DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Study of Density Difference and Density Ratio Estimation using Sparse Grids

Alexandre Mercier

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Study of Density Difference and Density Ratio Estimation using Sparse Grids

Untersuchung von Dichtedifferenz- und Dichteverhältnisschätzung mit Dünngittern

Author:	Alexandre Mercier
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Paul-Cristian Sarbu M.Sc. (Hons.)
Submission Date:	17.08.2020

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

Munich, 17.08.2020 Alexandre Mercier

Abstract

Using sparse grids for density estimation can reduce computational expenses in comparison to more popular kernel density methods by reducing the amount of examined points, especially for datasets in higher dimensions. The goal of this thesis is to study two recently added algorithms for density difference and density ratio estimation in the SG++ code library for sparse grids. Using a custom pipeline, experiments studying the behavior and accuracy of the algorithms compare the sparse grid results to the analytical solution and to additional kernel based density estimation methods. We aim to visualize and quantify the differences between sparse grid based solutions and other solutions to demonstrate their accuracy and usability for future use in high-dimensional applications and time-series segmentation.

Contents

1 Introduction

Density estimation uses a sample of data points and attempts to reconstruct the probability density function used to create the sample with interpolation techniques. Although density estimation has some applications, most rely on the comparison of two densities. For this purpose two types of functions exist, either density difference estimation functions or density ratio estimation functions.

Because they enable the comparison of densities either as a norm or relatively to another, they are used in many data processing tasks, such as time series segmentation, outlier detection and machine learning.

A naive approach is to use two steps, one to estimate the density of each set of samples and then a second one to calculate their ratio or their difference. This technique has an substantial loss in accuracy because small errors in the first step will only increase the errors caused in the second step. For this reason, better algorithms were developed, which are capable of estimating the ratios and differences in a single step.

They are however limited in the number of dimensions until the number of points exceeds their computational power. Since most evaluate every point in a grid, they are subject to the curse of dimensionality. With each dimension, the number of points in a grid increases exponentially. If a one-dimensional grid has just 10 points, this means that the same grid in five dimensions will have 10^5 points and the algorithms need to evaluate every single one of them. Since some applications (especially data-driven problems) might require hundreds of dimensions a way of reducing the required amount of points is necessary. Sparse grids are able to overcome the curse to some extent.

Pflüger [4] has done a lot of groundwork on spatially adaptive sparse grids for high-dimensional problems and started the code library SG++ specifically for sparse grid calculations. From there Peterstorfer [3] has added techniques to perform multi-dimensional density estimation using sparse grids (SGDE). Expanding on the density estimation techniques, algorithms for density difference and density ratio estimation using sparse grids were added.

The goal of the thesis is to study both the density difference estimation (SGDDE) implementation and the density ratio estimation (SGDRE) implementation from SG++. A pipeline is built to evaluate visually and numerically the effects of the influential parameters for both algorithms. The pipeline is then expanded to compare the estimations to kernel based density estimation methods. The *least-squares density difference* estimation algorithm (LSDD) [7] and *unconstrained least-squares importance fitting* (uLSIF) [2] were chosen for their direct density-difference and -ratio estimation without separate density estimation.

2 Theory

Before we can start analyzing both functions, it is necessary to understand the theory behind sparse grids and how density estimation algorithms can achieve accurate results. Then we will present and examine the code library and the functions which will be studied in the following chapters.

2.1 Sparse Grids

Sparse grids were originally developed by mathematician Sergey Smolyak for the solution of partial differential equations, but they have since then been used in numerous fields of applications. According to *sparsegrids.org*[6], three practical applications of sparse grid methods are the estimation of cosmological redshifts which can be used to calculate the distance of stars to earth, the topology optimization with B-Splines where the strength and elasticity of materials and objects can be computed and the uncertainty quantification for carbon capture and storage. The last application aims to estimate the risk of leaks in locations where underground carbon capture can be performed.

Sparse grids are particularly useful in high-dimensional applications, e.g. interpolation, because of the curse of dimensionality. In mathematics and computer science, the number of dimensions describes the number of metrics used to describe a datapoint. Since most applications require multiple metrics, datasets often end up having a lot of dimensions. When using a full grid to compute a solution, meaning each single point in the space is calculated, the computational load quickly surpasses even the newest computers.

With each added dimension, the complexity of a full grid increases exponentially ($\mathcal{O}(n^2)$). If a grid has just 10 points in a single dimension, with 5 dimensions, a full grid will have 10^5 points which all have to be evaluated. This exponential increase of points causes data to become sparse and datasets need to be reduced in size for a timely computation. This is called the curse of dimensionality. Sparse grids can reduce the effects of the curse by "requiring significantly fewer grid points than a full grid, while preserving the asymptotic error decay of full grid interpolation" [4].

We will restrict ourselves to the d-dimensional unit-hypercube $\Omega = [0,1]^d$ for simplicity in the following explanations. Any rectangular volume can be transformed easily to the unit-hypercube with rescaling. Sparse grids use hierarchical basis functions to interpolate the underlying spaces. When considering a one-dimensional function f, its interpolation is done using the standard hat function,

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

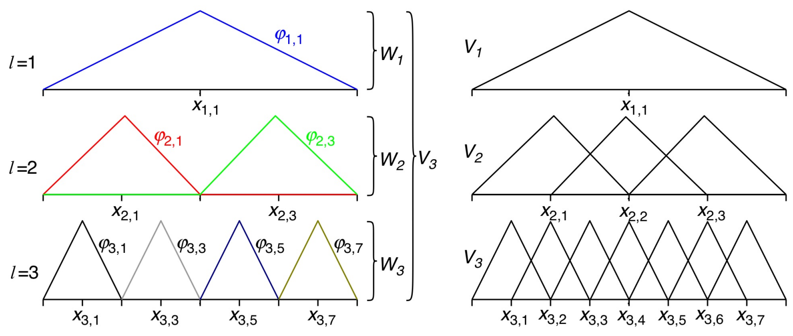

Figure 2.1: One-dimensional basis functions $\varphi_{l,i}$ and the corresponding grid points $x_{l,i}$ up to level n = 3 in the hierarchical basis (left) and the common nodal point basis (right)[4]

.

from which the one-dimensional hat basis function can be derived,

$$\varphi_{l,i}(x) := \varphi(2^l x - i)$$

where l is the level and i ($0 < i < 2^l$) the index of the function [4]. The functions are centered at the interpolation points of f: $x_{l,i} = 2^{-l} \cdot i$. We can then obtain a set of hierarchical subspaces,

$$W_l := \text{span}\{\varphi_{l,i}(x) : i \in I_l\}, \tag{2.1}$$

with I_l being the hierarchical index sets,

$$I_l := \{i \in \mathbb{N} : 1 < i < 2^l - 1, i \text{ odd}\} \tag{2.2}$$

On a full grid, the space of piecewise linear functions V_n is formulated as a sum of W_l [4],

$$V_n = \bigoplus_{l \le n} W_l \tag{2.3}$$

In Figure 2.1 the hierarchical subspaces W_l are displayed up to level $l = 3$ on the left side and the corresponding spaces of piecewise linear basis functions V_n on the right side. The functions in each hierarchical subspace cover the whole domain and have the same size and shape but do not overlap.

Then the interpolation $u(x) \in V_n$ can be written as a finite sum with $\alpha_{l,i}$ as the (hierarchical) surpluses [4]:

$$u(x) = \sum_{l \le n, i \in I_l} \alpha_{l,i} \varphi_{l,i}(x) \tag{2.4}$$

In Figure 2.2 such an interpolation with hierarchical basis functions is constructed. The border functions are ignored here for simplicity.

To extend the basis functions to d-dimensional applications, the tensor product approach is used

$$\varphi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j) \tag{2.5}$$

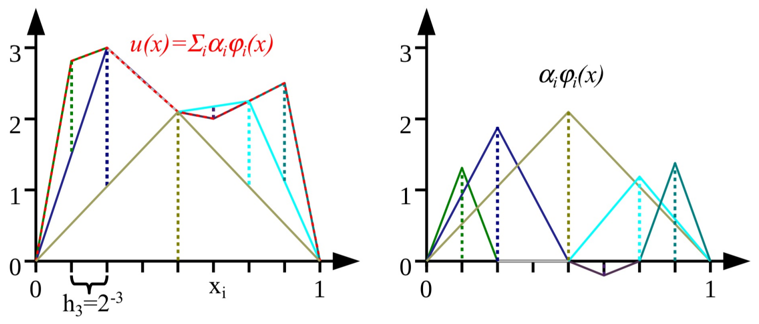

Figure 2.2: The same one-dimensional piecewise linear interpolation $u(x)$ of $f(x)$ as in Figure 2.1 (left), but this time using the hierarchical basis functions, and the corresponding weighted basis functions (right)[4]

.

where $\vec{l}$ and $\vec{i}$ are d-dimensional multi-indices indicate the level and index for each dimension. In this situation, full grids quickly become inefficient and unusable as they will have $(2^n - 1)^d$ grid points.

Therefore sparse grids could be used instead for d-dimensional problems by selecting only "those subspaces that contribute the most to the overall solution of the full-grid interpolation" [4]. The sparse grid space is obtained with,

$$C_n^1 = \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}. \tag{2.6}$$

This leaves out the subspaces from the full grid space which contain basis functions of small support. The resulting sparse grid interpolants $u(\vec{x})\ inV_n^1$ are given by:

$$u(\vec{x}) = \sum_{|\vec{l}_1 \leq n+d-1, \vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \varphi_{\vec{l},\vec{i}}(\vec{x}). \tag{2.7}$$

In Figure 2.3 the a grid of level 3 is shown as well as the selection of subspaces for $n = 3$.

2.1.1 Boundary treatment

The boundaries of sparse grids can be computed in different ways. Until now we ignored the boundary by assuming that the functions were zero at the boundary. In order to handle non-zero boundaries, an extra level, $l = 0$ can be introduced. It has two (overlapping) functions $\varphi_{0,0}$ and $\varphi_{0,1}$ as shown in Figure 2.4 on the left side. By adding the layer 0, we obtain a modified set of subspaces $\tilde{W}_{\vec{l}}$:

$$V_n^{0B(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} \tilde{W}_{\vec{l}}. \tag{2.8}$$

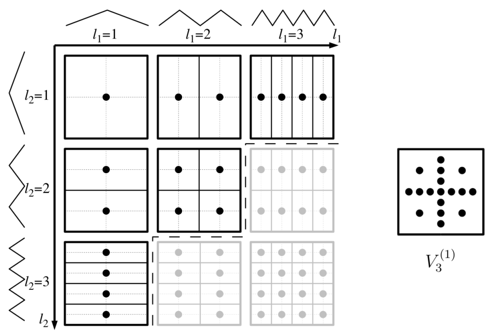

Figure 2.3: The two-dimensional subspaces $W_{\vec{l}}$ up to $l = 3(h_3 = 1/8)$ in each dimension. The optimal a priori selection of subspaces is shown in black (left) and the corresponding sparse grid of level $n = 3$ for the sparse grid space $V_3^{(1)}$ (right). For the full grid, the gray subspaces have to be used as well.[4]

.

The effect of this, is that there are twice as many boundary points inner grid points. In higher dimensions, this means that almost all the points are on boundaries. To mitigate this effect, level 0 and 1 can be collated to reduce the number of subspaces.

$$V_n^{B(1)} := \bigoplus_{|\vec{l}|_1 \leq n+d-1} \hat{W}_{\vec{l}}. \tag{2.9}$$

Doing so, the same number of points are spent on the grid's main axis and on the boundary. The structure is visible on the right in Figure 2.4. Still, this leads to a lot more points than $V_n(1)$, which is why in applications where the boundary behavior is not required, it can be advantageous to omit the boundary points. Alternatively, the interior basis functions can be modified to extrapolate towards the boundaries of the domain. The basis functions can be modified to extrapolate linearly towards the boundary for a grid level superior to 1. For a level 1 grid, the best option is to use a constant basis function. The modified one-dimensional piecewise linear basis functions are shown in Figure 2.5. The d-dimensional basis functions are obtained via tensor product as before. Now that the general concept of sparse grids is clear, we still wish to examine how they can be used to estimate densities.

2.1.2 Sparse Grids based Density Estimation (SGDE)

When a data set S of samples is drawn from an unknown distribution with an unknown probability density function (pdf) f, density estimation algorithms can be used to estimate the density function $\hat{f}$ of f by using the dataset [3]. $\hat{f}$ can then be used in data-mining and statistical applications to extract data or trends from the existing dataset.

According to Peterstorfer [3], a sparse grids based solution is to "employ spline smoothing to derive a smoother and more generalized estimator $\hat{f}$ of a highly-overfitted

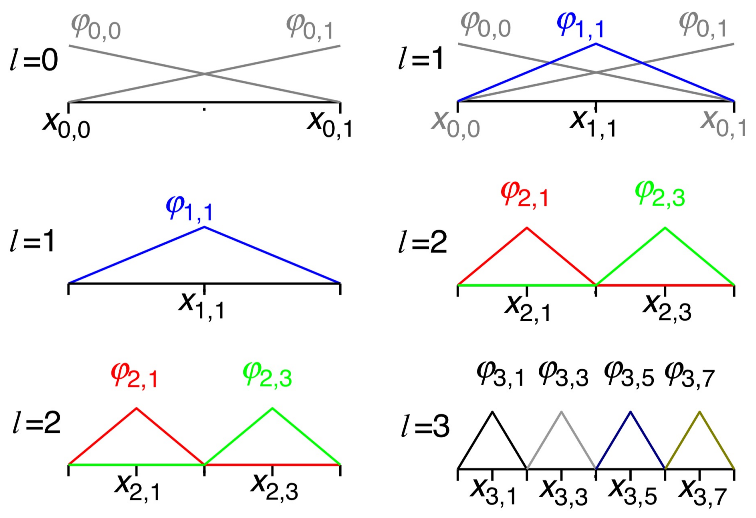

Figure 2.4: One-dimensional hierarchical basis functions $\varphi_{l,i}$ with two basis functions located on level 0 on the boundary (left) and one-dimensional hierarchical basis functions $\varphi_{l,i}$ with level 1 being extended by two extra level 0 basis functions located on the boundary (left)[4]

.

