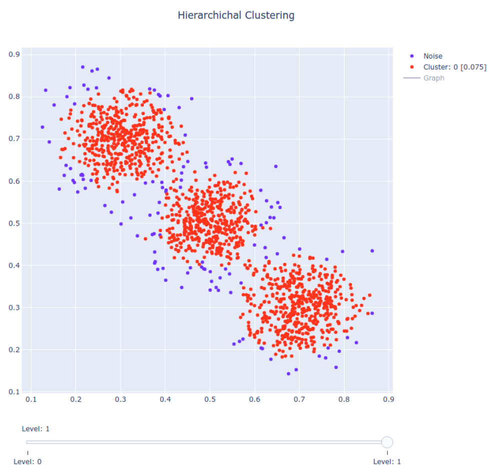
The second and third results are given each as a list of *τ-ϕ-representatives* [92], one per meta-cluster. Such a representative R, described by a computed value $\phi = \hat{P}(R, \tau, \alpha)$, is guaranteed, under the assumptions of the central limit theorem with respect to the number of possible worlds sampled, that it is, with a level of confidence α provided by the user and with probability ϕ, at most at a distance τ from the other clusterings in its respective meta-cluster. We have implemented both options to return such representatives described in the work of Züfle et al.: the *complete representative* approach, where both the values of τ and ϕ are computed from the meta-clusters, and the *τ_{max}-clustering* variant, where we restrict the maximum value of τ and only the corresponding value of the probability ϕ is to be correspondingly computed from the data.

5.3.2 Numerical results

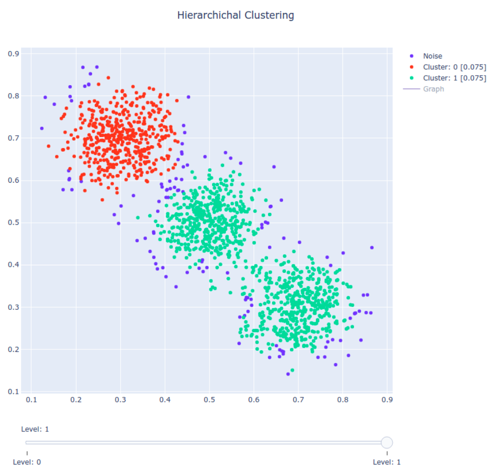
Goals. As the method itself was already proven to produce good quality clusters in the original work of Züfle et al. [92], our tests will instead simply focus on validating the use of world representatives for sparse grid uncertain clustering. We will use some of the same datasets previously used in the context of hierarchical clustering and assess both visually and numerically the obtained representatives.

The test scenarios. As ground-truth clusterings for which we generated representatives we used the flat clustering results for the datasets *3Gauss*, *2Moons*, and *HTRU2*, as described in Section 5.2.2. We set the uncertain clustering parameters from [92], i.e., we generated $|X| = 100$ representatives from $j = 10$ samples per data point, sampled independently from standard Gaussian distributions with variance $\sigma = 0.01$ with the strategy previously described. Unless otherwise specified, we set the algorithm to return 4 representatives for each scenario, with a limit $\tau_{max} = 0.1$ for the τ_{max}-clustering approach.

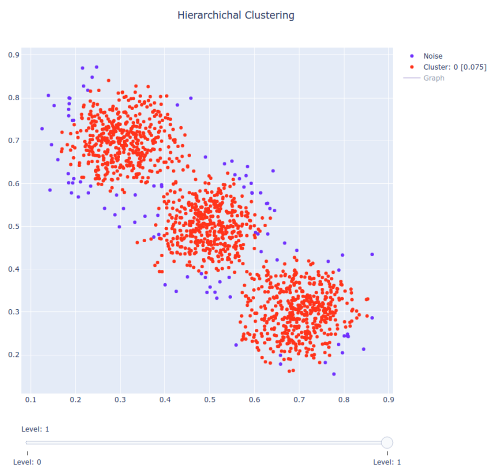
Representative clustering results. The first dataset investigated was *3Gauss*, for which we tried to get 4 useful representatives. In Fig. 5.11 we present the median representative and the complete representatives returned by the uncertain clustering approach. The median failed to capture the 3 clusters in the dataset, while the four complete representatives, similarly to what was obtained in [92], perfectly showcased the 4 possible cluster combinations of the three-mode dataset. The difference in our case was that the closest representative to the ground-truth was not returned with the largest probability, however it had the smallest associated τ distance. The results improved considerably when we imposed a maximum τ limit, as shown in Fig. 5.12: the closest representative to the ground-truth chosen from the four selected by the algorithm was returned with the largest probability.



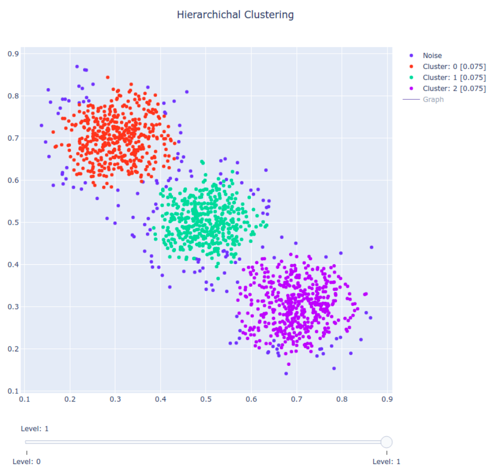
(a) Median representative X_1; $d_{\mathcal{C}} = 0.938$



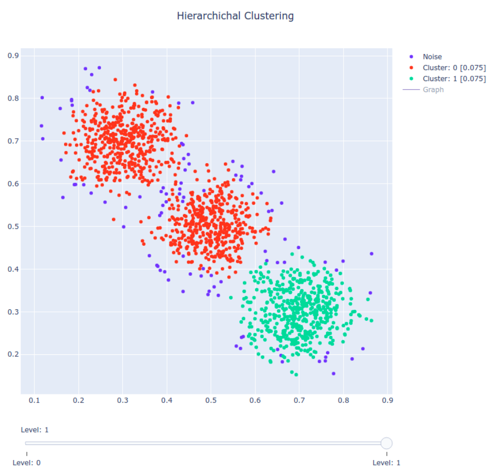
(b) Complete representative $\hat{P}(X_{20}, 0.154, 0.95) = 0.105$; $d_{\mathcal{C}} = 0.512$



(c) Complete representative $\hat{P}(X_{30}, 0.485, 0.95) = 0.402$; $d_{\mathcal{C}} = 0.942$

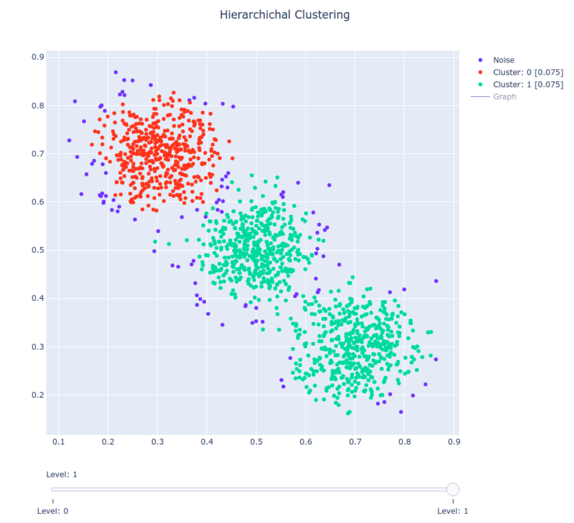


(d) Complete representative $\hat{P}(X_{82}, 0.130, 0.95) = 0.088$; $d_{\mathcal{C}} = 0.140$

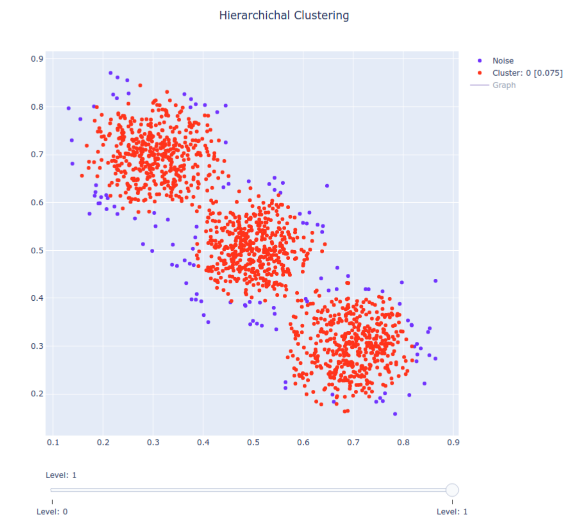


(e) Complete representative $\hat{P}(X_{63}, 0.150, 0.95) = 0.088$; $d_{\mathcal{C}} = 0.509$

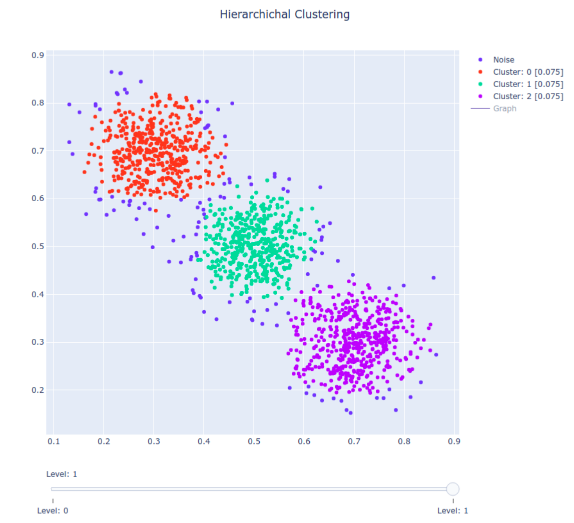
Figure 5.11: Median and 4 complete representatives for the *3Gauss* dataset. The returned median clustering cannot hope to reflect nicely the three modes of the dataset, but the complete representatives capture the 4 possible cluster combinations one would expect.



