Optimized Sparse Grid Combination Technique for Eigenvalue Problems

Anna-Luisa Schwartz

Born October 22, 1987 in Aachen

April 27, 2016

Master's Thesis Mathematics

Advisor: Prof. Dr. Jochen Garcke

Second Advisor: Prof. Dr. Michael Griebel

Institute for Numerical Simulation

Mathematisch-Naturwissenschaftliche Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

Contents

Notation

$\mathbb{N}$	the set of natural numbers: $\{0, 1, 2, 3, \ldots\}$
$\mathbb{R}$	the set of real numbers
$[m : n]$	the subset of $\mathbb{N}$ containing all elements from m until n
$\mathbf{A}$	A Matrix
a_{ij}	Entry of the matrix $\mathbf{A}$ at the crossing of ith row and jth column
$\mathbf{I}$	The identity matrix
$\mathbf{v}$	a vector
v_i	ith entry of vector $\mathbf{v}$
$\mathbf{1}$	column vector with all entries one

Abbreviations

FE	finite element
PDE	partial differential equation
LE	Laplacian Eigenmaps

1. Introduction

With the exponential growth in both processing and storage capacities of modern computers over the past decades, it is now possible to create, collect and store huge amounts of raw, high-dimensional data. It is however nearly impossible for humans to sort and process all of this information. This led to a rise of so-called *machine learning* methods, algorithms that work on the data to group and process it in a meaningful way. Especially interesting in this context are the *unsupervised learning methods*, which do not need any more information than the data itself. One important example for this class of methods is *clustering*, i.e. a division of the raw data into groups of "similar" items. An example application would be online retailing, where it is interesting to segment the customers according to their shopping preferences. A second, similar class of algorithms is formed by *embedding* methods. Here, the algorithm works on the high dimensional data and tries to identify a smaller set of underlying (virtual) parameters, which are enough to describe the data in a meaningful way. Embedding methods are for example used to make relations between high-dimensional data points visible for a human observer by mapping them into 2 or 3 dimensions.

So-called *spectral clustering* and *spectral embedding* methods are modern and often-used tools to perform such a data analysis. First developed in the late nineties for image segmentation purposes [SM00], today such methods are employed to arrange, interpret and analyze all sorts of data, from crash test simulations via customer analysis in online trading to the classification of diseases. Mathematically, these methods are based on the spectral decomposition of a matrix containing information about the similarity between the data points [Lux07].
However, all spectral learning methods are based on the assumption that the relevant data points are given at training time, i.e. the moment the embedding or clustering is calculated. For out-of-sample points that are to be taken into account, a new matrix has to be set up and a new, computationally expensive eigendecomposition needs to be run. This problem does not only occur in the online case, where we will encounter new data points and want to cheaply evaluate them "as they come". Alternatively, one might have a very large data set, and the solution of the resulting large eigenvalue problem (classically in $\mathcal{O}(n^3)$) would simply be too costly. In this case, if one has access to an efficient *out-of-sample-extension* algorithm, one can split the data into a small training set, on which one runs the expensive parts of the computation, and a larger out-of-sample set that can be processed by a simple function approximation.

A common approach to tackle this problem is the data-based Nyström method [BPV$^+$04]. Here, the evaluation function is constructed via an extrapolation

based on kernel functions over the training data points. In this thesis, we instead follow an alternative, grid-based approach by Peherstorfer, Pflüger and Bungartz [PPB11]. As traditional grids suffer from the *curse of dimensionality* and are thus not feasible for the usually very high-dimensional data analysis problems, the authors apply the *sparse grid technique*. This method was originally proposed for high-dimensional quadrature [Smo63] and has been widely expanded over the years. Today it is applied successfully in many different fields of mathematics, mostly the numerical solution of partial differential equations [BG04]. Its success is based on the fact, that instead of $\mathcal{O}(n^d)$ grid points in d dimensions, only $\mathcal{O}(n(\log n)^{d-1})$ grid points are needed to approximate sufficiently smooth functions, at a similar quality.

In contrast to Peherstorfer et al., who use the classic sparse grid method based on rather complicated, hierarchical basis functions, we suggest to instead apply *sparse grid combination techniques* to the problem of out-of-sample-extension. This class of methods decomposes the sparse grid to several coarser, anisotropic grids, which are easier to implement and to handle. An approach to using the *optimized combination technique (Opticom)* has been presented by Garcke in 2014 [Gar14], but has not yet been analyzed in more detail.

The aim of this thesis is first to comprehensively collect results on the numerical (finite-element-based) solution of eigenvalue problems, on the sparse grids combination techniques and on spectral embedding algorithms and to combine them all for derivation and analysis of a new algorithm for out-of-sample-extension for spectral data analysis. The theoretical considerations will be sided by numerical experiments, using program code that has been newly implemented to support this thesis.