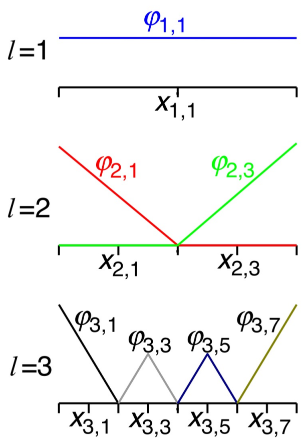

Figure 2.5: The modified one-dimensional basis functions $\varphi_{l,i}$ which are extrapolating towards the boundary. They are constant on level 1 and "folded up" if adjacent to the boundary on all other levels [4].

initial guess f_ϵ" [3]. This boils down to following minimalzation problem:

$$\hat{f} = \underset{u \in V}{argmin} \int_\omega (u(x) - f_\epsilon(x))^2 dx + \lambda \|Lu\|_{L^2}^2 \tag{2.10}$$

where $\|Lu\|_{L^2}^2$ is a regularization term and $\lambda > 0$ the regularization parameter. The parameter controls the smoothness of the curve versus its fidelity. If a large lambda is chosen, the curves will be to smooth and not represent the data anymore, but choosing a very small lambda will cause overfitting, a phenomenon where the estimation is adapted to the density of points of the initial dataset so closely, that the estimation function $\hat{f}$ doesn't represent the pdf f anymore.

Defining $\Phi = \{\phi_1, ..., \phi_N\}$ as the set of hierarchical basis functions of the adaptive sparse grid space $V_l^{(1)}$ of level $l \in \mathbb{N}$, an estimation $\hat{f}_N \in V_l^{(1)}$ is searched such that

$$\int_\Omega \hat{f}_N(x) \cdot \phi(x) dx + \lambda \cdot \int_\Omega L\hat{f}_N(x) \cdot L\phi(x) dx = \frac{1}{M} \sum_{i=1}^{M} \phi(x_i) \tag{2.11}$$

holds for all $\phi \in \Phi$ [3]. Since $\hat{f}_N$ is a linear combination of basis functions in Φ, this boils down to solving a system of linear equations

$$(R + \lambda C)\alpha = b \tag{2.12}$$

with α the coefficients of the basis functions, $R_{ij} = (\phi_i, \phi_j)_{L^2}$,$C_{ij} = (L\phi_i, L\phi_j)_{L^2}$ and $b_i = \frac{1}{M} \sum_{j=1}^{M} \phi_i(x_j)$. The coefficients α are then used as surpluses in the sparse grid.

The methods we are interested in for the thesis expand on normal density estimation to estimate the density difference or the density ratio between two given datasets P and Q. The naive approach would be to use two steps in the algorithms, where the first step generates density estimations $\hat{f}_P$ and $\hat{f}_Q$ of each dataset and then calculates their difference or ratio depending on the application.

However a small error in $\hat{f}_P$ or $\hat{f}_Q$ will lead to larger errors in the final result, which is why a single step approach is preferred. The approach is similar in both cases: the goal is to minimize a cost function consisting of a squared error term and a regularization term. To find a solution, the equation is turned into a variational problem, which can then be solved by applying the sparse grid approximation. The final result is a linear system of equations. The algorithms used in the sg++ library [5] are explained more in detail in subsection 2.2.2 and subsection 2.2.3.

Now that we understand how sparse grids are built and how they differ from a full grid, let's examine the code library implementing the algorithms for sparse grids we will study.

2.2 The SG++ library

To effectively use sparse grids, a code library called SG++ (also referred to as *sgpp*) was started in 2008 by Dirk Pflüger with one of the main publications [4] being released two years later. According to the website, “SG++ is a universal open-source toolbox for spatially adaptive sparse grid methods and the combination technique.” [6]. It provides both basic building blocks for sparse grid applications and higher level functions for specific applications. The project has been developed since then by researchers and students of the University of Stuttgart and the Technical University of Munich, expanding the capabilities of the library. Since the project is open source, anyone can contribute.

The core components of the library are coded in C++, which might explain its name. The language was chosen for its execution speed, reliability and compatibility. Additional matlab, python and java bindings exist for usage of SG++ in these languages. Since a large part of this thesis focuses on data analysis and visualization with specific functions of sg++, the python binding is used here. The python binding is called `pysgpp`. It provides most of the functionalities present in the library inside python but uses the C++ implementation in the background for fast execution.

The library is organized in multiple modules implementing different functionalities. The base module holds all the core functions and is required by all other modules. It is essential since it holds the sparse grid class required to create, configure and refine the grids. Additionally it is used to evaluate data points and includes the underlying data structures.

The other modules are used for specific tasks an applications which build on the base layer. Some of their applications are solving linear equations, optimizing smooth sparse grid interpolants or implementing quadrature algorithms.

2.2.1 The datadriven module

Since this thesis focuses on two functions from the datadriven module, it is worth explaining what this module can do. It regroups all functions related to data mining and machine learning. According to the its description [5], the module regroups all “operations depending on datasets”, adds “data mining support” and “specialized regularization”.

The functions we wish to study are in the datamining submodule. The submodule implements a pipeline “intended for out-of-the-box application of SG++ machine learning methods” [5]. It allows to users to run different algorithms with a simple configuration file (see subsection 2.2.4) without the need to understand or see the underlying functions. Understanding the parameters is nonetheless important as they influence the final result. The pipeline can be used for a lot of different “datamining tasks such as regression, density estimation and classification” [6].

Since the configuration file is used for all pipeline applications, the pipeline can be used without knowing its steps. Understanding how it works is nevertheless still required to use it effectively. As a first step, a factory object is created with the specific application in mind. In our case, this was either the `DensityDifferenceEstimationFactory` or the `DensityRatioEstimationMinerFactory`.

From the factory, a miner is created with the configuration file as input. The miner is then configured according to the file, with missing variables automatically set to default values. The next step is then to train the miner. It takes the datasets specified in the configuration file and uses them to train a model. The model also depends on the other parameters in the configuration file (see subsection 2.2.4).

The training of the model is separated in epochs. During an epoch, both dataset are used to train the model. In the case of SGDDE, this can happen in multiple steps, called batches. After each epoch a refinement step can be added. This step selectively adds grid points where the algorithm believes they are needed. The step can also delete some of the points it just added in the coarsening phase if it believes that the points do not improve the result. After a refinement, a new evaluation is needed to recalculate surpluses of the linear basis functions.

Two epochs without refinement in between do not improve the resulting grid and a refinement step without reevaluation afterwards is equally useless.

From the model, datapoints can be evaluated and the grid points extracted with their surpluses. The evaluation pipeline described chapter 3 does these steps automatically and can then be used to evaluate the result visually or numerically.

2.2.2 Sparse Grids based Density Difference Estimation (SGDDE)

The algorithm for density difference estimation with sparse grids was already implemented and usable before the thesis. By studying it and evaluating the results from different parameters, dataset sizes and distributions asserts its accuracy and is required to uncover errors and bugs. The algorithm takes two datasets, P and Q and computes their density difference $p(x) - q(x)$ where $p(x)$ is the density estimation of P and $q(x)$ the density estimation of Q.

The algorithm attempts to minimize a cost function,

$$J(r) = \int_\Omega \{r(x) - [p(x) - q(x)]\}^2 dx + \lambda \int_\Omega [r''(x)]^2 dx \tag{2.13}$$

consisting of an error function and a regularization term and where $r(x) = \sum_{k=1}^{N} \alpha_K \Phi_K$ The equation can be modified to a variational problem and then be solved as a system of linear equations:

$$(R + \lambda C) \cdot \vec{\alpha} = B_P - B_Q \tag{2.14}$$

with $B_{P,i} = \frac{1}{M_P} \sum_{j=1}^{M_P} \phi_i(x_{P,i})$ and $B_{Q,i} = \frac{1}{M_Q} \sum_{j=1}^{M_Q} \phi_i(x_{Q,i})$. Once the system is solved, the resulting sparse grid is saved and data points can be evaluated.

2.2.3 Sparse Grids based Density Ratio Estimation (SGDRE)

The algorithm for density ratio estimation with sparse grids was also implemented before the thesis. Its implementation is not as straightforward, because calculating the solution takes extra steps. The starting cost function

$$J(r) = \int_\Omega [r(x) - \frac{p(x)}{q(x)}]^2 q(x) dx + \lambda \int_\Omega [r''(x)]^2 dx, \tag{2.15}$$

is similar to the density difference estimation, except the ratio is calculated instead.

The resulting linear system of equations is

$$(\frac{1}{M_q}B_qB_q^T + \lambda C)\vec{\alpha} = \frac{1}{M_p}B_p\vec{e} \tag{2.16}$$

with $e = (1, 1, ..., 1)^T$, $(B_P)_i = \Phi_i(\vec{x_{P}}_j), j = 1..M_P$ and $(B_Q)_i = \Phi_i(\vec{x_{Q}}_j), j = 1..M_Q$ Just as before, the output is a sparse grid but this one estimates the density ratio of P and Q.

2.2.4 Configuration file

Configuration files are the main way of using the pipeline. They are written in JSON format and should contain all the parameters and variables to compute the expected solution. If some parameters are not specified default values are selected instead, so that most of the parameters can be omitted. Since the files cover all types of task and each task uses different parameters, we will focus on the parameters used in the thesis.

A configuration consist of three main parts:

- **The datasource configuration**: Inside, the paths to the datasets are specified and parameters about the datasets can be added to.
- **The scorer configuration**: It holds the metric that should be used evaluate the model. We only used the mean squared error scorer type in the thesis.
- **The fitter configuration**: It specifies what and how the datamining task should be performed. This configuration influences the grid which is built.

The datasouce configuration mainly contains the filepath to the datasets to use. Additionally, it is possible to specify the number of batches and the number of epochs to use during the learning phase of the miner. The batches parameter is used to split up datasets into smaller parts which are used sequentially. It is incompatible with SGDRE and thus ignored if the function is selected. The parameter can help when working with very large datasets but can lead to different results with small datasets. The epochs parameter specifies how often the miner should learn on the whole datasets. This parameter is usually used in combination with adaptivity.

"The scorer evaluates the model during training with respect to a metric" [6]. Here only the used metric can be specified, in both our applications this was the mean squared error.

The fitter configuration holds all parameters which determine how the sparsegrid should be constructed and what datamining task should be performed. The task is specified in *type*, in our case this was either density difference estimation or density ratio estimation. The configuration of the grid itself is done in the gridConfig subconfiguration. There the level of the grid, the type of grid and the dimensions of the grid can be set. Adaptivity, meaning refining the grid further in localized zones is also a subconfiguration, which was left empty when not in use but otherwise the number of refinements and the number of points to add in each refinement can be specified here. There are a few additional parameters inside which are used to improve and control the refinements. The regularization subconfiguration is used to control how much "the learning process is impacted by the regularization lambda". We only used the identity matrix regularizations but the lambda value will change often in the evaluation process. At last, the densityEstimation subconfiguration is set but not modified during the evaluation process. For our purpose it just specifies that the conjugate gradient should be used to estimate the density.

2.3 Kernel based methods

At last, we wish to present the kernel based methods used for comparison against the sparse grid methods. Both use advanced estimation algorithms which allow them to be fairly accurate. As they use direct estimation techniques, they have an improved accuracy compared to the naive approaches.

2.3.1 LSDD

This algorithm will be compared to SGDDE. It was developed by Sugiyama [7] and its implementation was taken from the paper's code repository for use in the evaluation.

The approach is to fit a density-difference model $g(x)$ to the true density-difference function $f(x)$ under the squared loss:

$$\underset{\hat{f}}{argmin} \int (g(x) - f(x))^2 dx \tag{2.17}$$

Once $g(x)$ is optimized, we obtain the least-squares density-difference estimator $\hat{f}$ given as

$$\hat{f}(x) = \sum_{l=1}^{n_P+n_Q} \hat{\theta}_l \exp(-\frac{|x - c_l|^2}{2\sigma^2}) \tag{2.18}$$

with $(c_1, ..., c_n) := (x_1, ...x_n)$ the gaussian kernel centers, θ_l a factor and σ the standard deviation.

2.3.2 uLSIF

This algorithm is used to evaluate SGDRE and is compared to it. It was also developed by Sugiyama [2].

Similarly to the sparse grid method, the approach is to minimze a squared loss function.

Unfortuately only matlab code existed in the repository of the project, so the matlab engine for python had to be added to the pipeline. This increases the complexity slows the pipeline down because datasets and parameters need to be converted before and after, but is a workable solution.

3 The evaluation process

To properly evaluate the accuracy of both the density difference estimation for sparse grids (SGDDE) and density ratio estimation for sparse grids (SGDRE) implementations, both methods will first be compared to the analytical solution and then to corresponding kernel based methods. For SGDDE a least-squares density difference estimation (LSDD) algorithm is used and for SGDRE an unconstrained least-squares importance fitting estimation(uLSIF) algorithm is used.

This means that each test will compute and compare two solutions against at least one sparse grid estimation. Some tests will focus on finding the best parameters for the sparse grids and may therefore have more than three solutions. To efficiently run these tests, it is necessary to implement a pipeline. The goal of the pipeline is to simplify and automate most parts of a test, while staying modular in order to run both the density difference and the density ratio evaluations separately. Having multiple steps built as separate functions improves flexibility and simplifies modifications, improvements, bug tracking and therefor the final evaluation step. This also prevents code duplication, as most steps are used for both evaluations.

In practice, a first pipeline was written to generate tests, while a second pipeline uses the tests to analyze and evaluate the new sparse grid methods either visually or numerically. In the following sections both are described in detail as they impact the evaluation of both sparse grid based estimations.

The dataset and evaluation pipeline were programmed in python. Although most of the data handling is done with the numpy module, other modules were used, including scipy for dataset generation, matplotlib for visualizations and of course pysgpp, the python version of SG++ for sparse grids. Additionally, the matlab engine for python was used, to compute the kernel-based method for density ratio estimation.

The code of the pipeline is available here: [8].

3.1 The testset pipeline

The first pipeline is used to create new datasets and configuration files using the scipy.stats module in python. A group of these form a testset. A basic testset consists of two datasets (P and Q), an analytical solution in the form of a mesh grid or a function and lastly a configuration file. Testsets can also include multiple configuration files for more advanced evaluations and parameter comparisons. The pipeline consists of two steps, which are described in subsection 3.1.1 and subsection 3.1.2.

3.1.1 Dataset generators

The structure of the created testsets is dictated by the evaluation procedure. First of all, DDE and DRE methods require two datasets to compute a solution. The primary dataset is generally called "P"-dataset and the secondary dataset the "Q"-dataset. To evaluate the sparse grids based solution, it is necessary to compare its output with the analytical solution as well as with similar functions, in our case kernel based estimations.