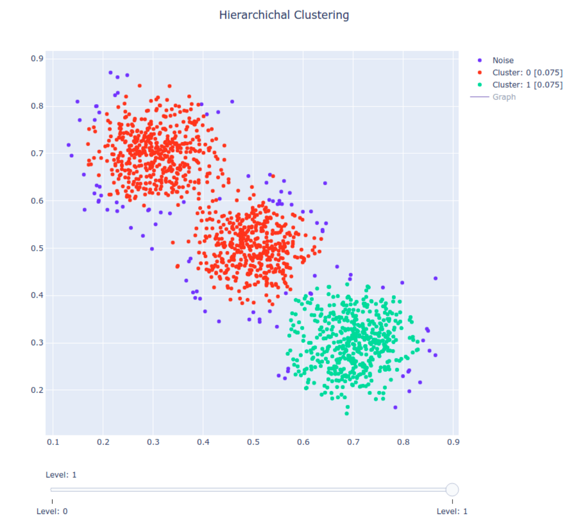
(a) τ_{max} representative ($\tau_{max} = 0.1$) $\hat{P}(X_{23}, 0.1, 0.95) = 0.002$; $d_{\mathcal{C}} = 0.520$



(b) τ_{max} representative ($\tau_{max} = 0.1$) $\hat{P}(X_0, 0.1, 0.95) = 0$; $d_{\mathcal{C}} = 0.945$



(c) τ_{max} representative ($\tau_{max} = 0.1$) $\hat{P}(X_{25}, 0.1, 0.95) = 0.034$; $d_{\mathcal{C}} = 0.159$



(d) τ_{max} representative ($\tau_{max} = 0.1$) $\hat{P}(X_{21}, 0.1, 0.95) = 0.002$; $d_{\mathcal{C}} = 0.501$

Figure 5.12: τ_{max} representatives for the *3Gauss* dataset. Not only do we still keep the 4 possible cluster combinations as for the complete representatives, but now the probabilities have a more clear and useful interpretation.

Our second dataset investigated is the 8-dimensional *HTRU2*. In this scenario the median representative by chance returned a very good clustering, as revealed by the small distance to the ground-truth. From the 4 complete representatives, shown in Fig. 5.13 against the median result, none were as close to the ground-truth clustering, although two of them were at statistically significant small distances to the target clustering. Again, using the strategy of limiting the maximum τ value, shown in Fig. 5.14, we retrieved with the second highest probability a representative that lies at the same distance to the ground-truth as the median result, therefore equaling the uncommon to be obtained best possible solution from the naive approach.

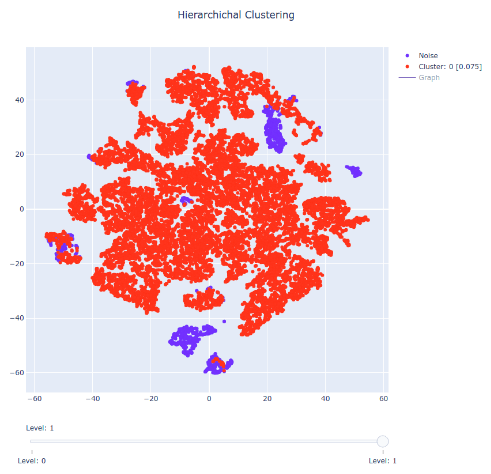
For our last test dataset, *2Moons*, we began by searching for two representatives, considering the simplicity of the scenario. These results are shown in Fig. 5.15. While the naive median selection failed by returning a clustering with three classes, the complete representative did identify a good clustering candidate, however, with both representatives returned with corresponding τ distances close or equal to 1.0, this was a statistically insignificant result. Results were improved by limiting τ to at most 0.1, which returned a very good clustering candidate with an attached probability of almost 40%.

Increasing the number of returned representatives to 4 improved slightly the quality of these complete representatives, with some of the τ values now lower, although still practically not useful. Once more, a τ_{max}-clustering was needed to return statistically significant results, with the selected representative with the highest attached probability having also the smallest distance to the ground-truth clustering. These results are shown in Figs. 5.16 and 5.17, respectively.

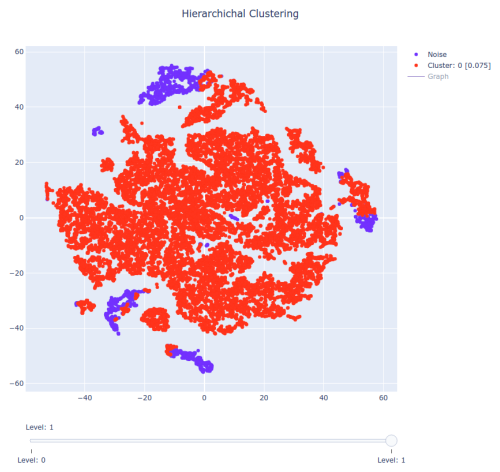
5.4 Summary

Our contributions in this chapter focused on increasing the usability of the standard clustering approach using sparse grids. In a first direction, we have integrated the clustering algorithm of Peherstorfer [61] into the SG++ data mining pipeline of [69], then extended it to a hierarchical clustering approach that provides a more powerful tool in solving the clustering task. Our numerical tests on datasets of various sizes and dimensionality showed the power of such a method, with the user being aided in making decisions by new metrics and new visualization options. Through a combination of the adaptivity in the SGDE approach, the choices of range and distributions of the density threshold, and the possibility of adjusting the cluster hierarchies through our split threshold, a user with expert knowledge has now many tools at its disposals to analyze a given input dataset and produce a high quality result to the otherwise complex task of clustering.

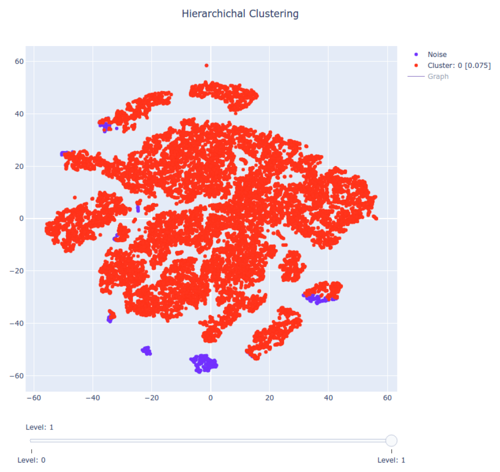
The introduction of the first (to the best of our knowledge) sparse grid uncertain clustering method, by marrying our standard clustering algorithm with an existing approach based on world representatives, provided a different contribution in our usability research direction. Our numerical tests showed that this approach allows



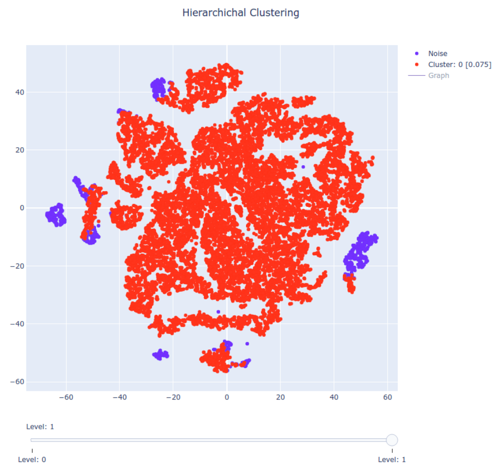
(a) Median representative X_{51}; $d_{\mathcal{C}} = 0.071$



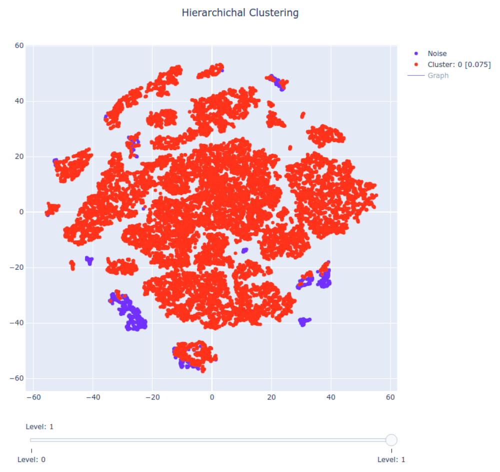
(b) Complete representative $\hat{P}(X_{78}, 0.117, 0.95) = 0.314$; $d_{\mathcal{C}} = 0.125$



(c) Complete representative $\hat{P}(X_{70}, 0.105, 0.95) = 0.013$; $d_{\mathcal{C}} = 0.502$

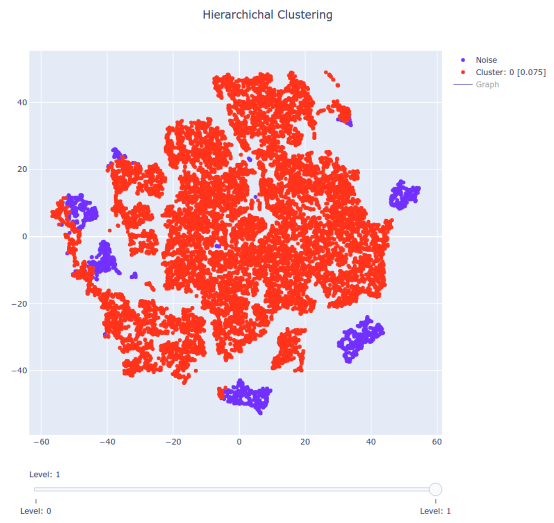


(d) Complete representative $\hat{P}(X_{23}, 0.172, 0.95) = 0.382$; $d_{\mathcal{C}} = 0.095$

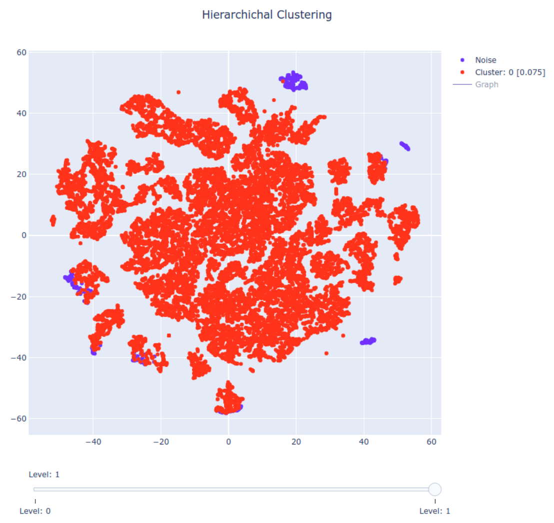


(e) Complete representative $\hat{P}(X_{65}, 0.107, 0.95) = 0.08$; $d_{\mathcal{C}} = 0.233$