Contributions

The main contributions of this thesis can be summarized as follows:

- Introduction of a framework to perform out-of-sample-extension for Laplacian Eigenmaps using the optimized sparse grids combination technique (Opticom)
- Comprehensive overview over underlying methods and presentation of the new approach in several contexts
- Extensive, class-based implementation of the proposed framework in C++
- Practical and theoretical analysis of the algorithm

Layout

The thesis is structured as follows. The next three chapters outline the basic frameworks that will be applied subsequently, beginning with Chapter 2 on the numerical approximation of eigenvalue problems. Here, some numerical solvers

for matrix eigenvalue problems are introduced, followed by an outline of the so-called Babuška-Osborn theory for the finite element approximation of eigenvalue problems.
Chapter 3 will then proceed to sparse grids methods, where the focus lies on the combination technique and the optimized combination technique (Opticom) and the application of these techniques to problems related to the applications in this thesis, namely machine learning and eigenvalue problems.
The idea of spectral data analysis methods is then described in detail in Chapter 4, where we start with a general motivation and overview to then describe one of the algorithms, Laplacian Eigenmaps, in detail. This chapter also covers convergence results for this family of spectral methods and gives a brief introduction to the existing approach for out-of-sample-extension, the so-called Nyström method.
In the following Chapter 5 we will use these foundations to establish and analyze our new proposed algorithm. We will consider the challenges and advantages of this new approach, show how it can be interpreted in existing frameworks and analyze its complexity.
This will be followed up by a chapter on numerical experiments, Chapter 6, where the algorithm is put to the test in several different clustering and embedding set-ups. Moreover, some studies on parameter choices and consistency are performed.
The concluding Chapter 7 will outline the results and provide a more general outlook for the topic.

2. Numerical Treatment of Eigenvalue Problems

Numerical methods are applied to solve or approximate eigenvalue problems in many domains, from iterative methods for the eigendecompositions of large matrices to finite-element-methods for the computation of eigenfunctions of infinite-dimensional differential operators.
The problems considered in this thesis lie in between these applications: The original problems from machine learning are finite-dimensional and stated as matrix problems (however possibly quite large). At the same time, we intend to find continuous functions extending these finite-dimensional eigenvectors, and to that end we apply finite element grids.
To deal with the arising finite-dimensional problems, we will shortly summarize Krylov subspace methods for the eigendecomposition of matrices, as well as the FEAST algorithm, which we use in our proposed algorithm.
As the sparse grid techniques (introduced in the subsequent Chapter 3) have already been applied for the determination of eigenvalues and -functions of differential operators, and we will consider similar (but unknown) operators, we will then present the very basics of the Babuška-Osborn theory [BO89, BO91], which constitutes the foundation for the finite-element approximation of eigenvalue problems.

2.1. Matrix Eigenvalue Problems

Let us start with the well-known definition of the eigenvalue problem for a real, square matrix.

Definition 1 (Eigenvalues of a real square matrix)**.**
A scalar $\lambda \in \mathbb{R}$ is called an eigenvalue *of the square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, if there exists a nonzero vector $\mathbf{x} \in \mathbb{R}^n$ such that*

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}. \tag{2.1}$$

The vector $\mathbf{x}$ is called an eigenvector *of $\mathbf{A}$ associated with λ. The set of all eigenvalues of $\mathbf{A}$ is called the* spectrum *of $\mathbf{A}$ and is denoted by $\Lambda(\mathbf{A})$.*

In the following, we will briefly describe a class of numeric techniques to solve such eigenvalue problems, especially in the case of large and sparse matrices. Let us just add a note in beforehand: The problems considered in this thesis are often not in the standard form 2.1, but stated in the so-called generalized form, with another square matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ on the right hand side:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{B}\mathbf{x}. \tag{2.2}$$

In that case, we call λ a *generalized eigenvalue* and $\mathbf{x}$ the corresponding *generalized eigenvector*. Trivially, this problem could be reduced to the standard form as $\mathbf{B}^{-1}\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$. However, this will not be possible, if the matrix $\mathbf{B}$ is singular. In that case still, generalized eigenvalues of Problem 2.2 may exist [Saa11].
If the matrices $\mathbf{A}$ and $\mathbf{B}$ are both symmetric and $\mathbf{B}$ is positive definite, the standard algorithms for the symmetric case (especially the Lanczos methods, presented subsequently) can also be applied under some modifications [Saa11]. Most importantly, the Euclidean inner product will need to be replaced by the inner product defined by $\mathbf{B}$, $\langle x, y\rangle_B := \langle x, \mathbf{B}y\rangle$.

2.1.1. Krylov Subspace Methods

The matrices considered in this thesis are often grid based and thus have the potential to become very large. Unfortunately, many established methods for the solution of eigenvalue problems of a given matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, such as the power method, the QR decomposition or Jacobi methods [GL12] require computational efforts of order $\mathcal{O}(n^3)$, which will be unfeasible for large n.