The "P"- and "Q"-datasets are created by two dataset generators. A generator uses the scipy.stats module from python, because it contains a lot of different probability distributions and provides both random variate samples (rvs) and probability density functions (pdf) which are both required to assemble a complete dataset. Some of the available distributions that were used are beta, normal, multivariate normal and dirichlet.

The dataset generator takes a distribution, parameters for this distribution and the number of samples as an input. It is called once for each dataset, such that both datasets can have different distributions. Nevertheless, both datasets need to have the same amount of dimensions. A generator will first generate a list of rvs from the given distribution. Then it will save a corresponding pdf to a json-file. In an optional third step, a mesh grid of pdf-values is precomputed and saved for one and two-dimensional datasets, as these will be plotted later on.

Since sparse grids are confined to the n-dimensional hypercube, the rvs list created in the first step either has to be rescaled or generated such that it fits inside the limited space. In the end, generating the dataset properly, without rescaling was chosen. This approach gives greater control over the list and enables us to select parts of a distribution. While scaling is an option it is not used in the final pipeline because both datasets have to be examined and scaled by the same amount before executing the evaluation pipeline and scaled back up afterwards, which makes debugging and comparing results unnecessarily cumbersome. This is especially true when comparing multiple tests with different scales. A drawback of drawing and selecting points inside the hypercube and cutting out others, is that the density values change. The real density, given by a pdf function, are not constrained to the hypercube. The algorithms estimating the samples will however only get points inside of the domain and base the density values on that. While the shapes stay the same, the density values of the analytical solution will not be the same as the estimations done by the sparse-grid-based- or kernel-based-methods. Depending on the portion of the distribution outside of the hypercube, the scaling will differ, meaning that if 99% of a distribution is inside the hypercube, the analytical values and estimations will be very similar, but the more the distribution is spread out, the more the values will differ. Since the kernel and sparse grid methods use the same dataset, they will still be comparable. The same applies to the calculated mse values. They can be compared between each other, but a high mse value will more likely indicate that a larger portion of the datasets should be outside the hypercube, rather than a bad estimation. All of this is only the case for distributions which had to be limited to the hypercube.

To generate the list properly, a redrawing algorithm is used when necessary. It takes a rvs function and the required amount of samples as an input and generates a list of points. For some distributions, like Beta or Dirichlet, redrawing is not necessary since they are already constrained to the hypercube but Normal and multivariate Normal distributions do not have such limits. The redrawing algorithm first generates a list of points of required length with the given rvs function, then eliminates all points which fall outside of the hypercube. If the list of points is too short, the whole list is redrawn with 10% more points. The process is repeated until the required length is reached. If too many points are left, the list is simply trimmed. By redrawing the entire list, a seed or random state can be used. This allows us to recreate the same dataset over and over if required.

In general two datasets are created for each test. Sometimes the same pair is used for multiple tests but mixing datasets from different pairs was avoided for clarity. Pairs can have different parameters and even distributions but always need the same amount of dimensions.

3.1.2 Configuration generator

Once new datasets are saved, a configuration file is generated. As seen in subsection 2.2.4, the SGDDE and SGDRE functions are best executed with a configuration file holding all the relevant parameters. At first they were created manually but since the parameters heavily influence the results, advanced tests like lambda- or adaptivity- optimization became necessary. These tests use multiple similar configuration files so that creating the files automatically is faster, cleaner and less prone to errors.

The configuration file generator takes the path and name for the P and Q datasets as main parameters and saves the output in a JSON file. A lot of relevant parameters can be set and if left out, default parameters are assigned.
The following parameters can be specified:

- fitter type (SGDDE or SGDRE)
- size of a batch
- number of epochs
- grid level
- grid type
- adaptivity configuration (multiple parameters)
- lambda value

If a list of values is specified for a single parameter, the generator automatically creates the Cartesian product of all possible configurations and saves them individually. Additionally, some more parameters can be set, but will not be included in the Cartesian product:

- scorer metric (default: mse)

- density estimation type (default: conjugate gradient)
- regularization type (default: identity)

That way, all configuration files can be quickly developed for a pair of datasets. Similarly, the evaluation pipeline can find and run all automatically created configuration files which belong together and run them sequentially for advanced tests.

3.2 The evaluation pipeline

The evaluation pipeline takes existing testsets as an input and calculates an analytical solution and a kernel based solution and at least one sparse grid solution. It can visualize one and two-dimensional datasets and calculate the mean squared error for any dimension. Usually, a list or grid of points to extract and evaluate the solution is given too, but the pipeline can create its own if none is supplied. All solutions of a test use the same points for an accurate evaluation.

The *fitter type* parameter in the configuration file decides which solution to calculate. The function described by the *fitter type* is run for the sparse grid solution but it also dictates the analytical and kernel based functions to use.

3.2.1 Sparse Grids density estimation

The first stage of the pipeline computes a solution using sparse grids. It takes all provided configuration files and computes individual solutions. Each configuration file is used to build a sparse grid miner, train it and extract the model from it. The model holds information like the coordinates of the grid points, their values and a function used to evaluate any point in the hypercube. After training the model, a list of points is extracted and saved for comparison with the analytical and kernel based solution.

Density Difference Estimation

If the configuration file has "density difference estimation" as *fitter type*, then this path of the pipeline is selected. Using the pysgpp module, a miner is built to estimate the difference in densities of the "P"-dataset and "Q"-dataset: $P - Q$. For more details on this, please refer to subsection 2.2.2. The result is a single sparse grid from which evaluated grid points and possibly a list or grid of points can be extracted and saved for later stages in the pipeline.

Density Ratio Estimation

This branch of the pipeline is fairly similar to the previous one, with the only major difference being the function used: $\frac{P}{Q}$. The result is again a single sparse grid which can be evaluated in any point of the hypercube. Although divisions by 0 can theoretically happen in this step, none were encountered due to the way the density estimations are performed, so no special handling was necessary.

3.2.2 Probability density function

The next step in the evaluation of sparse grids methods is of course to calculate the analytical solution. Using probability density functions or precomputed mesh-grids defined along with the datasets in subsection 3.1.1, the density difference or ratio is calculated. Since the densities are given, the operation doesn't require any data conversion, however divisions by 0 can occur when calculating the ratio and these cases need to be taken care of. The python numpy module sets a division by 0 to 'nan' by default. In visual evaluations this is not a problem, as the plotting function simply doesn't display these values. In the numeric evaluation (subsection 3.2.5) 'nan' values become a problem and are therefore identified and taken care of.

3.2.3 Kernel Based Density Estimation Methods

The last computation step is used to compare the estimation done by a kernel based method to the estimation performed by sparse grids in subsection 3.2.1. There are a multiple different algorithms which can be used, so two newer and accurate algorithms were selected from research papers [2, 7]. Both papers had implementations of their algorithms available online which were used during this section of the pipeline.

Both algorithms use the same P and Q datasets as the sparse grid methods. After the algorithm has generated an estimation, the same evaluation points as for the sparse grid and analytical solutions are extracted and saved.

Least Squares Density Difference estimation

For a kernel based density difference estimation, the least squares density difference algorithm (LSDD) described in [7] was chosen. The algorithm is available online [8] and could be used with small modifications. Parameters for its execution are either specified manually or automatically set by the pipeline to have a similar accuracy as for the sparse grid estimation.

Unconstrained Least-Squares Importance Fitting estimation

As kernel based density ratio estimation method, the unconstrained least-squares importance fitting algorithm (uLSIF) described in [2] is used. Since the original code was written for matlab and not available in python, more extensive changes had to be done than for LSDD. The pipeline uses the matlab engine for python to run the code natively. This means that the datasets and evaluation points need to be converted to the correct matlab format and the output back to a python array. Alternatively, a python implementation of the algorithm was added but isn't used in the final version of the pipeline. Parameters for uLSIF include the number of kernels and folds to use. The number of kernels is automatically set to the same amount as grid points used in SGDRE to properly compare the accuracy with the sparse grid counterpart, but can be manually overridden if needed.

3.2.4 Visualization and plotting

After all three computational steps, the pipeline can be configured to plot the results or to calculate mean squared errors. Studying the results visually helps quickly identifying accurate and inaccurate cases and offers an overview of the datasets. However the visualization pipeline is limited by the amount of dimensions. The output of a density estimation has one more dimension than the input dataset. This means that a three-dimensional dataset will generate a density in the fourth dimension. While plotting functions which can reduce the dimensionality were used on two-dimensional datasets effectively, they are ineffective at displaying three- and four-dimensional dataset in a readable three-dimensional format. Therefore the visual evaluation is only available for one- and two-dimensional datasets.

Both visualization versions aim to represent the original datasets, the analytical solution, the kernel based solution and then all the sparse grid based solutions. They use different plottypes though.

One-dimensional datasets

If the input datasets are one-dimensional, the computed solutions will be two-dimensional which allows plotting without grouping datapoints to reduce dimensionality. This can be leveraged to accurately evaluate solutions and was therefore very helpful in the early stages of evaluation.

The resulting figure consists of at least five plots, more if multiple configuration files where provided at the start. The first two plots are histograms of the P and Q datasets to get a sense for the distributions. Next is a two-dimensional plot of the analytical solution, which shows the exact result the estimations should tend towards. In the case of a density ratio calculation, a division by 0 could have occurred in subsection 3.2.2. In this case the line on the plot is simply interrupted where the result has no data. Next is the kernel based solution and lastly are all the sparse grid solutions in different plots for convenient evaluation. The plots are labeled and all x-axes start at 0 and end at 1, such that they can be compared easily.

Two-dimensional datasets

In case the input datasets are two-dimensional, the computed solutions need to be aggregated in order, because three dimensional plots can not be accurately evaluated when displayed in two-dimensions. Nevertheless, the visual representation proved useful for more complex tests than in one dimension and showed how the algorithms behave with increasing dimensions.

The resulting figure has at least eight plots, as each of the computed solutions now has two plots. The first two plots are once again histograms of the P and Q datasets, but this time they use a colorgradient to display the number of points per sector. This shows that some accuracy is lost, as the colors can't be accurately matched to values visually. Next are two plots for the analytical solution, the first is a contourplot which can show the general topography of the data and add a few values. The second plot is a heatmap, to better represent the general shape of the data, but it doesn't show values, so that both plots are needed for an accurate evaluation. The kernel based and sparse grid based results are displayed in the same way. In order to properly compare the results, all contourplots use the same contourlevels. That way, inaccurate data can be spotted easily, facilitating the evaluation. The plots are necessarily labeled and use the same axes as in section 3.2.4.

3.2.5 Numeric Evaluation: Mean Squared Error

As a last step of the pipeline, the mean squared error (mse) and the root mean squared error of all estimations can be computed. While this method doesn't have the precision of a plot and doesn't reflect localized problems, it can quantify the accuracy of methods and isn't affected by higher dimensions.

The points evaluated in this step are independent of the plotpoints. Since the mse doesn't need an evenly spaced grid of points, it can be used for other purposes, like inspecting the accuracy of edges or the center or specific peaks in the distribution. If no evaluation points are specified, it will generate a grid of 100 points in every direction automatically. In any case, it will use the same points to evaluate the sparse grid results and the kernel results and save the final mse and rmse for both.

Since the analytical solution is used to calculate the error, the density ratio calculation can result in extreme or undefined values. In case the pdf of Q has a 0 value, the computation will either have infinity as a result if the value of P is not 0 in the same point or 'nan' if it is. This is problematic because any calculation containing such a value is automatically extreme or undefined. To get an accurate mse, the squared error is replaced by 0 and ignored for these points. This does not impact the accuracy of the evaluation significantly because the same points are ignored for the sparse grid estimation and the kernel estimation.

4 Evaluation Results

A thorough study of both functions requires a stepwise approach. Consequently each algorithm is examined separately but both are subjected to similar tests. The first tests are in one dimension and allow us to understand how the sparse grid methods behave and compare to the other solutions by examining them visually. Following the very basic tests, the dimensionality will increase to at least two dimensions. The objective in this phase is to evaluate commonly used parameters and their influence on the results. Additionally, first manual parameter optimizations can be carried out by roughly estimating the shapes and values. Next, more specific tests are carried out with "unfriendly" datasets in order to better define weaknesses and strengths of the functions. Particular cases like the dataset sizes are examined. Lastly, datasets in higher dimensions are used and analyzed numerically by observing the mse and comparing the score to the kernel methods. The kernel methods have fewer parameters, but they can influence results nonetheless. By default 100 kernels and 5 folds are used, but when tests are run with different values, it will be specified.

The tests are run by a pipeline explained extensively in chapter 3. The code used to generate and plot the figures is available in [8]. A seed was used during the dataset generation of each test. By doing this, the code can be rerun and will always produce the same sets and results. If the seeds are changed or removed, the datasets will be different and the results they influence will therefore change too.

4.1 Visual Evaluation

4.1.1 Presentation

The first experiments are intentionally basic. They are used to present the pipeline output and explain which values and behaviors to look out for once complexity increases. Since the evaluation pipeline generates the same types of plots for DDE and DRE, only DDE output will be used to present the figure's structure.

Beta distribution (*TestDDE1*):

The first experience will be used to present the one dimensional result structure. Its testset consists of two beta distributions. They are good candidates because they are continuous and bound in $[0-1]$. Both consist of 500 samples in a single dimension, which is a large number in comparison to most real-world applications. By choosing a large set, overfitting becomes less likely and the algorithms have better odds of correctly estimating the density difference. The parameters chosen when creating the datasets result in slightly off-center peaks for both the P and Q dataset. The sparse grid estimation is configured with all parameters set to default.

Running *TestDDE1* yields 6 plots in a single figure as shown in Figure 4.1. The first two on the left will always be histograms of the P and Q datasets no matter the test. They mimic the shape of the distributions and allow the reader to estimate the densities of the datasets. The plot titles include the names of the datasets. In the second column on top, the analytical solution is shown. It is the result of the difference of the pdfs associated with P and Q respectively. Below it is the kernel based solution. In the case of density difference estimation, this will always be the result of LSDD (subsection 2.3.1) and when evaluating a density ratio estimation, the result of uLSIF (subsection 2.3.2) instead. The centerpoint $(0.5, 0)$ in black is added to every plot as it improves the ability to compare the plots. In the upper right corner of the plot are displayed the mse and rmse values. They help understand how precise and reliable the estimation is.

They also allow us to compare the estimation numerically with the sparse grid solution, shown in the last two plots. Advanced tests will have more sparse grid solutions for different configurations, to the right of the current plots. In case someone would want to inspect the configuration files, the plottitles correspond to the configuration name. Each sparse grid plot shows the density difference estimation in blue and the evaluated grid points in red. Additionally the mse and rmse values are displayed in the upper right corner. For comfort reasons, the configuration parameters of each sparse grid plot are displayed in the upper left corner of the corresponding plots.