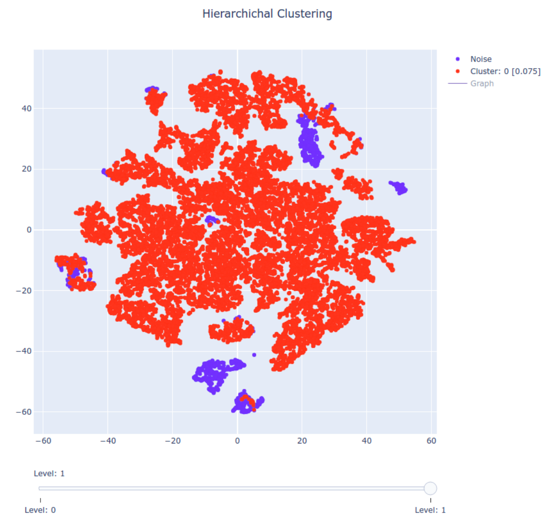
Figure 5.13: Median and 4 complete representatives for the *HTRU2* dataset. By chance, here the median representative is very close to the ground-truth, being better than the complete representatives, which still come relatively close to the target clustering.



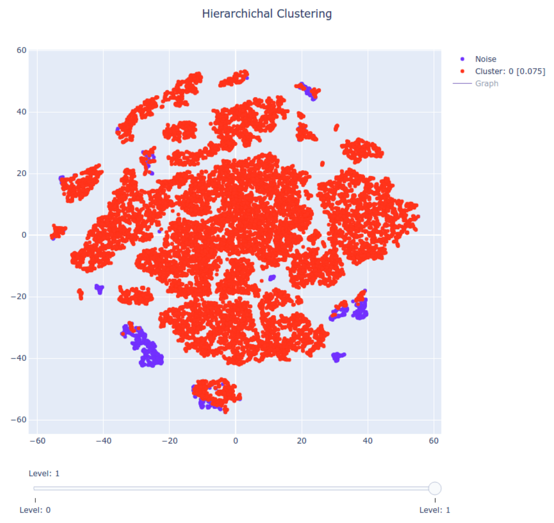
(a) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_5, 0.1, 0.95) = 0.256$; $d_{\mathcal{C}} = 0.184$



(b) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_3, 0.1, 0.95) = 0.007$; $d_{\mathcal{C}} = 0.518$

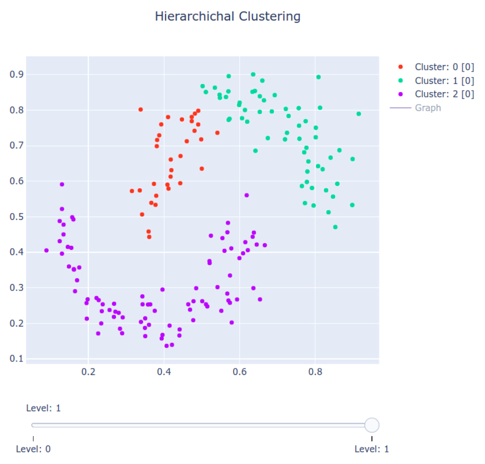


(c) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_{51}, 0.1, 0.95) = 0.139$; $d_{\mathcal{C}} = 0.071$

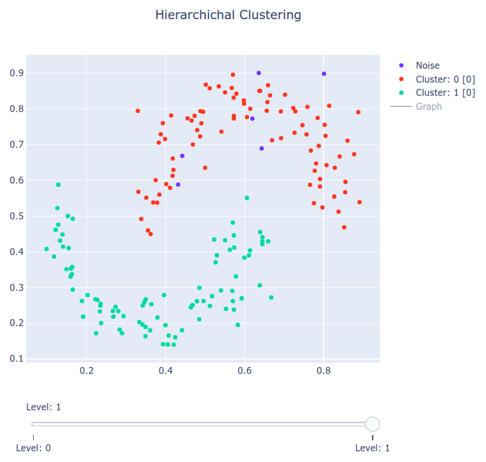


(d) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_{65}, 0.1, 0.95) = 0.072$; $d_{\mathcal{C}} = 0.233$

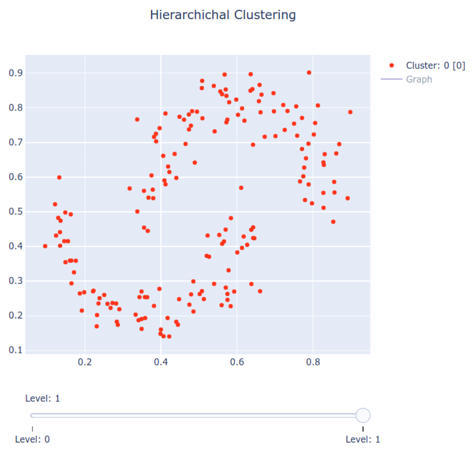
Figure 5.14: τ_{max} representatives for the *HTRU2* dataset. From the 4 clusterings one of them matches the surprisingly good median result, improving thus on the obtained complete representatives.



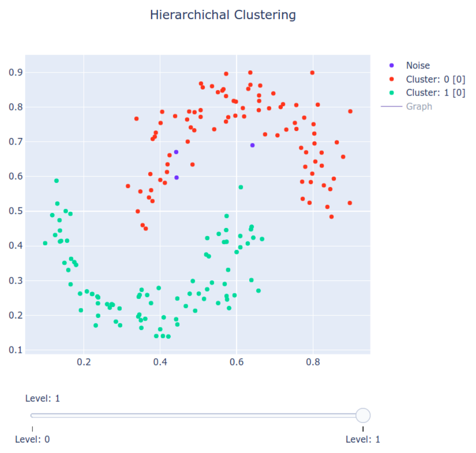
(a) Median representative X_{30}; $d_{\mathcal{C}} = 0.246$



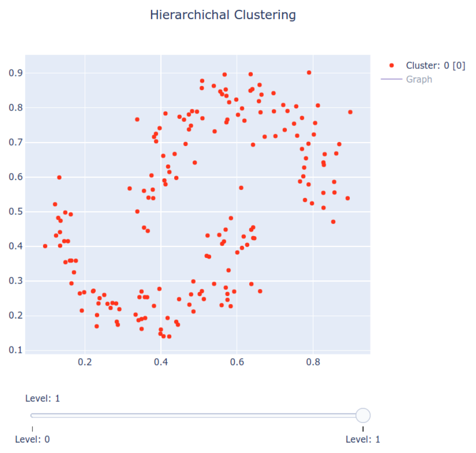
(b) Complete representative $\hat{P}(X_{65}, 0.986, 0.95) = 0.733$; $d_{\mathcal{C}} = 0.054$



(c) Complete representative $\hat{P}(X_1, 1, 0.95) = 1$; $d_{\mathcal{C}} = 1$

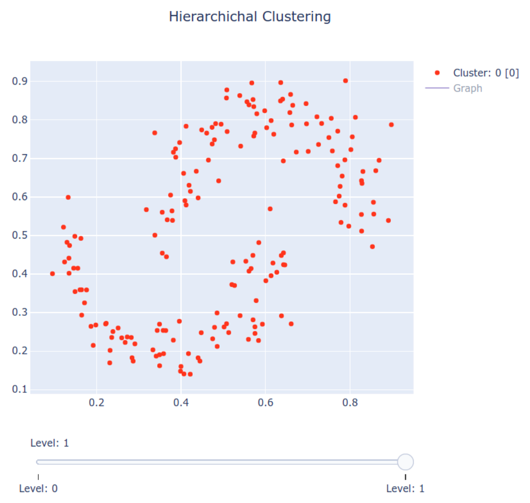


(d) τ_{max} representative ($\tau_{max} = 0.1$) $\hat{P}(X_4, 0.1, 0.95) = 0.382$; $d_{\mathcal{C}} = 0.022$

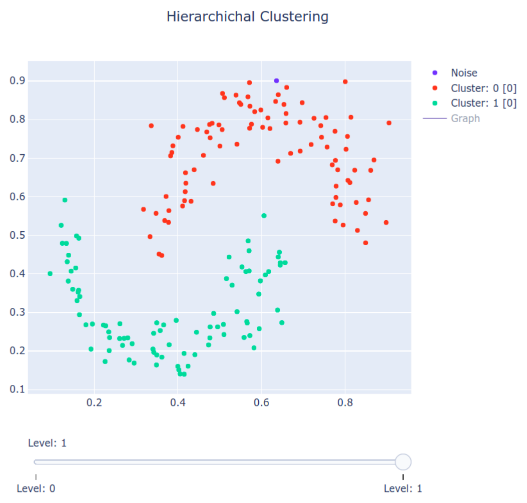


(e) τ_{max} representative ($\tau_{max} = 0.1$) $\hat{P}(X_1, 0.1, 0.95) = 0.007$; $d_{\mathcal{C}} = 1$

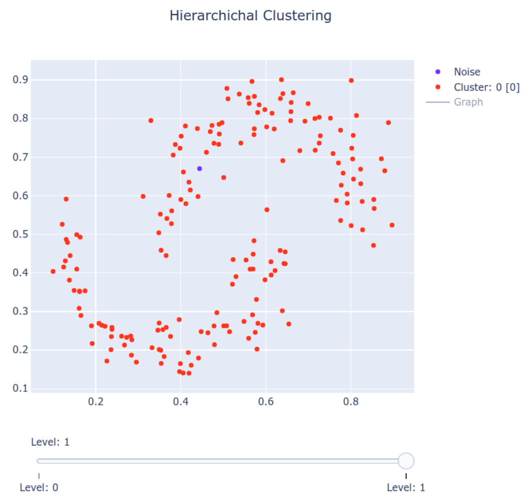
Figure 5.15: Uncertain clustering results for the *2Moons* dataset. Here, we looked for two representatives.