Examining the upper sparse grid plot, this simple experience already shows some of the characteristics of sparse grids. For one, the density difference line appears piecewise linear instead of a smooth curve. This stems from the linear basis functions used to estimate the density difference at the chosen grid points. In contrast, LSDD uses rounded kernels, which fit well in this example. The mean squared error is calculated using the points of P and Q. Here we get very low error values for both the LSDD solution and the sparse grid solution despite their obvious visual difference. This proves that both a visual and numeric evaluation are necessary to study the accuracy of the algorithms in more advanced tests.

Two-dimensional normal distribution (*TestDDE2*):

As a second experience, two-dimensional normal distributions are examined. P and Q are created by specifying a mean and a covariance matrix. The scipy function used here allows an array or a scalar to be specified instead and will transform it by multiplying it by the identity matrix. This time 1000 samples are used per dataset. Once again, the sparse grids are configured to use the default values.

TestDDE2 is shown in Figure 4.2. Since two-dimensional datasets output their density estimations as a third dimension but three dimensional plots decrease readability, plots which reduce the dimensionality are used. On top of improving readability, they enable visual comparisons. The two histograms for P and Q on the left side for example, use

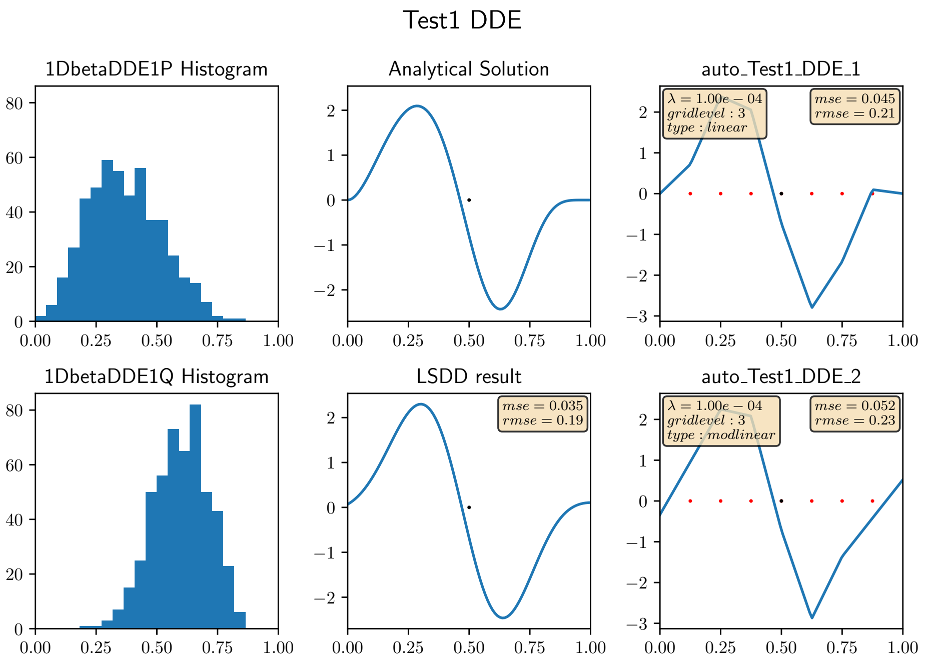

Figure 4.1: TestDDE1 shows basic output of the one dimensional visual pipeline.

colors to indicate the number of points in each bin. All the other representations are different when compared to one-dimensional figures. Each result now has two plots, the upper one is a contour plot and the lower one a heatmap. The contourlevels are chosen before plotting, so that every plot uses the same ones. That way the results can be compared more accurately with contourplot. To complement the values and thresholds given by the contourplots, heatmaps display more data and accurately reflect the shape of results, however, no values can be read from them. Once again, the centerpoint $(0.5, 0.5)$ is added in black to detect slight differences in the resutls. Thanks to the point, we can detect a different boundary of the purple oval in the heatmaps of the analytical, kernel based and sparse grid based results. This time, the entire second column is dedicated to the analytical solution and the third one to the kernel based method. The sparse grid results start in column 4 and are added to the left until a total width of 5 plots. If more plots need to be shown, a second row (Each row has 2 levels, on for the contourplot and the other for the heatmap) is created and the plots added there. Once again, mse and rmse values are displayed in the upper right corner of the contourplots and the sparse grid configuration in the upper left corner.

By examining the plots, some characteristics become apparent again just like in the first test and some appear here for the first time. Despite using less points than the LSDD algorithm and default parameters, SGDDE seems fairly accurate. Both the shapes of the peak (zone above 3.00 in contourplot) and the valley (zone below -3.00) follow the shape of the analytical solution, despite showing artifacts of the linear basis functions. The sparse grids heatmap reveals a crosslike shape in the valley stemming from linear functions in two dimensions. The peak zone is also narrower than the analytical solution, but the length of it appears accurate. In contrast, the LSDD result appears to have a very slightly larger width but the shape is accurate overall. This discrepancy in precision is reflected in the mse values as LSDDs value is barely above half of SGDDEs value.

On the contourplot of LSDD a characteristic of kernel algorithm starts to become visible: Due to the kernel estimation, the shape has some slight waves which will become more apparent in other experiments later on.

4.2 Density Difference Estimation

We begin with an evaluation of the SGDDE algorithm. An in-depth explanation about the algorithm can be found in subsection 2.2.2. In this section we examine the algorithms behavior against different datasets and parameters and compare its accuracy to the analytical and kernel based solutions with a limited amount of points.

4.2.1 Parameter testing

Since basic tests were run in subsection 4.1.1, those will not be repeated. We can now dive into the parameters used by the SGDDE algorithm and explore their effect on results. The goal is creating a ranking of the parameters by influence on the results and to find a configuration that calculates accurate results in any dimension. Lastly, we will vary the number of samples in each dataset to determine if there is a minimum amount of points required for an accurate result. This step could prove useful in case SGDDE is used for timeseries segmentation or other applications with limited datapoints in the future.

Influence of parameters

During the course of the evaluation, a combination of 7 parameters where modified most of the time as described in subsection 3.1.2. Other parameters may change occasionally for specific tests, but are not analyzed with as much depth and thoroughness. Two parameters which definitely have a big influence on results are the gridtype and the gridlevel. The gridtype can either be linear or modlinear in our case and influences the behavior towards the boundaries. The gridlevel will have an influence on the precision overall, but in higher dimensions every level generates a lot of new points, increasing the computing time. It is therefore beneficial to keep a low or moderate level and then improve accuracy in key zones by using adaptivity parameters.

Gridtype and gridlevel (*TestDDE3.1*) The next experience will use the same dataset to compare different combinations of gridlevel and gridtype. For now, we do not use adaptivity and select a low λ value to reduce smoothing and thus obtain the raw sparse grid output.

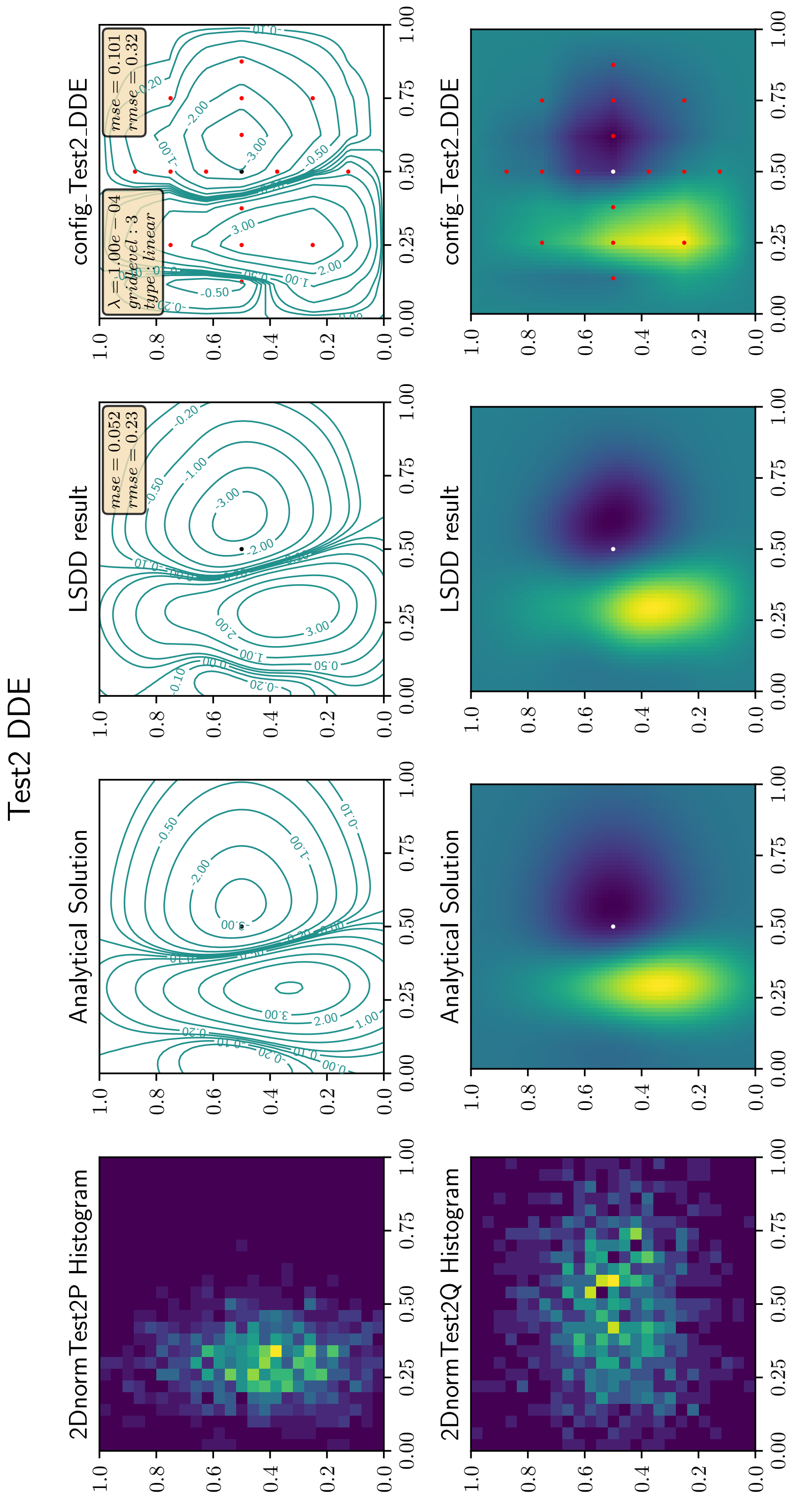

Figure 4.2: TestDDE2 shows basic functionalities of the two dimensional visual pipeline.

The result of *TestDDE3.1* in Figure 4.3 displays the modlinear estimations in the top row and the linear estimations in the bottom row. Each column represents a different gridlevel. Because dataset Ps density increases towards the border and Qs decreases, the analytical solution diverges from 0 at both ends and oddly resembles the indian peninsula.

The LSDD and linear SGDDE solutions both tend to return to the origin at the border regardless of the number of gridpoints. This results in inaccurate estimations at the border. Returning briefly to textitTestDDE1, the opposite effect is visible: the modlinear SGDDE solution doesn't converge as well as the linear at the border when the density tends toward the origin. This indicates that a linear gridtype should be chosen if the density difference at the borders tends towards 0 and a modlinear gridtype if not.

The grid level is the other important parameter examined in the experience. The grid of level 2 was intentionally chosen to show how the solution is constructed step by step and improves with each level, but is too low for most applications. Interestingly, the modlinear solution of level 2 has a better mse than the linear solution of level 4, once again highlighting the importance of choosing the right gridtype.

By observing the modlinear solutions, a clear progression is visible: the second plot adds details to the first line and the third one adds smaller details to the second one. However, the third one is noticeably different from the analytical solution and the mse confirms this, since the solution obtained with gridlevel 3 has a lower value than the level 4 mse. This phenomenon is called "overfitting". It is the reason why choosing good parameters is essential.

Lambda (*TestDDE3.2*) A parameter used to balance overfitting and smoothening is λ. In Figure 4.3 a very small value was intentionally chosen to see the effects of gridtype and gridlevel. To improve the estimation a larger value could be chosen. *TestDDE3.2* in Figure 4.4 explores the relation between the gridlevel and lambda. The experiment plots 3 grids of level 3 and 3 of level 4 respectively. Three lambda values were strategically selected for demonstation purposes and are shown here:

- $\lambda = 1$ is too large and thus the data is smoothed too much. The curves for both gridlevels are way too flat and don't have enough relief as can be seen in the respective contourplots.
- $\lambda = 3e^{-2}$ is very close to the optimal values for both grids. The data is similar to the analytical and LSDD solution. Both the heatmaps and contourplots are very similar to the analytical solution. To improve the result further, a modlinear gridtype could be used, since the solution shows the yellow peaks close to the borders but this is not the purpose of the experiment.
- $\lambda = 1e^{-5}$ is too small, resulting in a very coarse grid with almost no smoothening. This is called over-fitting. In practice, the estimation algorithm tries to follow too closely the density of the random variate sample and will deviate quickly from the actual solution when the sample has gaps or peaks. The problem becomes worse with smaller samples, as they can't fill every gap with points.