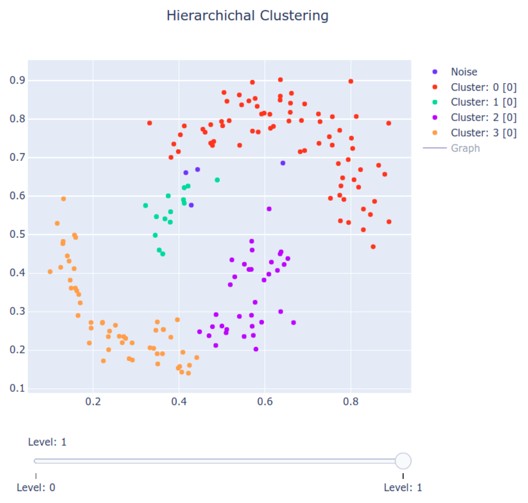
(a) Complete representative $\hat{P}(X_1, 1, 0.95) = 1$; $d_{\mathcal{C}} = 1$



(b) Complete representative $\hat{P}(X_{55}, 0.723, 0.95) = 0.113$; $d_{\mathcal{C}} = 0.022$

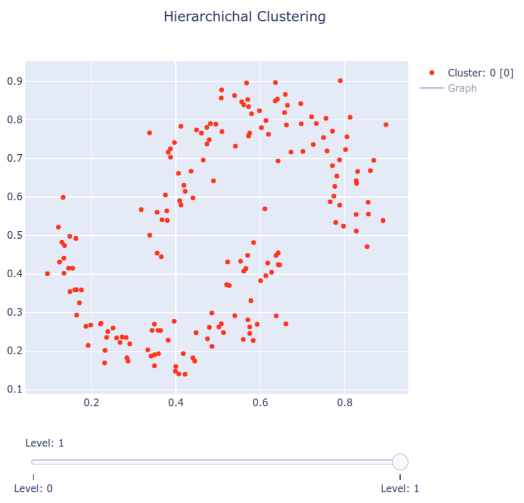


(c) Complete representative $\hat{P}(X_{50}, 0.98, 0.95) = 0.699$; $d_{\mathcal{C}} = 1.000$

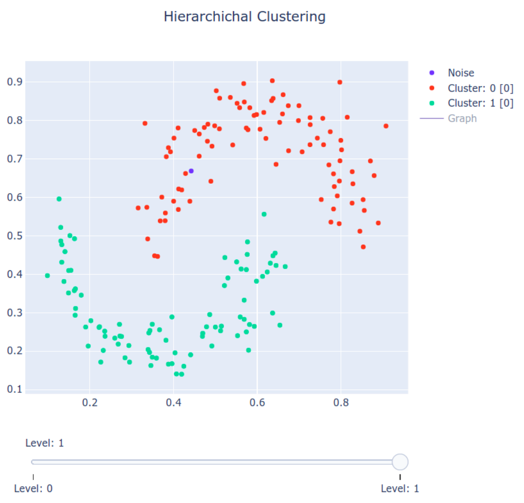


(d) Complete representative $\hat{P}(X_{59}, 0.413, 0.95) = 0.049$; $d_{\mathcal{C}} = 0.395$

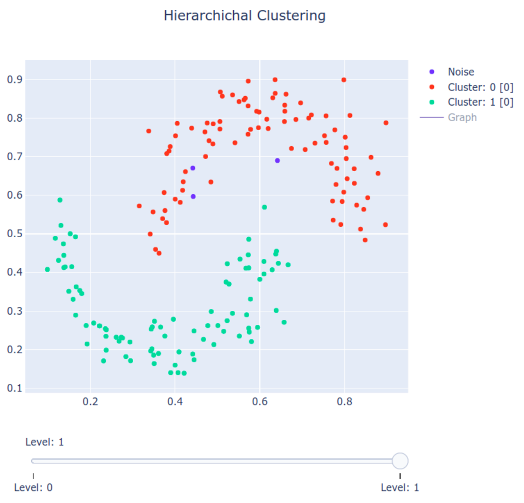
Figure 5.16: Four complete representatives for the *2Moons* dataset. While results were slightly better than those for only 2 representatives, these still have limited practicality.



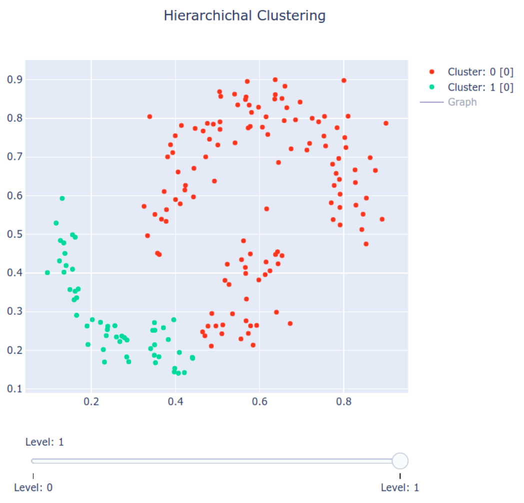
(a) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_1, 0.1, 0.95) = 0.007$; $d_{\mathcal{C}} = 1$



(b) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_{22}, 0.1, 0.95) = 0$; $d_{\mathcal{C}} = 0.022$



(c) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_4, 0.1, 0.95) = 0.382$; $d_{\mathcal{C}} = 0.022$



(d) τ_{max} representative ($\tau_{max} = 0.1$)
$\hat{P}(X_3, 0.1, 0.95) = 0.027$; $d_{\mathcal{C}} = 0.640$

Figure 5.17: Four τ_{max} representatives for the *2Moons* dataset. Limiting the maximum distance allowed to the ground-truth clustering again proves to improve substantially the statistical usefulness of the approach.

a user to retrieve with various degrees of confidence good clustering candidates from uncertain databases, with the τ_{max} strategy consistently outperforming the more direct complete representative selection and obtaining statistically significant results which a user can use to make an informed, sound decision.

In all these contributions the focus was placed on the end user, with automated processes, visualization options and ease of use being put at the forefront, while of course not neglecting the algorithmic and performance aspects that make for good scientific implementations.

6 Conclusions and Outlook

In this thesis, we have taken a multi-faceted approach to exploring and expanding the use of sparse grids to solving various learning tasks, examining the relationships between algorithmic development, performance aspects, and user-oriented implementation. We began in Chapter 3 with the introduction of what we called complex density estimators, i.e., sparse grid-based algorithms to approximate nontrivial functions of one or two probability densities. Previously only the estimation of a single density, via the approach of Peherstorfer [60], was possible. Our work in this direction was inspired by the existence of successful kernel estimators for solving these tasks.

Through our tests on a variety of datasets of various dimensionality, sampled from simple and mixed distributions, we have found that our density difference, ratio, and relative ratio estimators can equal and even outperform qualitatively their kernel-based counterparts. Not only were we able to produce accurate approximations close to the input data points, a trademark of kernel-based methods, but, due to the grid-based nature of our approach, our sparse grid algorithms were able to capture the target function in the whole computational domain. Unfortunately, our density derivative estimator obtained relatively modest results in our tests. We as of now have not been able to pinpoint the causes of the poor visual and numerical results of this approach, whether the stem from the requirement to use higher degree bases, the specific not-a-knot B-splines used, or , therefore further analysis will be required in order to fully assess our methods capabilities compared to the similar existing kernel estimator. While also requiring the use of the more computationally expensive not-a-knot B-spline basis, the density derivative ratio estimator however proved to be quite robust, most likely due to its more expensive, data-dependent resulting system matrix, obtaining overall qualitatively better results than the corresponding kernel estimator we have compared it against.

Future improvements and extensions to our complex density estimators will be in no small degree helped by the fact that they have been integrated into the SG++ data mining pipeline [69]. A concrete first step forward would be the implementation of cross-validation-based hyperparameter selection strategy, similar to the existing ones for other learning tasks in the pipeline, which is not a trivial task for the estimators requiring two input datasets. Also, implementation and study of te applicability of concept drift strategies, as outlined in Appendix A, could be of interest. As our delving in this density estimation research direction was inspired by the large number of applications of kernel-based counterparts (e.g., [78, 80, 81, 48]), we hope that our newly introduced density-based estimators will also find similar use, expanding the overall reach of sparse grids.

In Chapter 4 we took a more HPC-oriented look at the tasks of regression and classification using sparse grids, and their applications. We first presented our optimization study of the legacy code of Heinecke [34] geared towards the recent KNL architecture. While more academic in nature, through our optimization process we obtained both competitive speed-ups of upwards of 1.7× compared to the previous Haswell CPU architecture, as well as corroborating various results and guidelines found in literature regarding best settings for the KNL processor when running similar applications. In the second part of the chapter, this optimized code was used in our separate study of sparse grid-based time series prediction, applied to real-world currency exchange rates. Going beyond the replication of the results of Garcke et al. [24, 25] on equivalent freely available raw data, instead of the original commercial source, we expanded the existing work in this direction to encompass also the use of modified basis functions, as well as (spatially) adaptive sparse grids. With all reasonable steps taken to ensure our results can be compared to those obtained by Garcke et al., our tests revealed that for this type of financial data there is no need for boundary points in the utilized sparse grids, with modified basis functions obtaining even superior prediction results. On the other hand, spatial adaptivity using the generic surplus-based strategy did not produce any significant improvement, stemming in our opinion from the intrinsic distribution of data points in embedded space. In the end, we obtained a better solution to the currency exchange rate prediction problem with the aid of the short simulation time provided by the optimized regression code, something which can open the road towards the use of sparse grids for profitable trading strategies in an online setting as well.

Our third direction of study was presented in Chapter 5, where we focused on the usability aspects of sparse grid-based clustering methods. To this end, we have started by integrating the previous approach of Peherstorfer [61] into the SG++ data mining pipeline, then provided two extensions in the form of an integrated hierarchical clustering algorithm and an uncertain clustering Python-based add-on. The numerical simulations carried out showed the ability of our hierarchical clustering to reveal useful new insights into datasets, with a good visualization strategy, four implemented clustering metrics, and newly introduced pipeline parameters providing novel tools for a user to produce and analyze high quality clusters. In line with our usability perspective to the task of clustering, we introduced in the second part of the chapter the first (to our knowledge) uncertain clustering algorithm using sparse grids, by combining a known world representatives approach with our pipeline's standard clustering method. The results from our tests confirmed that, by using this algorithm, a user can be provided with statistically significant clustering representatives when working with uncertain data, mirroring existing results using other underlying clustering algorithms, like DBSCAN [92].

For our new clustering methods, some concrete future steps of improvement can already be outlined. The hierarchical clustering implementation, while driven mainly by the usability concerns, can benefit for more performance-oriented upgrades, beyond what is currently provided "as is" by the data mining pipeline. For example, a parallelized hash-based algorithm for the construction of the nearest neighbor graph could be implemented, as proposed in [62]. Additionally, the runtime of the graph operations could be improved by using distributed implementations of the vantage-point tree algorithms. Lastly, the cluster splitting process can also be distributed across the children, providing more performance from an otherwise overall sequential cluster hierarchy construction algorithm. For the uncertain clustering method, while limited additional (parallel) performance from what was already presented could be obtained, more work can be done in understanding the limits of world representatives approach when applied to the sparse grid-based clustering, as well as testing out a possible uncertain hierarchical clustering extension of the method.

With an array of brand new algorithms, various study cases based on existing and new methods and implementations, as well as a multi-faceted approach to numerical solutions, our work tried, and we believe it also succeeded, to provide more insight into various learning algorithms using sparse grids, while also opening new exciting avenues of research.

List of Figures

List of Tables

Acronyms

AoS	Array-of-structures.
ARI	Adjusted Rand index.
AVX	Advanced Vector Extensions.
BGL	Boost Graph Library.
CDF	Cumulative distribution function.
CG	Conjugate gradient.
CHI	Caliński-Harabasz index.
CLSDD	Constrained least squares density difference.
CPU	Central processing unit.
CSV	Comma separated values.
DBI	Davies-Bouldin index.
DDE	Density difference estimation.
DDerivE	Density derivative estimation.
DDerivRE	Density derivative ratio estimation.
DDR (SDRAM)	Double Data Rate (Synchronous Dynamic Random-Access Memory).
DRE	Density ratio estimation.
EUR	European Union euro (€).
FMA	Fused multiply-add.
FMI	Fowlkes-Mallows index.
GBP	British pound sterling (£).
GPU	Graphics processing unit.
HBM	High-bandwith memory.
HPC	High-performance computing.
HTML	HyperText Markup Language.
IO	Input-output.
IPCC	Intel Parallel Computing Center.
ISO	International Organisation for Standarditsation.

JSON	JavaScript Object Notation.
KDE	Kernel density estimation.
KLIEP	Kullback-Leibler importance estimation procedure.
KMM	Kernel mean matching.
KNC	Knight's Corner.
KNL	Knight's Landing.
LRZ	Leibniz Supercomputing Center (German: Leibniz-Rechenzentrum).
LSDD	Least squares density difference.
LSDDR	Least squares density derivative ratios.
LSH	Locality-sensitive hashing.
LSIF	Least squares importance fitting.
MAE	Mean absolute error.
MaxAE	Maximum absolute error.
MCDRAM	Multi-Channel Dynamic Random-Access Memory.
MISE	Mean integrated squared error.
MISED	Mean integrated squared error for derivatives.
MPI	Message Ppassing Interface.
NUMA	Non-uniform memory access.
OpenCL	Open Computing Language.
OpenMP, OMP	Open Multi-Processing.
OS	Operating system.
PAC	Partition around medoids.
PCA	Principal component analysis.
PDE	Partial differential equation.
PDF	Probability density function.
RDRE	Relative density ratio estimation.
RMSE	Root mean squared error.
RuLSIF	Relative unconstrained least squares importance fitting.
ScaLAPACK	Scalable LAPACK (Linear Algebra PACKage).
SGDDE	Saprse grid density difference estimation.
SGDDerivE	Sparse grid density derivative estimation.
SGDDerivRE	Sparse grid density derivative ratio estimation.
SGDE	Sparse grid(-based) density estimation.

SGDRE	Sparse grid density ratio estimation.
SGRDRE	Sparse grid relative density ratio estimation.
SIMD	Single-instruction multiple-data.
SNC	Sub-NUMA clustering.
SoA	Structure-of-arrays.
t-SNE	t-distributed stochastic neighbor.
UCI	University of California, Irvine.
uLSIF	Unconstrained least squares importance fitting.
USD	United States dollar ($).
VM	V-measure.
VP	Vantage-point.
VPU	Vector processing unit.

Bibliography

[1] V. B. B. Anguiano. *Visualization of High Dimensional Models within the SG++ Data Mining Pipeline.* Student research paper, Technical University of Munich, Oct. 2019.

[2] V. B. B. Anguiano. Integration and Visualization of Sparse-Grid based Clustering Methods in the SG++ Datamining Pipeline. Master's thesis, Technical University of Munich, Apr. 2020.

[3] A. Azzalini and A. Capitanio. Distributions generated by perturbation of symmetry with emphasis on a multivariate skew t-distribution. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(2):367–389, 2003. _eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/1467-9868.00391. URL: `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00391`, `doi:https://doi.org/10.1111/1467-9868.00391`.

[4] A. Azzalini and A. D. Valle. The multivariate skew-normal distribution. *Biometrika*, 83(4):715–726, Dec. 1996. _eprint: https://academic.oup.com/biomet/article-pdf/83/4/715/702865/83-4-715.pdf. `doi:10.1093/biomet/83.4.715`.

[5] J. Bäck, F. Nobile, L. Tamellini, and R. Tempone. Stochastic spectral Galerkin and collocation methods for PDEs with random coefficients: a numerical comparison. In J. Hesthaven and E. Ronquist, editors, *Spectral and High Order Methods for Partial Differential Equations*, volume 76 of *Lecture Notes in Computational Science and Engineering*, pages 43–62. Springer, 2011. URL: `https://sites.google.com/view/sparse-grids-kit/home`.

[6] R. Bellman. *Adaptive control processes: a guided tour.* Princeton University Press Princeton, N.J, 1961. Type: Book.

[7] S. Ben-David and M. Ackerman. Measures of Clustering Quality: A Working Set of Axioms for Clustering. In *Advances in Neural Information Processing Systems*, volume 21, page 8. Curran Associates, Inc., 2009. URL: `https://proceedings.neurips.cc/paper/2008/file/beed13602b9b0e6ecb5b568ff5058f07-Paper.pdf`.

[8] B. Bohn. On the Convergence Rate of Sparse Grid Least Squares Regression. In J. Garcke, D. Pflüger, C. G. Webster, and G. Zhang, editors, *Sparse Grids*

and Applications - Miami 2016, volume 123, pages 19–41. Springer International Publishing, Cham, 2018. URL: `http://link.springer.com/10.1007/978-3-319-75426-0_2`, `doi:10.1007/978-3-319-75426-0_2`.

[9] B. Bohn and M. Griebel. An Adaptive Sparse Grid Approach for Time Series Prediction. In J. Garcke and M. Griebel, editors, *Sparse Grids and Applications*, volume 88, pages 1–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: `http://link.springer.com/10.1007/978-3-642-31703-3_1`, `doi:10.1007/978-3-642-31703-3_1`.

[10] B. Bohn, M. Griebel, and J. Oettershagen. Optimally Rotated Coordinate Systems for Adaptive Least-squares Regression on Sparse Grids. _eprint: 1810.06749, 2019.

[11] H.-J. Bungartz and M. Griebel. Sparse Grids. *Acta Numerica*, 13:147–269, May 2004. URL: `http://www.journals.cambridge.org/abstract_S0962492904000182`, `doi:10.1017/S0962492904000182`.

[12] T. Caliński and J. Harabasz. A dendrite method for cluster analysis. *Communications in Statistics - Theory and Methods*, 3(1):1–27, Jan. 1974. Publisher: Taylor & Francis. URL: `https://www.tandfonline.com/doi/abs/10.1080/03610927408827101`, `doi:10.1080/03610927408827101`.

[13] D. L. Davies and D. W. Bouldin. A Cluster Separation Measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, Apr. 1979. URL: `http://ieeexplore.ieee.org/document/4766909/`, `doi:10.1109/TPAMI.1979.4766909`.

[14] T. DeFreitas, H. Saddiki, and P. Flaherty. GEMINI: a computationally-efficient search engine for large gene expression datasets. *BMC Bioinformatics*, 17(1):102, Dec. 2016. URL: `http://www.biomedcentral.com/1471-2105/17/102`, `doi:10.1186/s12859-016-0934-8`.

[15] D. Dua and C. Graff. UCI Machine Learning Repository, 2017. URL: `http://archive.ics.uci.edu/ml`.

[16] P. Esling and C. Agon. Time-Series Data Mining. *ACM Comput. Surv.*, 45(1), Dec. 2012. `doi:10.1145/2379776.2379788`.

[17] I.-G. Farcas. *Context-aware Model Hierarchies for Higher-dimensional Uncertainty Quantification.* Dissertation, Technical University of Munich, Munich, Germany, 2020.

[18] M. Fischer. A Recommender System using Clustering with Sparse Grid Density Estimation. Master's thesis, Technical University of Munich, Apr. 2016.

[19] E. B. Fowlkes and C. L. Mallows. A Method for Comparing Two Hierarchical Clusterings. *Journal of the American Statistical Association*, 78(383):553–569, 1983. Publisher: [American Statistical Association, Taylor & Francis, Ltd.]. URL: `http://www.jstor.org/stable/2288117`, `doi:10.2307/2288117`.