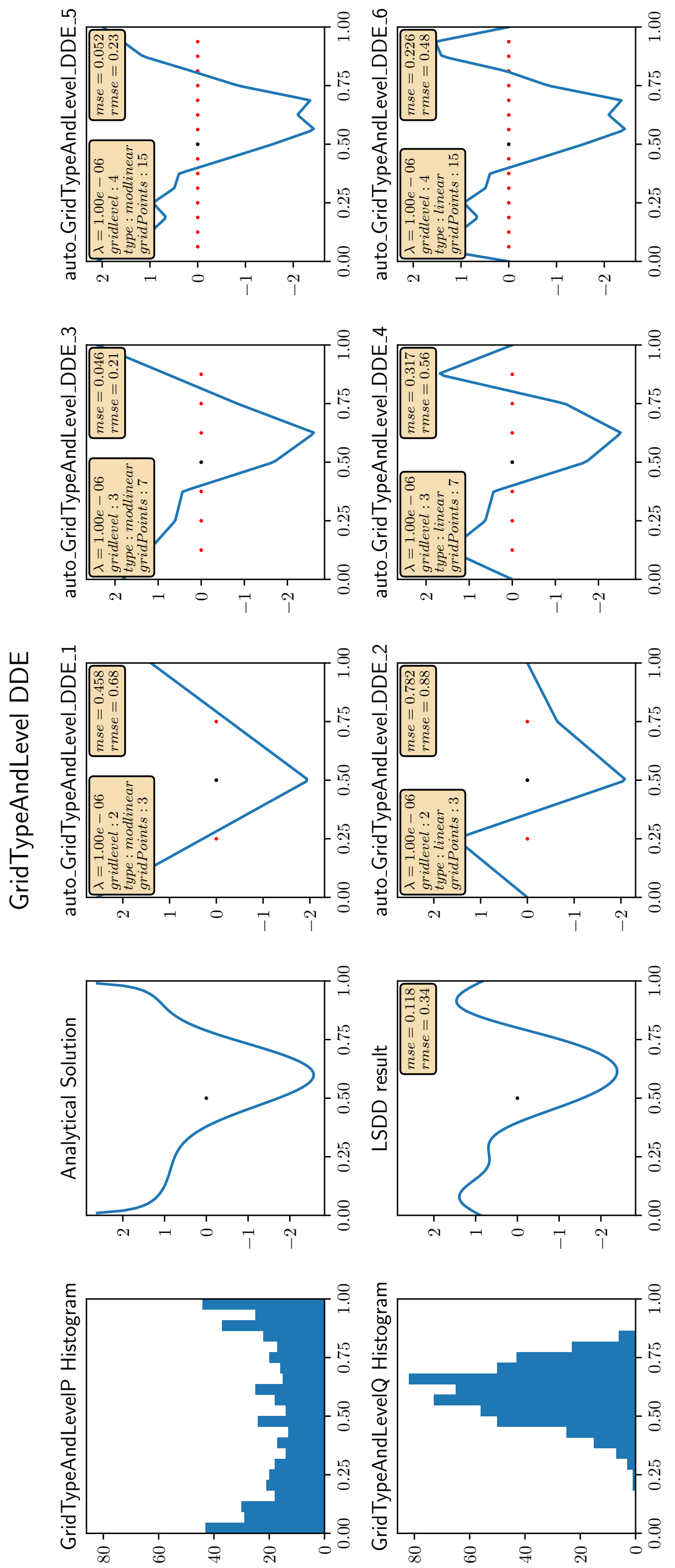

Figure 4.3: TestDDE3.1 compares linear and modlinear gridtypes as well as different gridlevels for two beta distributions. The edgebehavior is worth studying too.

The sparse grid mse is actually better than the kernel solution for both gridlevels in the experience, showing the strength of sparse grid density difference estimation when using the correct parameters.

Since the mse is calculated every time the SGDDE algorithm is run, we can plot the mse of the estimation, lambda values and gridlevels. In order to generate Figure 4.5, the same datasets from *TestDDE3.2* were used with 50 different lambda values. The result shows the evolution of the mse. When the gridlevel increases, the optimal lambda shifts slightly towards bigger values but the most interesting effect are the changes to the mse curve. With increasing gridlevel, choosing a small lambda will lead to overfitting slightly earlier and much faster since the curve is steeper near 0. On the contrary, choosing a lambda which is slightly too big, results in less estimation losses with increasing the gridlevel. Figure 4.5 shows that increasing the gridlevel will increase slightly the estimation quality, but with diminishing returns. A grid of level 4 with a well chosen lambda value can significantly reduce compute time with larger datasets in higher dimensions without loosing a lot of precision when compared to estimations with higher gridlevels. A similar curve with a logarithmic scale instead of a linear one is shown in Figure 4.11, however it uses another dataset and another algorithm.

This example is in no way a proof or a proper way of optimizing the estimation, but it visualizes overfitting and highlights the importance of correctly choosing the lambda value as well as the interaction between gridlevel and lambda.

Adaptivity (*TestDDE3.3*) The parameters studied in the next experiment are used to configure the refinement step. After a sparse grid is generated, regardless of the gridlevel, some points of interest might still be poorly estimated, especially when they are not in the center of the hypercube. To help estimate densities in these zones, the SG++ pipeline uses refinements. As explained in subsection 2.2.4, refinements have multiple parameters which influence the final result. The parameters specify when and how often the refinement function is called in the creation process of the grid, how many refinement points are added, how the pipeline should choose points to refine and how many points should be removed by coarsening the grid.

This experiment focuses on the parameters which influence the sparse grid density difference estimation the most. Those are the number of refinements and the number of points to refine in each refinement. To use refinements, the number of epochs is increased, since the grid needs to be reevaluated after being refined. Thus a refinement step is calculated between two epochs until the maximum number of refinements is reached. The result is then used to improve the estimation in the next epoch. At the end of each refinement, points with a very low influence are removed by the coarsening function to reduce computation complexity.

The SGDDE algorithm can also use batching to split large datasets into more manageable sizes. Since small sample numbers are used in the experiment and to stay consistent with previous experiments batching will not be used here.

The plots using refinements can be identified by looking at the text information, as

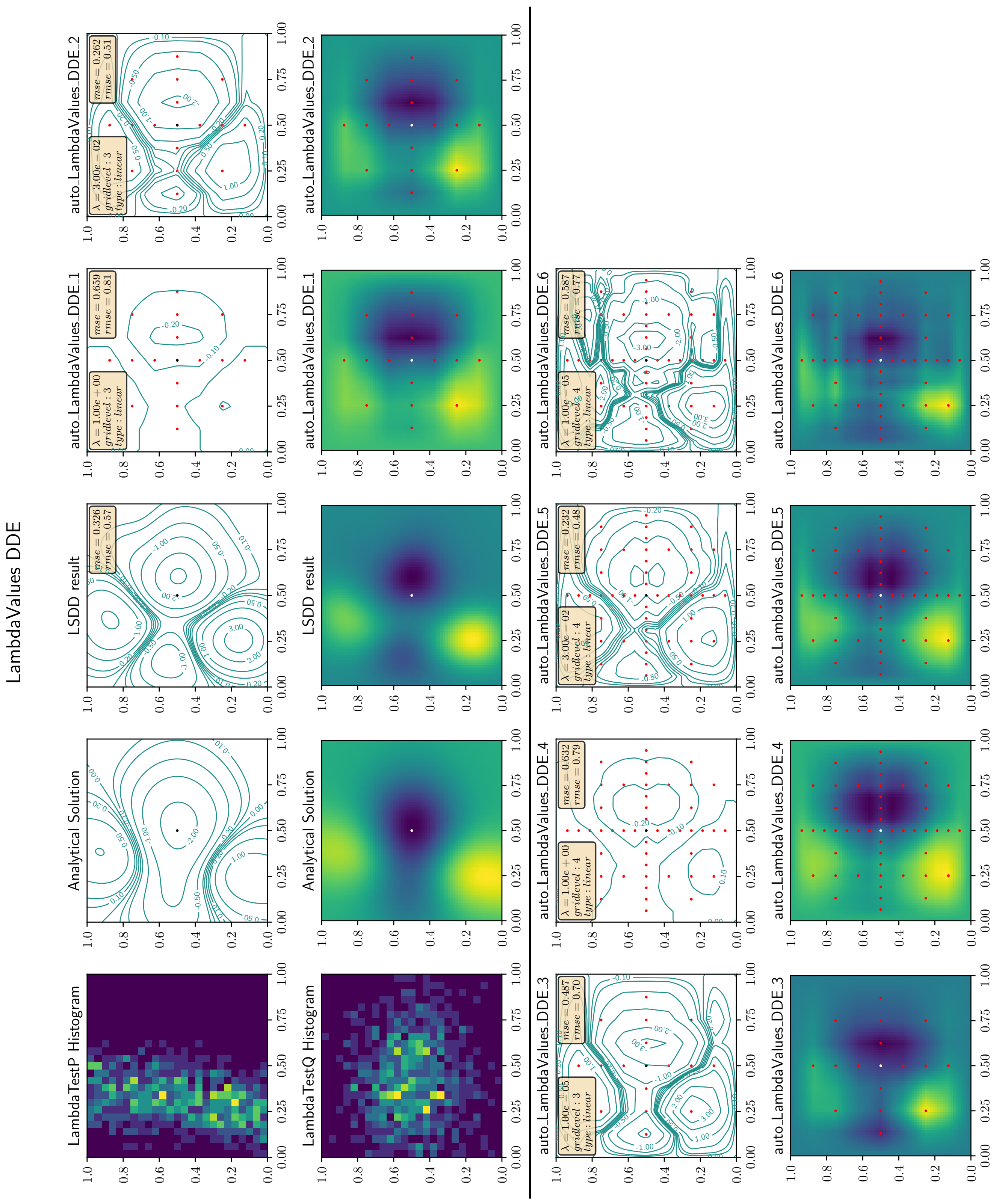

Figure 4.4: TestDDE3.2 compares different lambda values and shows its effects graphically and numerically.

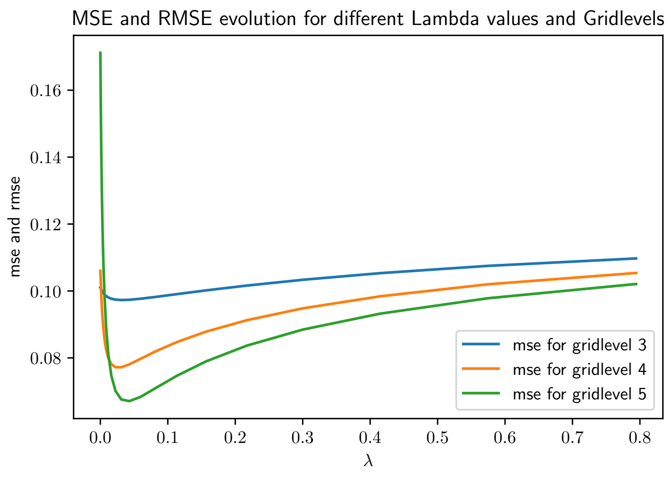

Figure 4.5: The mse and rmse are minimal for $\lambda_3 = 2.94e^{-2}$, $\lambda_4 = 4.04e^{-2}$ and $\lambda_5 = 5.53e^{-2}$ in *TestDDE3.2* (linear scale)

the values for the number of refinements ("num.ref.") and the number of points added in each refinement ("ref.points.") are displayed.

First, the one-dimensional proof of concept in Figure 4.6 illustrates the power of refinements: It contains SGDDE plots, all of which use the same except for the number of gridpoints. The first one has a grid of level 3 and the last one a grid of level 4. The other two plots start at a level 3, but are refined to different degrees. Comparing the mse values, it becomes apparent, that in this case, a single refinement step with two refined points can immensely improve the final result. By refining just two points, the mse value jumped from 2.096 down to 0.309, almost matching the mse of the level 4 grid with less gridpoints. Doing a second refinement step (plot 3) shows the potential of sparse grid refinement. While the grid has the same amount of points as a regular level 4 sparse grid, its mse is just above a third of it.

Further testing has shown that placing points strategically where sharp slopes occur, thus refining the grid further than a full grid would in these zones, improves accuracy of the estimation, but when the estimation doesn't have strong slopes, the accuracy did stay level and even decreased slightly in some cases. The accuracy also dropped when too many points where added

A two-dimensional analysis is made with two multivariate normal distributions specifically chosen for their steep slopes: Test3.3b. In Figure 4.7, different settings are compared to grids of level 4 and 5 without refinements. The refinement settings are chosen so that each sparse grid starts at a level 4 and refines roughly the same amount of points, but in a different order.

The first plot with refinements will run 10 refinements and in each refinement only refine 2 points. This means, that this grid can go very deeply into specific zones, but since no depthlimit was set, this configuration can lead to a single part of the grid being refined, while the rest is ignored. The last grid using refinements is the contrary, it only

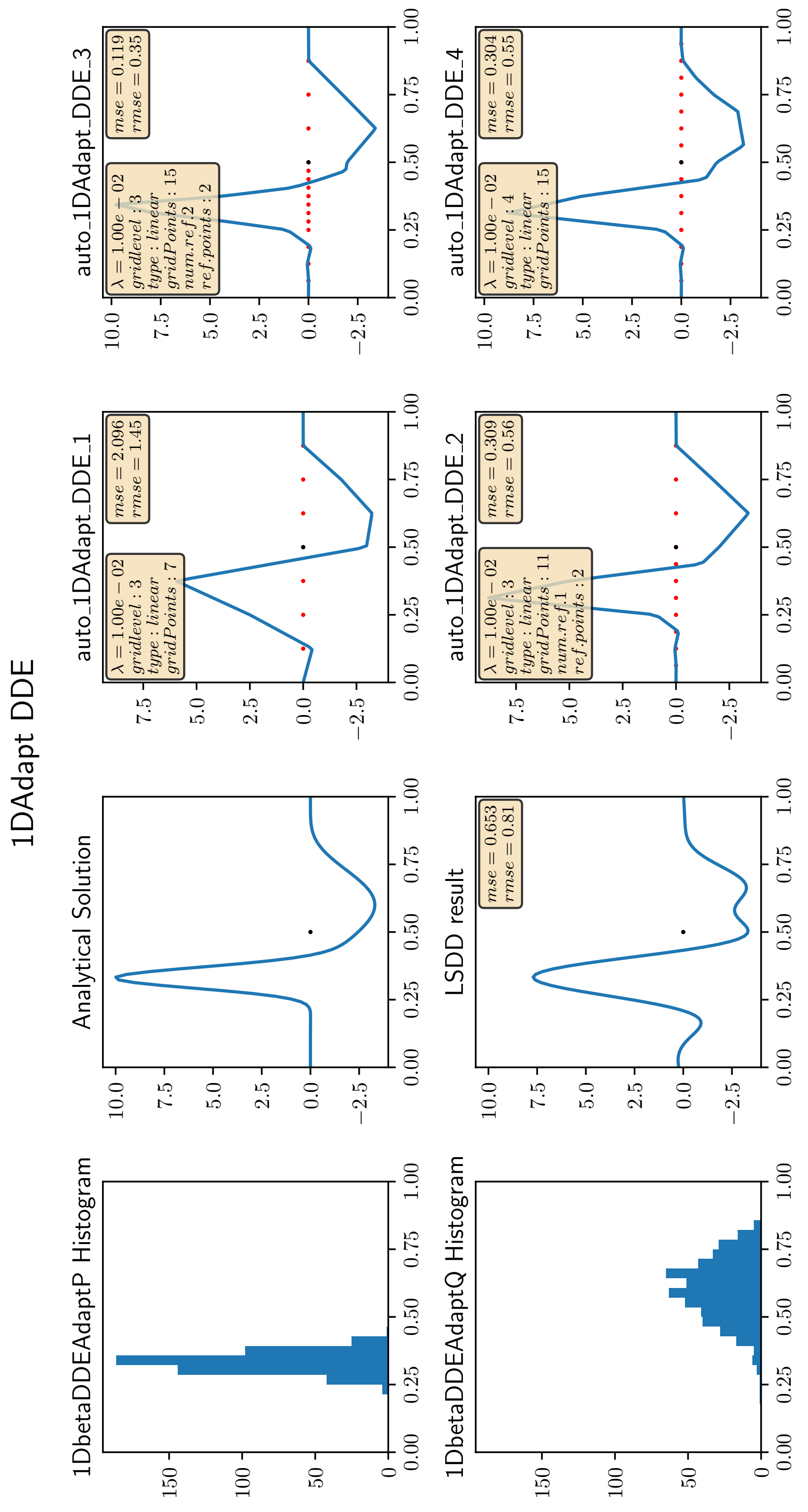

Figure 4.6: TestDDE3.3a demonstrates how adaptivity can match and surpass increasing the gridsize in some cases.

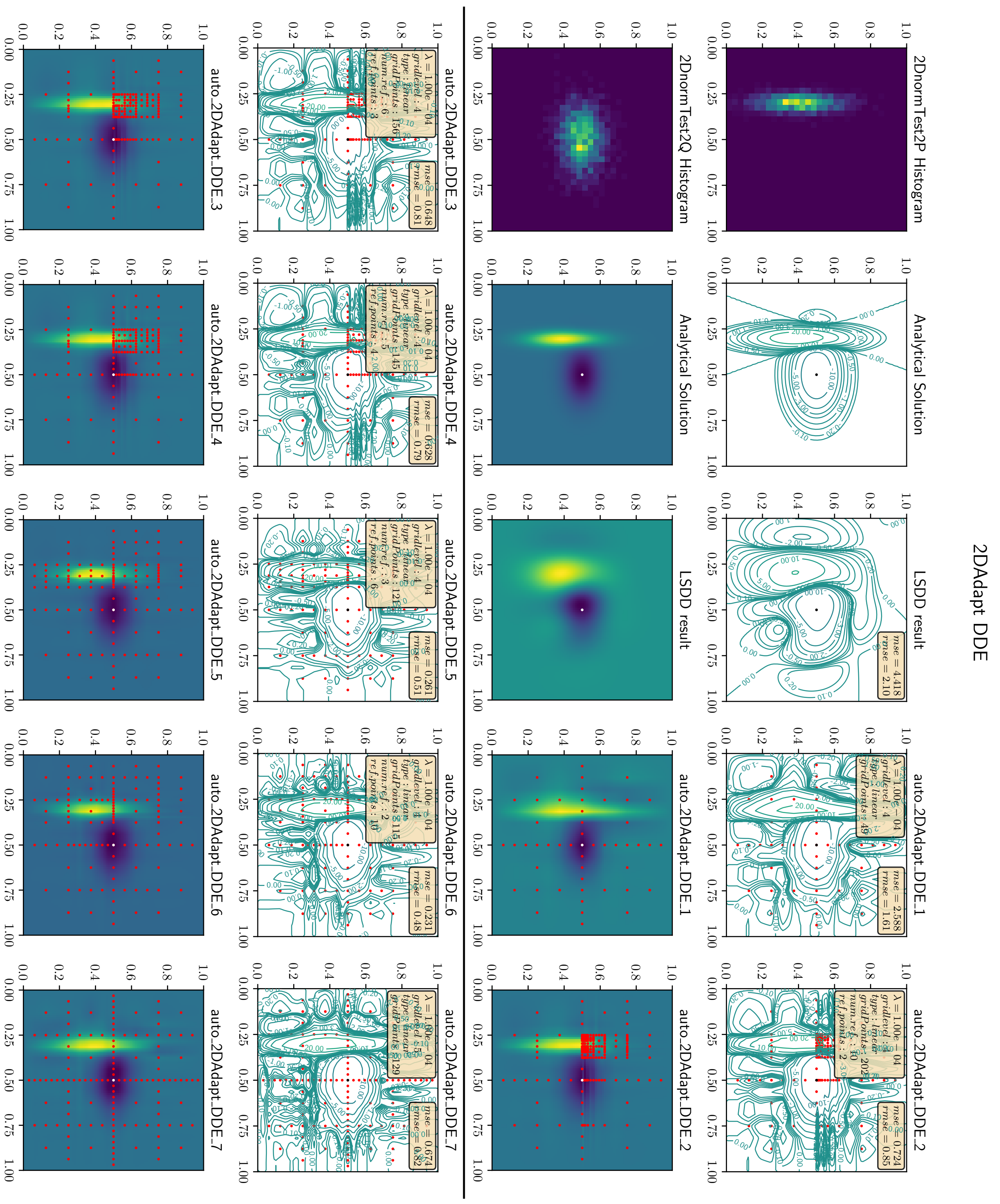

Figure 4.7: TestDDE3.3b compares results using a large number of refinements, with a low number of points to a low number of refinements with a large number of points.

uses 2 refinements, but each of them can refine 10 points. In this case, the sparse grid will be shallower, meaning the sparse grid might not have enough points to estimate peaks and valleys of the density difference estimation accurately, however the rest of the hypercube get's more gridpoints, improving the density difference estimation overall. In this example, the shallow refinement seems like the best configuration, as its mse is lower than the other refined grids and also lower than the level 5 sparse grid without adaptivity.

Choosing the right parameters to configure the refinement step is very important. Depending on the shapes of the used datasets, a balance has to be found between going in depth and improving the grid overall. Doing the same analysis on two dirichlet distributions offered the best mse for a balanced configuration of 5 refinements with 4 points each.

Now that all the useful parameters have been explored and analyzed individually, they can be used together to create powerful, computationally lightweight estimations. There are however two aspects we haven't looked at yet, one being the size of the samples used in the estimations and the other being higher dimensional datasets.

4.2.2 Varying sample sizes

For the next experiment, the existing pipeline is adjusted to use different datasets in each step instead of different configuration files. The same configuration parameters are set and used for each of the tests. The goal is to see if larger or smaller datasets (from the same distribution) will lead to significantly different results. It is expected that a very small sample size will have worse results than a large sample, because the algorithm doesn't have enough data for an accurate density estimation, but how far can an estimation improve by adding samples remains to be tested. The experiment will use the same dataset size for P and Q.

In Figure 4.9 different sizes are represented. The plot is different from the previous plots, because now the heatmaps are replaced by the LSDD estimations. So for each dataset size, there is a SGDDE and a LSDD result. This enables us to compare how both algorithms behave for different dataset sizes and how their accuracy evolves.

The sparse grid is configured with a linear grid of level 3 and a smoothening value of $\lambda = 1e^{-2}$. The size of the datasets is indicated in each graph along the mse and rmse values. The first dataset has 50 samples of each distribution. This is clearly a very low number of samples, but it is a good starting point, because some applications like time series segmentation could use this quantity of samples even in higher dimensions. The LSDD estimation for this few points is very wavy, a characteristic of kernel methods. Once the amount of samples increases, more kernels and better estimation help reduce the waviness as can be seen in the other plots.

When the number of samples in a dataset is low, each sample has more influence over the final estimation. This would be more visible in the first sparse grid plot if a smaller lambda and thus less smoothening would have been done. Due to the choice of lambda,

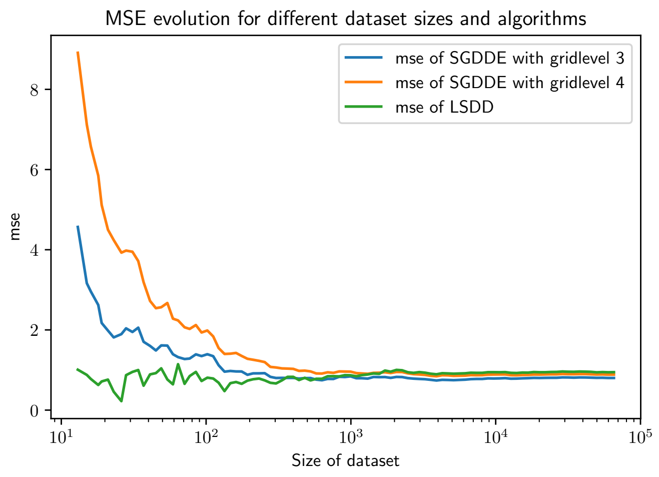

Figure 4.8: LSDD doesn't require a large dataset for accurate results but doesn't improve with more samples, SGDDE improves until about 100 samples before the mse flattens off

an accurate estimation is possible with sparse grids for such a low number of points. Further experiments showed, that a promising shape could be expected once 20 points were used in each dataset, however the mse was still comparatively very high. With only 20 points, jumps in density start to become visible and can be identified, but the grid is not accurate enough to evaluate specific zones or values. With the same amount of points, the kernel solution is very wavy and can therefor not be used except for jump detection. Being able to discern at least density jumps with such a small amount of points could prove useful in a number of applications. Please keep in mind, that in higher dimensions, more points might be needed to obtain a similar result.

When the number of points in each dataset increases, the proper shape begins to appear in both the LSDD and SGDDE plots. The curves in LSDD straigthen out and misplaced isolines are rectified for both algorithms. When examining the mse values, a clear progression can be seen. Obviously, a low number of points will result in a higher mse value, but just by looking at the few numbers gathered from the plots, we can already see a trend for this density difference estimation. The mse appears to quickly drop at the start until it reaches its asymptotic limit. LSDD appears to be more accurate than SGDDE with a lower number of points here, but adding more points improves its score a lot less, such that SGDDE appears to be the better algorithm when having access to large datasets.

To improve the analysis, Figure 4.9 displays the mse values for a hundred different dataset sizes for LSDD and for SGDDE with gridsizes 3 and 4. No refinements were used and lambda was fixed at $\lambda = 1e{-}3$. The figure clearly shows that increasing the gridlevel when a very small number of points is used will result in a less accurate estimation, but the difference quickly dissipates once over 200 samples are used in each dataset. Examing the LSDD estimation's mse, jumps in the curve are observed with small datasets. This is probably caused by the position of points and their influence on the kernel positions. Once the dataset size is above a hundred, values stabilize and all three curves converge to very similar values.

As we saw in this analysis, having a small number of samples will have an impact on the estimations accuracy. Once datasets become large enough however, choosing the right algorithm and configuring it properly will influence the estimation more than the number of samples.

4.3 Density Ratio Estimation

The SGDRE is the other algorithm we want to examine. It is arguably the more interesting and challenging one. The detailed explanation of density ratio estimation, its applications and how the sparse grid implementation works can be found in subsection 2.2.3. In this section we want to start with basic experiments and gradually work towards higher dimensions and using the same parameters as before effectively. The algorithm will once again be compared against the analytical solution and uLSIF, a kernel based density ratio estimation algorithm. First the main parameters will be examined, similarly to SGDDE. Then the experiments will focus on varying the size of samples and high dimensional datasets.

With SGDRE and other ratio functions, particular care has to be taken when choosing the Q dataset, as very low densities in it, will lead to high values in the result. In case the algorithm calculating the mse finds a point where either the analytical or the estimative solution is exactly 0, the point is ignored in order to preserve an accurate mse value.

4.3.1 Parameter testing

Let's reexamine some of the parameters we already examined for SGDDE. Although they haven't changed, the current algorithm behaves differently and other values might therefore produce better estimations. Because of the possibility of dividing by 0 or at least obtaining very high values, it is necessary to find and examine special cases where a very small parameter modification can lead to significantly different results. In general, results became inaccurate and in rare cases unusable for analysis when the density of the Q dataset tends towards 0 and Ps density doesn't because in these cases the mse will quickly jump to very high values. Although plotting and an analysis of the results don't provide good results, this behavior has advantages in some applications. Time series segmentation effectively uses these outlier-values to detect a possible segmentation for example.

Gridtype and gridlevel (*TestDRE1.1*) Let's start with one-dimensional data, since there hasn't been an introduction to Density ratio estimations yet. *TestDRE1* in Figure 4.10 takes two beta distributions and computes their ratio or ratio estimation. This

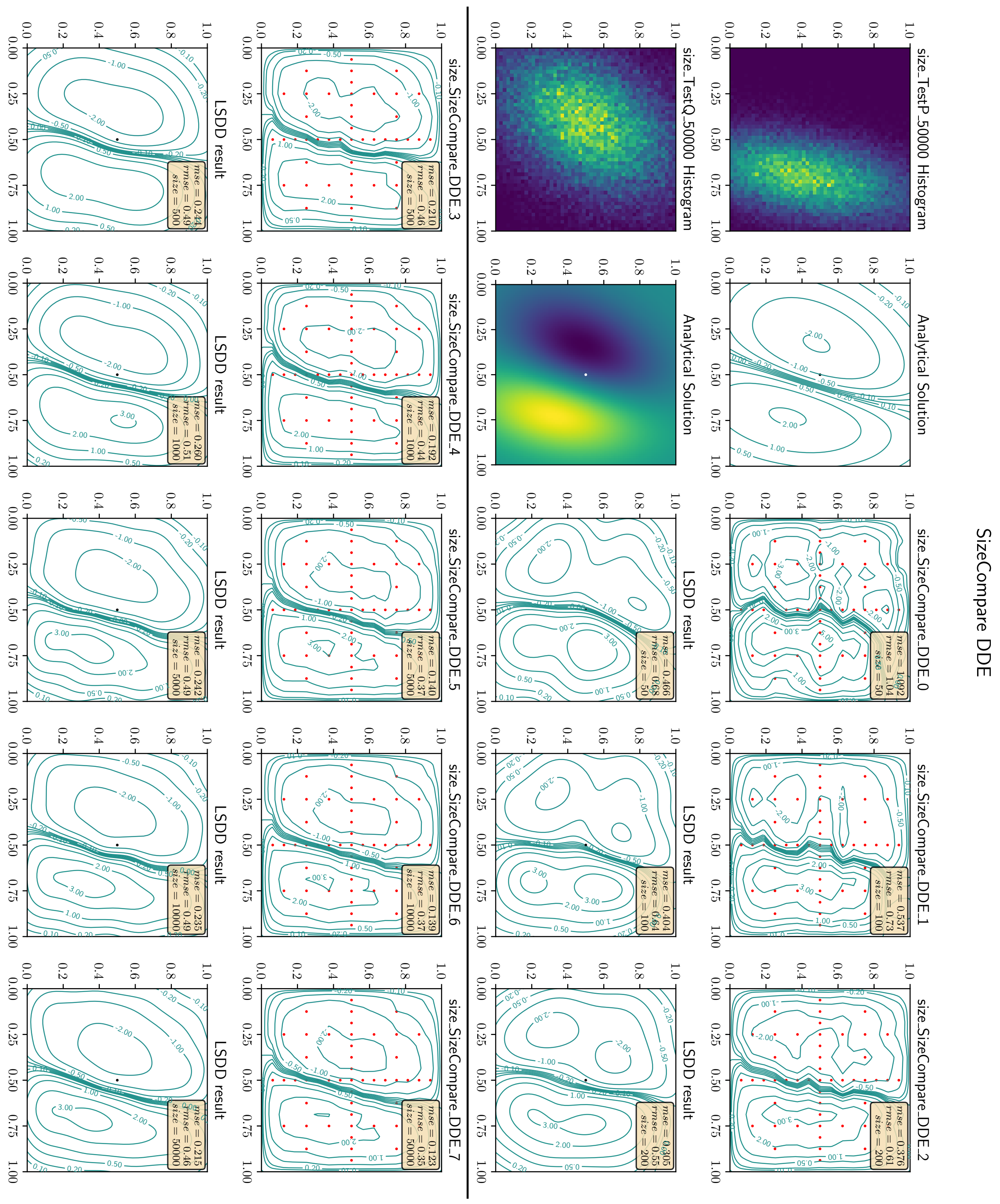

Figure 4.9: In this experiment the same configuration file is used for each plot and the same distribution is used to create the datasets, but the number of samples used for each estimation is different

test consists of 500 samples in P and 600 samples in Q. As before, it displays the analytical solution and the kernel solution to the right of the dataset histograms and all computed sparse grid configurations to the right of that. The chosen lambda ($\lambda = 1e^{-1}$) is slightly above the optimal value, such that it doesn't cause overfitting and such that the estimation still has a rather accurate result.