[20] B. Fraboni and David Coeurjolly. vptree-draw, Mar. 2021. URL: `https://github.com/bfraboni/vptree-draw`.

[21] F. Franzelin. *Data-driven uncertainty quantification for large-scale simulations.* Dissertation, University of Stuttgart, Stuttgart, Germany, 2017. ISBN: 978-3-8439-3696-5.

[22] J. Garcke. Regression with the Optimised Combination Technique. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 321–328, Pittsburgh, Pennsylvania, 2006. ACM Press. URL: `http://portal.acm.org/citation.cfm?doid=1143844.1143885`, `doi:10.1145/1143844.1143885`.

[23] J. Garcke. Sparse Grids in a Nutshell. In J. Garcke and M. Griebel, editors, *Sparse Grids and Applications*, volume 88, pages 57–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. URL: `http://link.springer.com/10.1007/978-3-642-31703-3_3`, `doi:10.1007/978-3-642-31703-3_3`.

[24] J. Garcke, T. Gerstner, and M. Griebel. Intraday Foreign Exchange Rate Forecasting using Sparse Grids. In J. Garcke and M. Griebel, editors, *Sparse grids and applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 81–105. Springer, 2013. `doi:10.1007/978-3-642-31703-3_4`.

[25] J. Garcke, T. Gerstner, and M. Griebel. Time Series Forecasting using Sparse Grids. Study, Fraunhofer SCAI, Bonn, 2013. URL: `http://publica.fraunhofer.de/dokumente/N-326438.html`.

[26] J. Garcke and M. Griebel. Data Mining with Sparse Grids using Simplicial Basis Functions. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, pages 87–96, San Francisco, California, 2001. ACM Press. URL: `http://portal.acm.org/citation.cfm?doid=502512.502528`, `doi:10.1145/502512.502528`.

[27] J. Garcke and M. Griebel. Semi-supervised Learning with Sparse Grids. In *Proc. of the 22nd ICML Workshop on Learning with Partially Classified Training Data*, page 13, 2005.

[28] J. Garcke, M. Griebel, and M. Thess. Data Mining with Sparse Grids. *Computing*, 67(3):225–253, Oct. 2001. URL: `http://link.springer.com/10.1007/s006070170007`, `doi:10.1007/s006070170007`.

[29] T. Gerstner and M. Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3):209, Jan. 1998. `doi:10.1023/A:1019129717644`.

[30] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods: part I. *ACM SIGMOD Record*, 31(2):40–45, June 2002. URL: `https://dl.acm.org/doi/10.1145/565117.565124`, `doi:10.1145/565117.565124`.

[31] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering Validity Checking Methods: Part II. *SIGMOD Rec.*, 31(3):19–27, Sept. 2002. Place: New York, NY, USA Publisher: Association for Computing Machinery. `doi:10.1145/601858.601862`.

[32] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson. How Do Scientists Develop and Use Scientific Software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8, Vancouver, BC, Canada, May 2009. IEEE. URL: `http://ieeexplore.ieee.org/document/5069155/`, `doi:10.1109/SECSE.2009.5069155`.

[33] M. Hegland, G. Hooker, and S. Roberts. Finite Element Thin Plate Splines in Density Estimation. *ANZIAM Journal*, 42:712, July 2009. URL: `http://journal.austms.org.au/ojs/index.php/ANZIAMJ/article/view/2232`, `doi:10.21914/anziamj.v42i0.2232`.

[34] A. Heinecke. *Boosting Scientific Computing Applications through Leveraging Data Parallel Architectures.* Verlag Dr. Hut, München, 2014.

[35] G. Hinton and S. Roweis. Stochastic Neighbor Embedding. In *Advances in Neural Information Processing Systems*, volume 15, page 8. MIT Press, 2003. URL: `https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf`.

[36] M. Hofert. On Sampling from the Multivariate t Distribution. *The R Journal*, 5(2):129–136, 2013. `doi:10.32614/RJ-2013-033`.

[37] J. Huang, A. Gretton, K. M. Borgwardt, B. Schölkopf, and A. J. Smola. Correcting Sample Selection Bias by Unlabeled Data. In *Advances in Neural Information Processing Systems*, volume 19, page 8. MIT Press, 2007. URL: `https://proceedings.neurips.cc/paper/2006/file/a2186aa7c086b46ad4e8bf81e2a3a19b-Paper.pdf`.

[38] R. Izbicki, A. Lee, and C. Schafer. High-Dimensional Density Ratio Estimation with Extensions to Approximate Likelihood Computation. In *Proceedings of the International Conference on Artificial Intelligence and Statistics, AISTATS*, 2014.

[39] I. A. Jabbie, G. Owen, and B. Whiteley. Performance comparison of Intel Xeon Phi Knights Landing. *SIAM Undergraduate Research Online (SIURO)*, 10:14, 2017.

[40] Jochen Garcke. *Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern.* Dissertation, Rheinische Friedrich-Wilhelms-Universität Bonn, Germany, 2004. URL: `https://hdl.handle.net/20.500.11811/2063`.

[41] T. W. Jones. *Estimation of a Density Function with Applications to Reliability.* Dissertation, Virginia Polytechnic Institute and State University, Virginia, United States, 1973. URL: `http://hdl.handle.net/10919/74166`.

[42] José E. Chacón and Tarn Duong. Data-driven Density Derivative Estimation with Applications to Nonparametric Clustering and Bump Hunting. *Electronic Journal of Statistics*, 7(none):499–532, Jan. 2013. `doi:10.1214/13-EJS781`.

[43] T. Kanamori, S. Hido, and M. Sugiyama. A Least-Squares Approach to Direct Importance Estimation. *J. Mach. Learn. Res.*, 10:1391–1445, Dec. 2009.

[44] D. Kelly, D. Hook, and R. Sanders. Five Recommended Practices for Computational Scientists Who Write Software. *Computing in Science & Engineering*, 11(5):48–53, Sept. 2009. URL: `http://ieeexplore.ieee.org/document/5228715/`, `doi:10.1109/MCSE.2009.139`.

[45] A. Klimke and B. Wohlmuth. Algorithm 847: Spinterp: Piecewise Multilinear Hierarchical Sparse Grid Interpolation in MATLAB. *ACM Trans. Math. Softw.*, 31(4):561–579, Dec. 2005. Place: New York, NY, USA Publisher: Association for Computing Machinery. `doi:10.1145/1114268.1114275`.

[46] J. Kremer, F. Gieseke, K. Steenstrup Pedersen, and C. Igel. Nearest Neighbor Density Ratio Estimation for Large-scale Applications in Astronomy. *Astronomy and Computing*, 12:67–72, Sept. 2015. URL: `https://linkinghub.elsevier.com/retrieve/pii/S2213133715000657`, `doi:10.1016/j.ascom.2015.06.005`.

[47] N. Kumar, L. Zhang, and S. Nayar. What Is a Good Nearest Neighbors Algorithm for Finding Similar Patches in Images? In D. Forsyth, P. Torr, and A. Zisserman, editors, *Computer Vision – ECCV 2008*, volume 5303, pages 364–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. Series Title: Lecture Notes in Computer Science. URL: `http://link.springer.com/10.1007/978-3-540-88688-4_27`, `doi:10.1007/978-3-540-88688-4_27`.

[48] S. Liu, M. Yamada, N. Collier, and M. Sugiyama. Change-Point Detection in Time-Series Data by Relative Density-Ratio Estimation. *Neural Networks*, 43:72–83, July 2013. arXiv: 1203.0453. URL: `http://arxiv.org/abs/1203.0453`, `doi:10.1016/j.neunet.2013.01.012`.

[49] R. J. Lyon, B. W. Stappers, S. Cooper, J. M. Brooke, and J. D. Knowles. Fifty Years of Pulsar Candidate Selection: From simple filters to a new principled real-time classification approach. *Monthly Notices of the Royal Astronomical Society*, 459(1):1104–1123, June 2016. _eprint: https://academic.oup.com/mnras/article-pdf/459/1/1104/8115310/stw656.pdf. doi:10.1093/mnras/stw656.

[50] M. Mahmoudi. Density derivative estimation for stationary and strongly mixing data. *Alexandria Engineering Journal*, 59(4):2323–2330, 2020. URL: https://www.sciencedirect.com/science/article/pii/S1110016820300880, doi:https://doi.org/10.1016/j.aej.2020.02.024.

[51] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008.

[52] I. Markov. VP-tree: Content-Based Image Indexing. In *Proceedings of SYRCoDIS*, page 4, Moscow, Russia, 2007. event-place: Moscow, Russia. doi:http://ceur-ws.org/Vol-256/submission_12.pdf.

[53] T. D. Nguyen, M. C. Du Plessis, T. Kanamori, and M. Sugiyama. Constrained Least-Squares Density-Difference Estimation. *IEICE Transactions on Information and Systems*, E97.D(7):1822–1829, 2014. URL: http://jlc.jst.go.jp/DN/JST.JSTAGE/transinf/E97.D.1822?lang=en&from=CrossRef&type=abstract, doi:10.1587/transinf.E97.D.1822.

[54] D. Nichols and M. Twidale. The Usability of Open Source Software. *First Monday*, 8(1), Jan. 2003. URL: https://journals.uic.edu/ojs/index.php/fm/article/view/1018, doi:10.5210/fm.v8i1.1018.

[55] M. Obersteiner and H.-J. Bungartz. A Generalized Spatially Adaptive Sparse Grid Combination Technique with Dimension-wise Refinement. *SIAM Journal on Scientific Computing*, 43(4):A2381–A2403, July 2021. doi:10.1137/20m1325885.

[56] C. M. Pancake. Improving the Usability of Numerical Software through User-Centered Design. In *in The Quality of Numerical Software: Assessment and Enhancement*, pages 44–60. Hall, 1998.

[57] E. Parzen. On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics*, 33(3):1065 – 1076, 1962. Publisher: Institute of Mathematical Statistics. doi:10.1214/aoms/1177704472.