Three gridlevels are compared for both the linear and modified linear solutions. While the analytical solution has a form very close to a parabola with its peak almost perfectly in the center, both the uLSIF and SGDRE solutions are influenced by the datasets and all of them show a skew towards the left.

With each gridlevel, the mse is improving, but visually they are very similar. Since the solution goes to the origin at the borders, the modlinear solution is expected to have problems, but it improves with each added level. The linear estimations of level 2 and 3 appear extremely similar, hinting at good other parameters. It also suggests the reliability of SGDRE algorithms since the influence of a single parameter doesn't change the result.

The gridlevel 4 solutions have a jagged part at the highest values and are shifted to the right. This might be caused by overfitting, but changing the lambda value doesn't improve the result. Creating larger datasets can mitigate the jaggedness, but it wasn't possible to completely remove it. Since those are not the causes, it is likely a result of the random variate samples and the seeds used to create them. The histograms at least support the hypotheses, because the P dataset has a drop at about 0.6, where the density ratio estimation also drops. Changing the seeds will of course significantly alter the results.

In the development phase of the datasets of *TestDRE1.2*, an interesting but problematic discovery concerning the gridlevels was made: when choosing low gridlevels, subtle details might be lost completely. In Figure 4.13 the correct solution resembles a leaning vertical bar and uLSIF is able to accurately represent this slope. A sparse grid of level 3 without refinement however, doesn't have enough points to accurately represent it. While the contourplot hints at a slight inclination, the heatmap completely ignores the slope since the bar is completely straight. For comparison, a grid of level 4 was added. This one is able to capture the slope but nevertheless keeps the characteristic straightness of sparse grids. What is meant is, instead of having one small bar, the heatmap displays the slope with 3 smaller vertical bars. This is actually similar to how pixelated images reduce slopes to steps. Choosing a lambda value close to the optimum can reduce the effect. If "sloped" data is used, it is therefore recommended to increase the gridlevel or refine the grid with adaptivity.

Lambda (*TestDRE1.2*) The next experiment uses a two dimensional test set to examine if the lambda parameter in SGDRE behave in a similar way as during density difference estimation. A modlinear grid is used, as the edge of the analytical solution doesn't go towards 0 for interesting values. At first glance, Figure 4.12 suggests that lambda behaves similarly. The sparse grid is once again flat and inaccurate when a large lambda

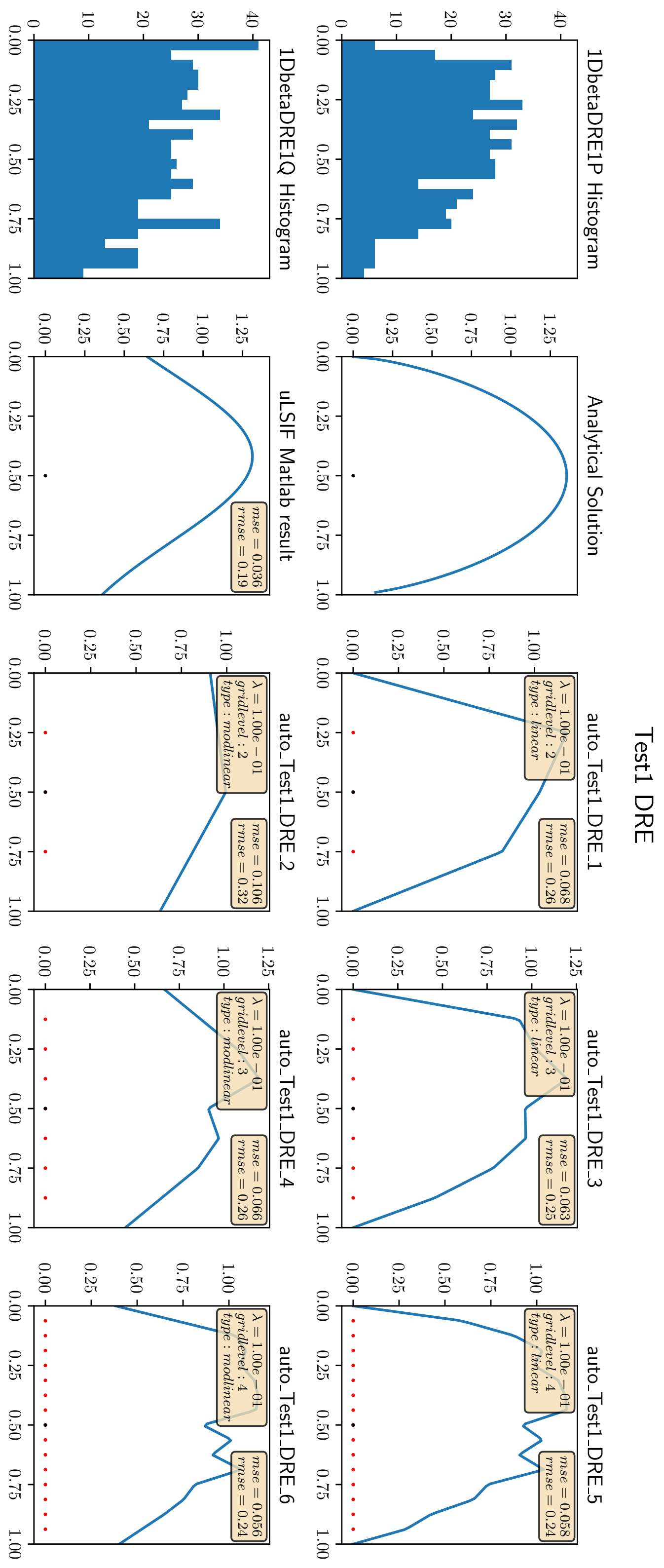

Figure 4.10: TestDRE1.1 compares gridtypes and gridlevels for density ratio estimation algorithms.

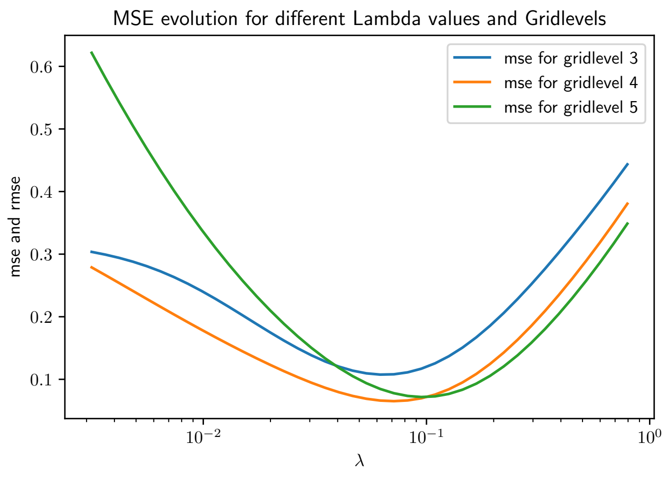

Figure 4.11: The mse are minimal for $\lambda_3 = 6.57e^{-2}$, $\lambda_4 = 6.95e^{-2}$ and $\lambda_5 = 9.68e^{-2}$ in *TestDRE1.2* (logarithmic scale)

value is chosen and gradually improves up to a threshold. After the threshold, the sparse grid algorithm overfits.

Visually the heatmaps don't display the overfitting in this instance, but the contourplots do: The limit seems to be around $\lambda = 5.00e^{-2}$ and definitely above $\lambda = 1.00e^{-2}$ because with the second lambda, the contours start to get very unstable. With even smaller lambda values, contourlines form around the grid points (especially visible in the lower right corner of the plots). The mse values confirm the overfitting since they increase rapidly below the previously mentioned lambda. By plotting the mses for different lambdas (Figure 4.11) we can in fact determine that the optimal lambda is around $\lambda = 6.95e^{-2}$.

The graph uses a logarithmic scale to better display the evolution of the mse. If a linear scale was used, the plot would be similar in shape to Figure 4.5. Here we see that choosing logarithmic steps to optimize lambda is the correct approach, because linear steps will very quickly lead to overfitting the data. The graph also shows that a similar and lower mse can actually be achieved with a grid of level 4 than level 5. For higher dimensional datasets this actually means that a good results can be achieved without long runtimes.

The curve displayed in Figure 4.11 changes when other distributions, samples or parameters are used. Running the same test on normal distributions often resulted in a flat line for very small lambdas and an increase of the mse value once lambda was larger than a threshold specific to the dataset.

Adaptivity (*TestDRE1.3*) The last parameters we are looking at, are once again part of the refinement configuration. All except for the last sparse grid solution start at a grid level 4 and are subsequently refined using different parameters. The last plot

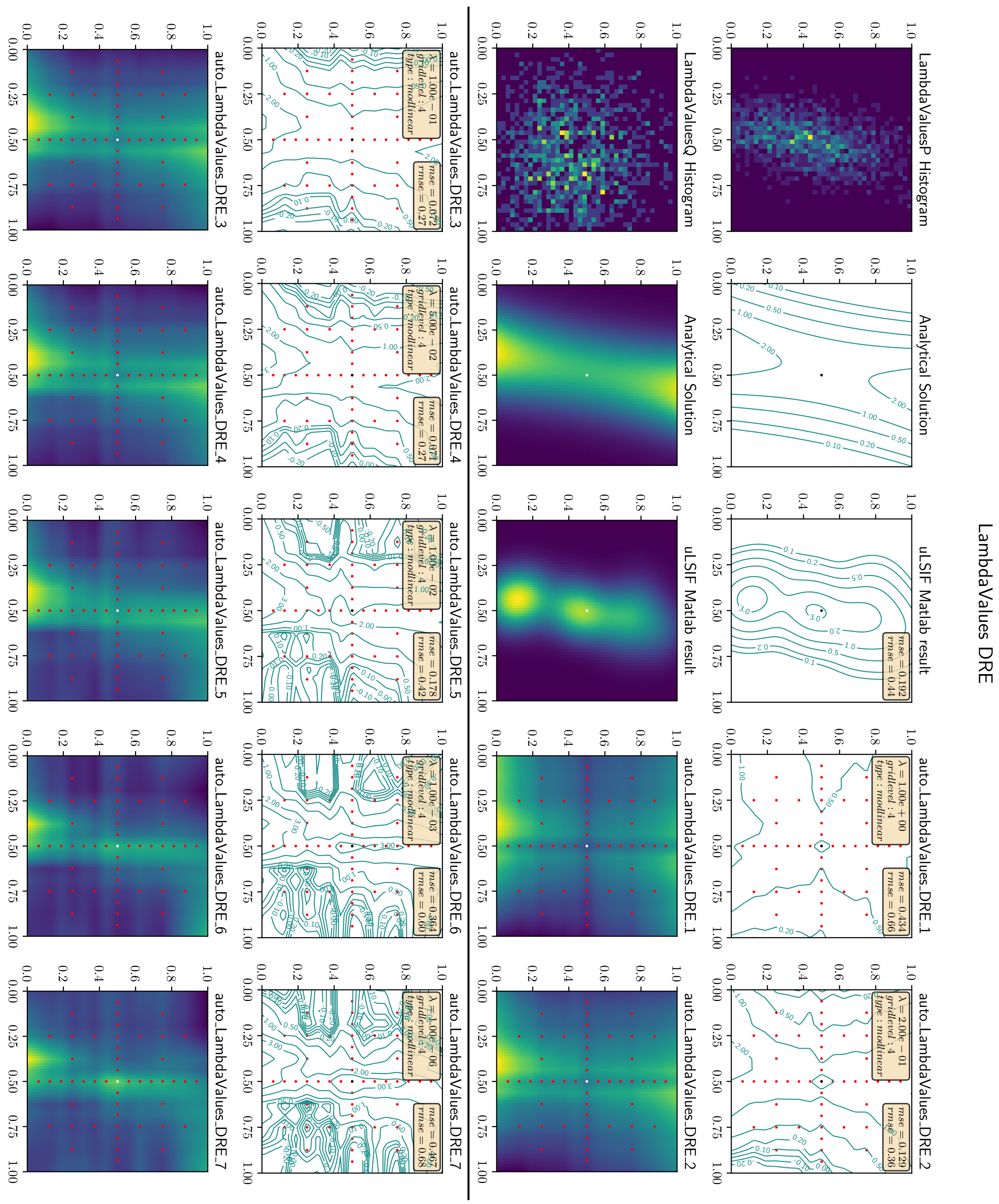

Figure 4.12: TestDRE1.2 compares solutions with different lambda values for density ratio estimation.

shows a level 5 grid as a comparison. By doing this, the grids obtain more or less the same number of total gridpoints which enables us to compare the mse values and the obtained plots.

The results of one of the experiments are visible in Figure 4.15. For the "Q" dataset a wide and fairly flat dataset was chosen to avoid peaks and undefined values where its density might tend to 0, for the P dataset a multivariate normal distribution is used. The analytical solution shows an elliptical peak slightly above the isoline 10. The kernel method is able to accurately mimic the shape of the result, however its highest isoline in the contourplot is at a high of 3. This scale problem is the result of eliminating point which fall outside the hypercube and is explained in detail in subsection 3.1.1. Even with such differences it is still possible to evaluate the density ratio estimations stemming from both examined algorithms, as they both use the same datasets. The analytical solution will only serve to compare the expected shapes as those remain the same.

The experiment compares shallow refinements (low number of refinement steps, high number of refined points in each step) to deep refinements (high number of refinement steps, low number of refined points in each step) when using the SGDRE algorithm. Additionally two regular grids of level 4 and 5 are also shown. Despite its shape being similar to the analytical solution, the kernel based estimation had a worse mse than any of the SGDREs in this experiment. This indicates that linear sparse grids can estimate density ratios accurately despite their shape being coarser than for example smooth kernel functions.

In this testset, although the differences are marginal, the best mse is not obtained when using one of the extremes, but rather using a balanced configuration of 6 refinements with 3 refinement points each. This demonstrates once again how important it is to fit each grid to its application by optimizing the correct parameters. There is no single best configuration for every dataset. By choosing parameters for a specific application, the number of gridpoints can be reduced while still improving the final result compared to a standard setting. This is especially important for higher dimensional datasets.

4.3.2 Varying sample sizes

We have analyzed most parameters used for density ratio estimation, however we still need to examine the influence the number of points in a dataset might have on results. For this purpose, three-dimensional datasets were generated and their mse values plotted in Figure 4.14.The same configuration was used in combination with SGDRE to estimate densities on 100 sizes of datasets. Additionally estimations were also generated with the kernel based method uLSIF.

Since density ratio estimation algorithms can lead to very high values when the density of "Q" gets close to zero and small datasets are used, a logarithmic y-axis is used to display the results of the analysis. In this case, the mse values for any of the algorithms are very high until at least 100 samples, suggesting that density ratio estimation methods do not handle small datasets as well as density difference estimation methods. Comparatively, the LSDD algorithm appears to be more accurate with small

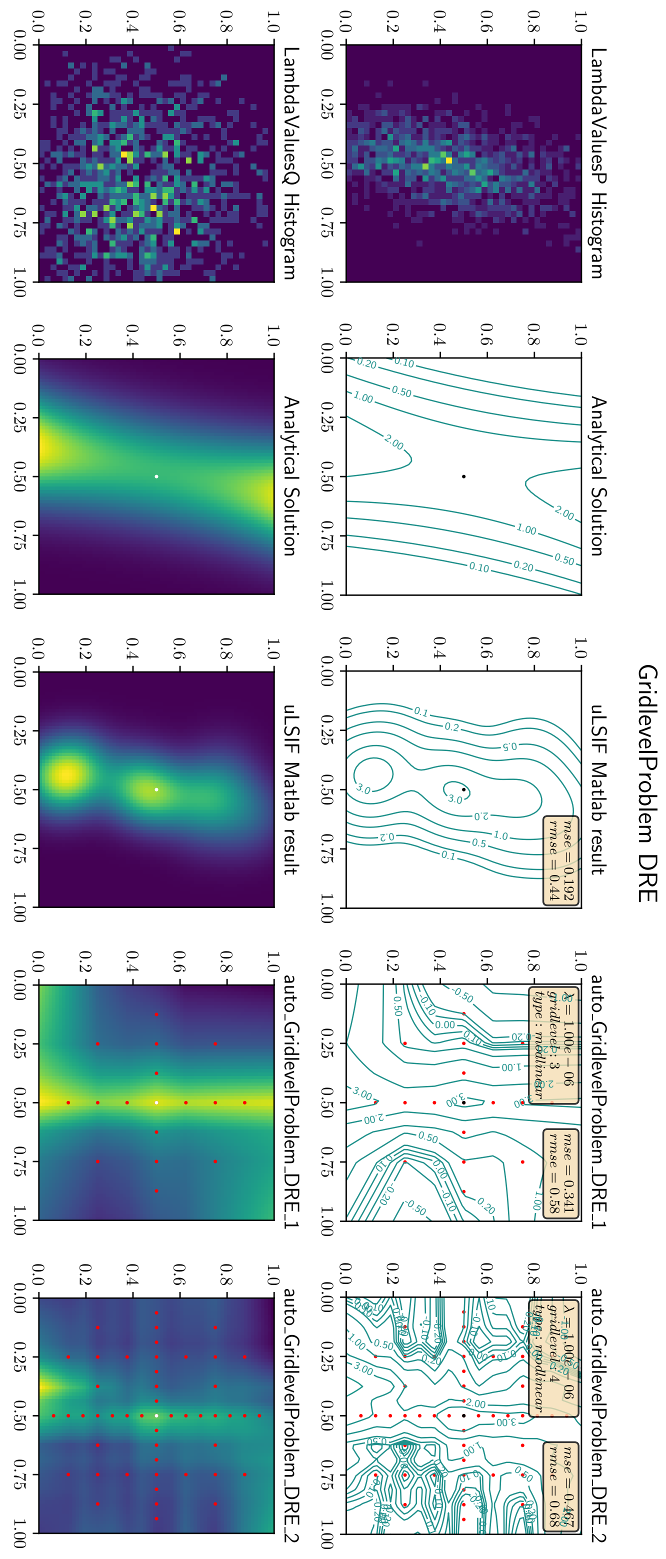

Figure 4.13: Choosing a small gridsize can lead to precision problems.

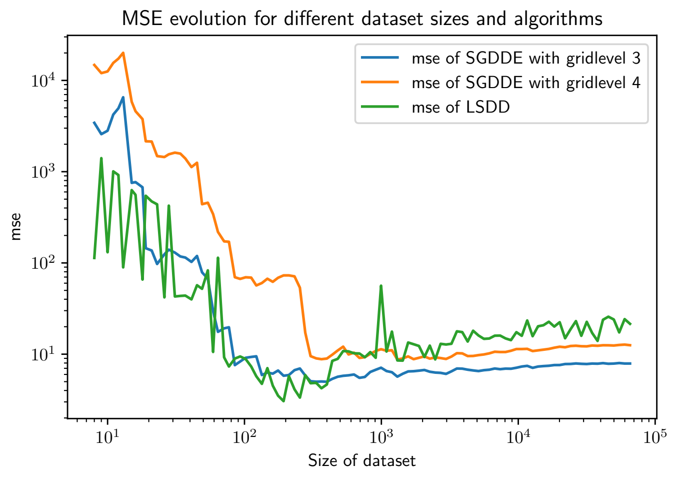

Figure 4.14: This experience compares the mse values of uLSIF and SGDRE algorithms for differently sized datasets

datasets, but its volatility and unpredictability in these ranges might render it unusable in certain applications.

Until around 300 samples, the sparse grid methods in 3D do not have a very good accuracy, but both improve it quickly until their accuracy surpasses the uLSIF method. In this scenario, the grid of level 3 is consistently better than the grid of level 4. During development, other constellations were observed, for example when the higher level was more accurate. As described in the precedent section, the accuracy of the estimations depend heavily on the parameters. Here only one set of parameters is used, except for the gridlevel, reducing the parameter optimization.

None of the algorithms appear to improve their mse values a lot past 300 or 400 samples, in fact the curves suggest that the mse values either stagnate or surprisingly even increase slightly for larger datasets. When working with larger datasets, we once again see that SGDRE achieves a better overall accuracy than its kernel based counterpart.

Estimating density ratios from small datasets is fairly inaccurate because of the fact that low densities in the Q-dataset lead to very high values in the final result. The algorithms therefore require a minimum amount of points to be able to counteract this. Once there are enough sample points, both the LSDD and SGDRE algorithms have comparable results.

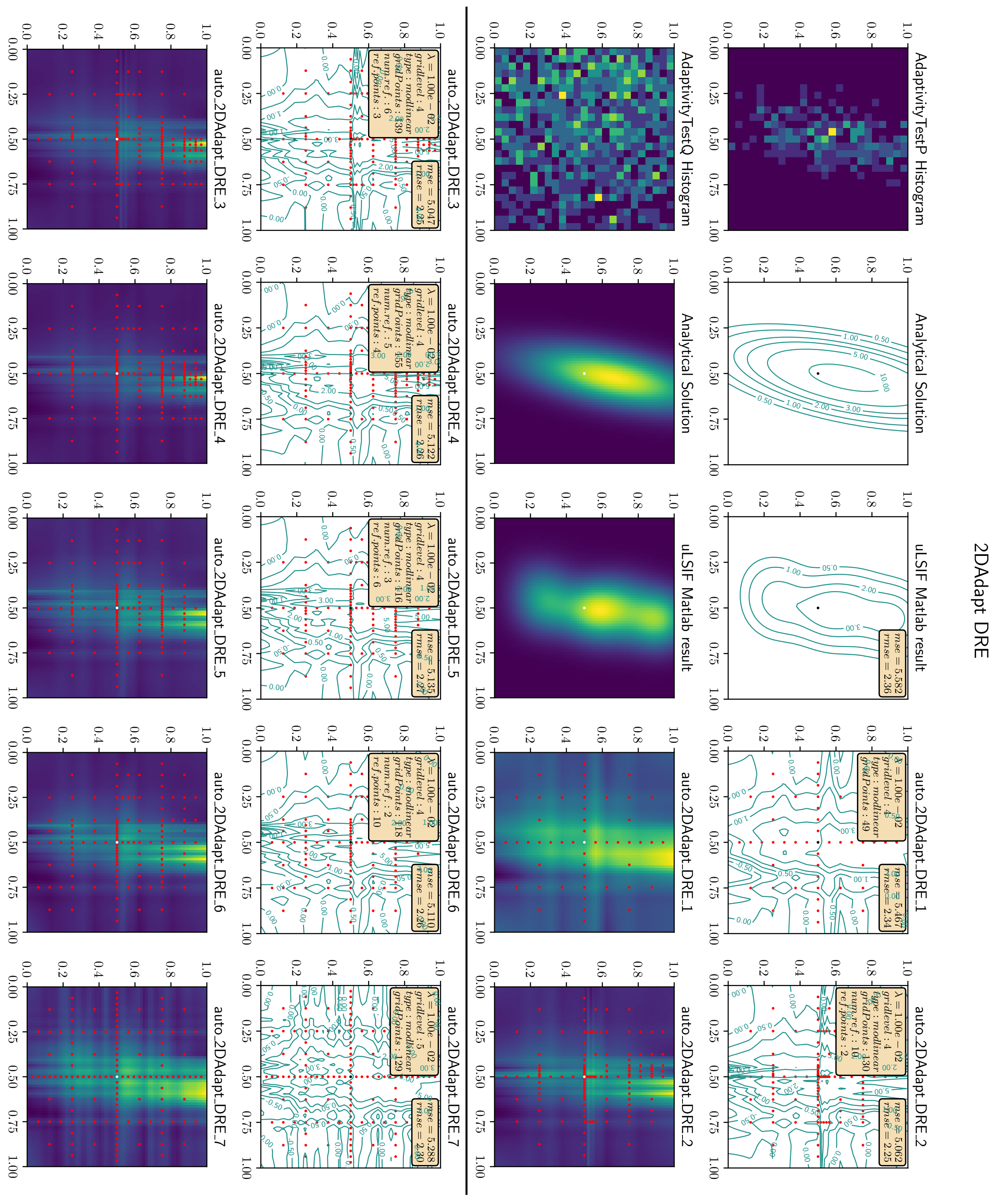

Figure 4.15: TestDRE1.3 compares different refinement configurations of SGDRE

5 Conclusion

In this thesis, both the sparse grid density difference estimation (SGDDE) and sparse grid density ratio estimation (SGDRE) were studied. The evaluation pipeline enables both a visual and numerical evaluation of the functions. Through it, large amounts of experiments could be run with just a few lines of code. The pipeline allows the comparison of parameters by running different configurations on the same datasets P and Q. It was shown that globally, the sparse grid methods had comparable results to the kernel based methods as long as the core parameters, mainly the grid type, the grid level and lambda were properly chosen. When they were optimized, they usually had a better accuracy than the kernel methods. In some cases, especially when the density of Q was very low, the accuracy of SGDRE varied strongly, because a division by a value close to 0 tends to infinity.

Using refinements, the grids can get more accurate estimations with fewer points than an additional level without refinements. It is important to strike the correct balance between the number of refinements and the number of points to refine during each refinement. When steep slopes and sudden changes were detected in the datasets, a lot of refinements with less points were the best suited, when the datasets did not cause spikes, accuracy could be improved by spreading the points evenly in the unit-hypercube.

By running the same configuration with over 100 datasets of different sizes, it was shown that the accuracy of estimations when varying the size of the datasets have a lower limit. This was true for both sparse grid methods and kernel method, as past a number of samples the estimation did not improve but either stagnated or even decreased slightly. The kernel methods appear to be slightly more accurate with very small sample sizes, but often get surpassed by the sparse grid methods when the datasets increase in size.

List of Figures

Bibliography

[1] J. Garcke and M. Griebel. *Sparse Grids and Applications*. ISBN: 978-3-642-31702-6. DOI: `10.1007/978-3-642-31703-3`.

[2] S. Liu, M. Yamada, N. Collier and M. Sugiyama. “Change-point detection in time-series data by relative density-ratio estimation”. In: *Neural Networks* 43 (July 2013), pp. 72–83. ISSN: 0893-6080. DOI: `10.1016/j.neunet.2013.01.012`.

[3] B. Peherstorfer, D. Pflüger 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*, pp. 443–451. DOI: `10.1137/1.9781611973440.51`.

[4] D. Pflüger, B. Peherstorfer and H.-J. Bungartz. “Spatially adaptive sparse grids for high-dimensional data-driven problems”. In: *Journal of Complexity* 26.5 (2010). SI: HDA 2009, pp. 508–522. ISSN: 0885-064X. DOI: `10.1016/j.jco.2010.04.001`.

[5] *SG++ git repository*. URL: `https://github.com/SGpp` (visited on 17/08/2020).

[6] *sparsegrids.org*. URL: `https://sparsegrids.org/` (visited on 17/08/2020).

[7] M. Sugiyama, T. Kanamori, T. Suzuki, M. Christoffel, P. Song and L. I. Takeuchi. “Density-difference estimation”. In: *Neural Computation* 25 (10 Oct. 2013), pp. 2734–2775. ISSN: 0899-7667. DOI: `10.1162/NECO_a_00490`.

[8] *Thesis code repository*. URL: `https://gitlab.lrz.de/bachelorthesis-alex/sgpp-ts-seg`.