[58] B. Peherstorfer. *Model Order Reduction of Parametrized Systems with Sparse Grid Learning Techniques*. Dissertation, Technical University of Munich, Munich, 2013.

[59] B. Peherstorfer, F. Franzelin, D. Pflüger, and H.-J. Bungartz. Classification with Probability Density Estimation on Sparse Grids. In J. Garcke and D. Pflüger, editors, *Sparse Grids and Applications - Munich 2012*, pages 255–270, Cham, 2014. Springer International Publishing.

[60] B. Peherstorfer, D. Pflüge, and H.-J. Bungartz. Density Estimation with Adaptive Sparse Grids for Large Data Sets. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 443–451. Society for Industrial and Applied Mathematics, Apr. 2014. URL: `https://epubs.siam.org/doi/10.1137/1.9781611973440.51`, `doi:10.1137/1.9781611973440.51`.

[61] B. Peherstorfer, D. Pflüger, and H.-J. Bungartz. Clustering Based on Density Estimation with Sparse Grids. In B. Glimm and A. Krüger, editors, *KI 2012: Advances in Artificial Intelligence*, pages 131–142, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[62] D. Pfander, G. Daiß, and D. Pflüger. Heterogeneous Distributed Big Data Clustering on Sparse Grids. *Algorithms*, 12(3):60, Mar. 2019. URL: `https://www.mdpi.com/1999-4893/12/3/60`, `doi:10.3390/a12030060`.

[63] D. Pfander, A. Heinecke, and D. Pflüger. A New Subspace-Based Algorithm for Efficient Spatially Adaptive Sparse Grid Regression, Classification and Multi-evaluation. In J. Garcke and D. Pflüger, editors, *Sparse Grids and Applications - Stuttgart 2014*, pages 221–246, Cham, 2016. Springer International Publishing.

[64] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems.* Verlag Dr. Hut, Munich, 2010.

[65] F. Queiroz, R. Silva, J. Miller, S. Brockhauser, and H. Fangohr. *Track 1 Paper: Good Usability Practices in Scientific Software Development.* figshare, Aug. 2017. URL: `https://figshare.com/articles/journal_contribution/Track_1_Paper_Good_Usability_Practices_in_Scientific_Software_Development/5331814/3`, `doi:10.6084/m9.figshare.5331814.v3`.

[66] A. Rosenberg and J. Hirschberg. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL: `https://www.aclweb.org/anthology/D07-1043`.

[67] M. Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956. Publisher: Institute of Mathematical Statistics. `doi:10.1214/aoms/1177728190`.

[68] G. v. Rossum and F. L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[69] K. M. Röhner. *Learning from Data with Geometry-Aware Sparse Grids*. Dissertation, Technical University of Munich, Munich, 2020.

[70] P.-C. Sârbu and H.-J. Bungartz. Optimization of a Sparse Grid-Based Data Mining Kernel for Architectures Using AVX-512. In *2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 364–371, Lyon, France, Sept. 2018. IEEE. ISSN: 1550-6533. URL: `https://ieeexplore.ieee.org/document/8645913/`, `doi:10.1109/CAHPC.2018.8645913`.

[71] H. Sasaki, T. Kanamori, A. Hyvärinen, G. Niu, and M. Sugiyama. Mode-Seeking Clustering and Density Ridge Estimation via Direct Estimation of Density-Derivative-Ratios. *Journal of Machine Learning Research*, 18(180):1–47, 2018. URL: `http://jmlr.org/papers/v18/17-380.html`.

[72] H. Sasaki, Y.-K. Noh, and M. Sugiyama. Direct Density-Derivative Estimation and Its Application in KL-Divergence Approximation. In G. Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 809–818, San Diego, California, USA, May 2015. PMLR. URL: `http://proceedings.mlr.press/v38/sasaki15.html`.

[73] E. Schubert and P. J. Rousseeuw. Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms. *Information Systems*, 101:101804, 2021. URL: `https://www.sciencedirect.com/science/article/pii/S0306437921000557`, `doi:https://doi.org/10.1016/j.is.2021.101804`.

[74] W. Sickel and T. Ullrich. Spline interpolation on sparse grids. *Applicable Analysis*, 90(3-4):337–383, 2011. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00036811.2010.495336. `doi:10.1080/00036811.2010.495336`.

[75] J. Siek, L.-Q. Lee, and A. Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., USA, 2002.

[76] S. A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Doklady Akademii Nauk SSSR*, 148:1042–1045, 1963. Publisher: Academy of Sciences of the Union of Soviet Socialist Republics - USSR (Akademiya Nauk SSSR).

[77] M. Stoyanov, D. Lebrun-Grandie, J. Burkardt, and D. Munster. Tasmanian, Sept. 2013. Published: [Computer Software]

https://doi.org/10.11578/dc.20171025.on.1087. doi:10.11578/dc.20171025.on.1087.

[78] M. Sugiyama, T. Kanamori, T. Suzuki, M. C. d. Plessis, S. Liu, and I. Takeuchi. Density-Difference Estimation. *Neural Computation*, 25(10):2734–2775, Oct. 2013. URL: https://www.mitpressjournals.org/doi/abs/10.1162/NECO_a_00492, doi:10.1162/NECO_a_00492.

[79] M. Sugiyama, S. Nakajima, H. Kashima, P. V. Bünau, and M. Kawanabe. Direct Importance Estimation with Model Selection and Its Application to Covariate Shift Adaptation. In *NIPS*, 2007.

[80] M. Sugiyama, T. Suzuki, and T. Kanamori. Density Ratio Estimation: A Comprehensive Review. *RIMS Kokyuroku*, pages 10–31, 2010.

[81] M. Sugiyama, T. Suzuki, and T. Kanamori. *Density Ratio Estimation in Machine Learning*. Cambridge University Press, Cambridge, 2012. URL: http://ebooks.cambridge.org/ref/id/CBO9781139035613, doi:10.1017/CBO9781139035613.

[82] N. Tomašev and M. Radovanović. Clustering Evaluation in High-Dimensional Data. In M. E. Celebi and K. Aydin, editors, *Unsupervised Learning Algorithms*, pages 71–107. Springer International Publishing, Cham, 2016. URL: http://link.springer.com/10.1007/978-3-319-24211-8_4, doi:10.1007/978-3-319-24211-8_4.

[83] J. Valentin. *B-Splines for Sparse Grids: Algorithms and Application to Higher-Dimensional Optimization*. Dissertation, University of Stuttgart, Germany, 2019. arXiv:1910.05379, doi:10.18419/opus-10504.

[84] J. Valentin and D. Pflüger. Hierarchical Gradient-Based Optimization with B-Splines on Sparse Grids. In J. Garcke and D. Pflüger, editors, *Sparse Grids and Applications - Stuttgart 2014*, pages 315–336, Cham, 2016. Springer International Publishing.

[85] Veenendaal, G. van. Tree-GP: A Scalable Bayesian Global Numerical Optimization algorithm. Master's thesis, Utrecht University, Uthrecht, Netehrlands, Feb. 2015. URL: https://dspace.library.uu.nl/handle/1874/307362.

[86] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, page 12, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[87] S. Winter, S. Wagner, and F. Deissenboeck. A Comprehensive Model of Usability. In J. Gulliksen, M. B. Harning, P. Palanque, G. C. van der Veer, and J. Wesson, editors, *Engineering Interactive Systems*, pages 106–122, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[88] B. Xiao and G. Biros. Parallel Algorithms for Nearest Neighbor Search Problems in High Dimensions. *SIAM Journal on Scientific Computing*, 38(5):S667–S699, Jan. 2016. URL: `http://epubs.siam.org/doi/10.1137/15M1026377`, `doi:10.1137/15M1026377`.

[89] M. Yamada, T. Suzuki, T. Kanamori, H. Hachiya, and M. Sugiyama. Relative Density-Ratio Estimation for Robust Distribution Comparison. *Neural Computation*, 25(5):1324–1370, May 2013. URL: `https://www.mitpressjournals.org/doi/abs/10.1162/NECO_a_00442`, `doi:10.1162/NECO_a_00442`.

[90] P. N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, USA, 1993. Society for Industrial and Applied Mathematics. event-place: Austin, Texas, USA.

[91] C. Zenger. Sparse Grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Dfferential Equations*, volume 31 of *Notes on Numerical Fluid Mechanics*, pages 241–251. Vieweg, 1991. URL: `https://www5.in.tum.de/pub/zenger91sg.pdf`.

[92] A. Züfle, T. Emrich, K. A. Schmid, N. Mamoulis, A. Zimek, and M. Renz. Representative Clustering of Uncertain Data. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 243–252, New York New York USA, Aug. 2014. ACM. URL: `https://dl.acm.org/doi/10.1145/2623330.2623725`, `doi:10.1145/2623330.2623725`.

A Concept Drift Formulations for Complex Density Estimators

A.1 Density Difference Estimation

Similarly to the formulation for SGDE as described in [69], we can derive a modified version of Eq. (3.13) for a given minimal learning rate $\beta \in [0, 1]$ as:

$$
\begin{aligned}
\boldsymbol{b}^{p(k)} - \boldsymbol{b}^{q(k)} = &\left(\min\left((1-\beta)^{\tilde{M}_p^{(k)}}, \frac{M_p^{(k-1)}}{M_p^{(k)}} \right) \boldsymbol{b}^{p(k-1)} - \right. \\
&\left. \min\left((1-\beta)^{\tilde{M}_q^{(k)}}, \frac{M_q^{(k-1)}}{M_q^{(k)}} \right) \boldsymbol{b}^{q(k-1)} \right) \\
+ &\left(\max\left(1-(1-\beta)^{\tilde{M}_p^{(k)}}, \frac{\tilde{M}_p^{(k)}}{M_p^{(k)}} \right) \tilde{\boldsymbol{b}}^{p(k)} - \right. \\
&\left. \max\left(1-(1-\beta)^{\tilde{M}_q^{(k)}}, \frac{\tilde{M}_q^{(k)}}{M_q^{(k)}} \right) \tilde{\boldsymbol{b}}^{q(k)} \right).
\end{aligned}
\tag{A.1}
$$

This formulation is independent of batch size (due to the exponents used), guaranteeing that each new data point has a weight factor of at least β in the right-hand side.

A.2 Density Derivative Estimation

The batch update rule in Eq. (3.41) can also be modified to account for concept drift. For a given minimal learning rate $\beta \in [0, 1]$, the update rule becomes:

$$
\begin{aligned}
(-1)^{|\boldsymbol{j}|}\partial^{(\boldsymbol{j})}\boldsymbol{b}^{p(k)} = &\min\left((1-\beta)^{\tilde{M}_p^{(k)}}, \frac{M_p^{(k-1)}}{M_p^{(k)}} \right) (-1)^{|\boldsymbol{j}|}\partial^{(\boldsymbol{j})}\boldsymbol{b}^{p(k-1)} \\
+ &\max\left(1-(1-\beta)^{\tilde{M}_p^{(k)}}, \frac{\tilde{M}_p^{(k)}}{M_p^{(k)}} \right) (-1)^{|\boldsymbol{j}|}\partial^{(\boldsymbol{j})}\tilde{\boldsymbol{b}}^{p(k)}.
\end{aligned}
\tag{A.2}
$$

Again, the idea is to guarantee a minimal contribution of factor β for each new data point in the final right-hand side, with an exponential term to account for the batch size.

B Data Distributions for Testing the Complex Density Estimators

In this appendix we give an overview of the mathematical formulations of the different data distributions used in the testing of our complex density estimators in Chapter 3. Here we will provide the general multivariate formulations of the distributions, as the univariate formulations are just edge cases where the various parameters degenerate from vectors/matrices to scalars.

Multivariate normal distribution. The probability density function (PDF) of the d-dimensional multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ at point $\boldsymbol{x} \in \mathbb{R}^d$ is given by:

$$(2\pi)^{-\frac{d}{2}} \det(\boldsymbol{\Sigma})^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}, \tag{B.1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^d$ is the location vector, and $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$ is the positive-definite covariance matrix.

To sample the data distribution and evaluate the density function we have used the existing implementation available in the `scipy.stats` Python package. This code was extended to allow also the evaluation of the gradient of the PDF, needed for the complex estimators that involve derivatives.

Multivariate t-distribution. The probability density function of the d-dimensional multivariate t-distribution $\mathcal{T}(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu)$ at point $\boldsymbol{x} \in \mathbb{R}^d$ is given by:

$$\frac{\Gamma\left(\frac{\nu+d}{2}\right)}{\Gamma\left(\frac{1}{2}\right)(\pi\nu)^{\frac{d}{2}} \det(\boldsymbol{\Sigma})^{\frac{1}{2}}} \left(1 + \frac{(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}{\nu}\right)^{-\frac{\nu+d}{2}}, \tag{B.2}$$

where $\boldsymbol{\mu} \in \mathbb{R}^d$ is the location vector, and $\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$ is the positive-definite scale matrix, and $\nu \in \mathbb{N}^*$ is the degrees of freedom.

No implementation existed of this distribution in the `scipy.stats` Python package at the time, therefore we have created our own, using the same class structure in order to maintain compatibility. The implementation of the PDF and its gradient are straight-forward, following a similar numerical approach as the existing multivariate normal distribution.

To implement the distribution sampler we have used a known stochastic representation of a t-distributed random variable $\boldsymbol{X} \sim \mathcal{T}(\boldsymbol{\mu}, \boldsymbol{\Sigma}, \nu)$ (see, e.g., [36]):

$$\boldsymbol{X} = \boldsymbol{\mu} + \sqrt{W} A \boldsymbol{Z}, \tag{B.3}$$

where $W = \frac{\nu}{\chi^2_\nu}$ with χ^2_ν denoting an independent random variable following a (univariate) chi-squared distribution with ν degrees of freedom, A is the Cholesky factor of the scale matrix $\mathbf{\Sigma}$ (i.e, A is the matrix that satisfies $\mathbf{\Sigma} = AA^T$), and $\boldsymbol{Z} \in \mathbb{R}^d$ is a vector of d independent standard normal random variables (i.e., $\boldsymbol{Z} = (Z_1, \dots, Z_d)$ with $Z_i \sim \mathcal{N}(0,1),\ i \in 1, \dots, d$).

Multivariate skew-normal distribution. Let $\phi_d(\boldsymbol{y};\ \mathbf{\Sigma})$ denote the PDF of the d-dimensional multivariate normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$ evaluated at point $\boldsymbol{y} \in \mathbb{R}^d$ for given covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$ and let $\Phi_1\{\boldsymbol{y}\}$ denote the cumulative distribution function (CDF) of the standard univariate normal distribution $\mathcal{N}(0,1)$ evaluated at point $\boldsymbol{y} \in \mathbb{R}^d$. Then the PDF of a multivariate skew-normal distribution $\mathcal{SN}(\boldsymbol{\xi}, \boldsymbol{\Omega}, \boldsymbol{\alpha})$ at point $\boldsymbol{x} \in \mathbb{R}^d$ is given by:

$$2\ \phi(\boldsymbol{x} - \boldsymbol{\xi};\ \boldsymbol{\Omega})\ \Phi_1 \left\{ \boldsymbol{\alpha}^T \boldsymbol{\omega}^{-1} (\boldsymbol{x} - \boldsymbol{\xi}) \right\}, \tag{B.4}$$

where $\boldsymbol{\xi} \in \mathbb{R}^d$ is the location vector, $\boldsymbol{\Omega} \in \mathbb{R}^{d \times d}$ is the positive-definite scale (or dispersion) matrix, $\boldsymbol{\alpha} \in \mathbb{R}^d$ is the shape (or skewness) vector, and $\boldsymbol{\omega} = \mathrm{diag}(\boldsymbol{\Omega})^{\frac{1}{2}} \in \mathbb{R}^d$ is a vector whose elements are the square roots of the main diagonal values of the scale matrix $\boldsymbol{\Omega}$. We opted here for the distribution formulation of Azzalini and Dalla Valle [4].

No implementation existed of this multivariate distribution in the `scipy.stats` Python package, therefore we have again created our own, maintain compatibility by using the same class structure as the other multivariate distributions. The implementation of the PDF and its gradient are again straight-forward. To implement the distribution sampler we have used the stochastic representation of a skew-normal random variable $\boldsymbol{X} \sim \mathcal{SN}(\boldsymbol{\xi}, \boldsymbol{\Omega}, \boldsymbol{\alpha})$ using the so-called *conditioning method*, as presented in [3].

Multivariate skew-t-distribution. Let $t_d(\boldsymbol{y};\ \boldsymbol{\xi}, \boldsymbol{\Omega}, \nu)$ denote the PDF of the d-dimensional multivariate t-distribution $\mathcal{T}(\boldsymbol{\xi}, \boldsymbol{\Omega}, \nu)$ evaluated at point $\boldsymbol{y} \in \mathbb{R}^d$ and let $T_1\{\boldsymbol{y};\ \tau\{$ denote the cumulative distribution function (CDF) of the standard univariate t-distribution $\mathcal{T}(0,1,\tau)$ with $\tau > 0$ degrees of freedom evaluated at point $\boldsymbol{y} \in \mathbb{R}^d$. Then the PDF of a multivariate skew-t-distribution $\mathcal{ST}(\boldsymbol{\xi}, \boldsymbol{\Omega}, \boldsymbol{\alpha}, \nu)$ at point $\boldsymbol{x} \in \mathbb{R}^d$ is given by:

$$2\ t_d(\boldsymbol{x};\ \boldsymbol{\xi}, \boldsymbol{\Omega}, \nu)\ T_1 \left\{ \boldsymbol{\alpha}^T \boldsymbol{\omega}^{-1} (\boldsymbol{x} - \boldsymbol{\xi}) \left(\frac{\nu + d}{(\boldsymbol{x} - \boldsymbol{\xi})^T \mathbf{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\xi}) + \nu} \right)^{\frac{1}{2}} ;\ \nu + d \right\}, \tag{B.5}$$

where $\boldsymbol{\xi} \in \mathbb{R}^d$ is the location vector, $\boldsymbol{\Omega} \in \mathbb{R}^{d \times d}$ is the positive-definite scale (or dispersion) matrix, $\boldsymbol{\alpha} \in \mathbb{R}^d$ is the shape (or skewness) vector, $\nu \in \mathbb{N}^*$ is the degrees of freedom, and $\boldsymbol{\omega} = \mathrm{diag}(\boldsymbol{\Omega})^{\frac{1}{2}} \in \mathbb{R}^d$ is a vector whose elements are the square roots of the main diagonal values of the scale matrix $\boldsymbol{\Omega}$. We opted here for the distribution formulation of Azzalini and Capitanio [3].

No implementation existed of this multivariate distribution in the `scipy.stats` Python package, therefore we have once more created our own, again using the same class structure as the existing code in order to maintain compatibility. The implementation of the PDF and its gradient are similarly straight-forward as in the case of the skew-normal distribution.

To implement the sampler we have used the stochastic representation of a skew-normal random variable $\boldsymbol{X} \sim \mathcal{ST}(\boldsymbol{\xi}, \boldsymbol{\Omega}, \boldsymbol{\alpha}, \nu)$ described in [3], i.e.,

$$\boldsymbol{X} = \boldsymbol{\xi} + \sqrt{W}\boldsymbol{Z}, \tag{B.6}$$

where $W = \frac{\nu}{\chi^2_\nu}$ with χ^2_ν denoting an independent random variable following a (univariate) chi-squared distribution with ν degrees of freedom, and $\boldsymbol{Z} \in \mathbb{R}^d$ is an independent multivariate skew-normal random variable with $\boldsymbol{\xi} = \boldsymbol{0}$ (i.e., $\boldsymbol{Z} \sim \mathcal{SN}(\boldsymbol{0}, \boldsymbol{\Omega}, \boldsymbol{\alpha})$).