The class of so-called *Krylov subspace methods* works in a way that avoids expensive matrix-matrix operations but instead uses only matrix-vector operations. Starting with a vector $\mathbf{z} \in \mathbb{R}^n$ one calculates $\mathbf{A}\mathbf{z}$ and iteratively $\mathbf{A}^2\mathbf{z}$ and so on. The k-dimensional subspace spanned by these vectors,

$$\mathbb{R}^n \supset \mathcal{K}_k(\mathbf{A}, \mathbf{z}) := \operatorname{span}(\mathbf{z}, \mathbf{A}\mathbf{z}, \mathbf{A}^2\mathbf{z}, \ldots, \mathbf{A}^{k-1}\mathbf{z})$$

is called a *Krylov subspace*. Well-known examples of algorithms for the solution eigenvalue problems via working on these spaces are the *Lanczos algorithm* for Hermitian eigenvalue problems and the more general *Arnoldi's method*. Both methods are based on the use of orthogonal projections onto Krylov subspaces. To this end, an iterative algorithm is performed, starting with an initial vector $\mathbf{z} \in \mathbb{R}^n$, chosen randomly or applying some sort of *filter* $\mathbf{z} = \rho(\mathbf{M})\mathbf{z}^{(0)}$ on a random $\mathbf{z}^{(0)} \in \mathbb{R}^n$. In iteration k of the algorithm an orthonormal basis of $\mathcal{K}_{k-1}(\mathbf{A}, \mathbf{z})$ is extended to an orthonormal basis of $\mathcal{K}_k(\mathbf{A}, \mathbf{z})$ via a variation of the classical Gram-Schmidt orthogonalization process. Then, one computes the eigenvalues of the orthogonal projection $\mathbf{H}_k$ of $\mathbf{A}$ onto the subspace. These so-called Ritz eigenvalues will typically converge to the extreme eigenvalues of $\mathbf{A}$. In the general Arnoldi case however, a thorough mathematical foundation has not yet been given. The Lanczos case for symmetric matrices $\mathbf{A}$ has a more complete theory [Saa11].
Especially for large sparse matrices, where these matrix-vector-operations can be implemented very efficiently, Krylov subspace methods have become the standard method to solve eigenproblems.

We will refrain from going further into detail, as in this thesis we applied a modern, alternative solver: FEAST, introduced in the upcoming section. A very thorough overview on the numerical solution of matrix eigenvalue problems that gives many different statements and implementations of the aforementioned algorithms can be obtained from the classic textbooks by Saad [Saa11] (especially Chapter 6) and

Golub and Van Loan [GL12] (especially Chapter 10).

2.1.2. The FEAST Algorithm

For the matrix eigenvalue problems in this thesis, we applied a rather modern approach: the so-called FEAST algorithm developed by Eric Polizzi [Pol09]. It was especially designed for large sparse generalized eigenproblems, and is built in a way that allows for parallelization at multiple steps in the algorithm.
The algorithm is based on ideas from quantum mechanics, the so-called density matrix-method and the contour iteration technique [Pol09]. For a given interval $[\lambda_{\min}, \lambda_{\max}]$ or a contour in the complex plane, it will find all eigenvalues within that contour and the associated eigenvectors. To this end, the algorithm performs a numerical integration along a complex contour by solving several independent linear systems, each with varying right hand sides. Then an eigenvalue problem is produced, which is several orders of magnitude smaller (the size of the number of eigenvalues present inside the given interval/contour) and can be solved with a standard method like the QR algorithm. All computational complexity thus stems from the linear systems on the contour, which can be solved in parallel and very efficiently with modern solvers. Due to the broad theory from different fields of mathematics and quantum mechanics, we will not describe the algorithm in further detail, but refer to the original publication [Pol09] and the additional paper [PT14], in which the authors proved, that their algorithm corresponds to a certain class of subspace iteration methods (cf. [Saa11], Chapter 5) combined with a filter based on a spectral projection. For the understanding of the work in this thesis, this basic characterization of the method will be sufficient.

2.2. Finite Element Approximation

For the approximation of eigenvalues and eigenvectors of differential operators, finite elements methods have been established as a valuable numerical tool. As in the classic Ritz-Galerkin approach for boundary value problems, we start from a variational formulation of the eigenvalue problem [Bof10, BO91]. In the remainder of this section, unless otherwise specified, we use notation and definitions according to the extensive survey on finite element approaches to eigenvalue problems by Boffi, [Bof10].

Let V and H be real Hilbert spaces, $V \subset H$ compactly embedded and $a : V \times V \longrightarrow \mathbb{R}$ and $b : H \times H \longrightarrow \mathbb{R}$ be symmetric and continuous bilinear forms. We additionally suppose that a is coercive,

$$\exists \alpha > 0, \text{ such that } a(v,v) \geq \alpha \, ||v||_V^2 \quad \text{for all } v \in V,$$

and that b is positive definite, i.e.,

$$b(v,v) > 0 \quad \text{for all } v \in V \setminus \{0\}.$$

Then we can formulate the following symmetric variationally posed eigenvalue problem:

Problem 2 (Symmetric variationally posed eigenvalue problem)**.**
Find $\lambda \in \mathbb{R}$ *and* $u \in V \setminus \{0\}$ *such that*

$$a(u, v) = \lambda b(u, v) \quad \text{for all } v \in V. \tag{2.3}$$

We then call λ a *generalized eigenvalue* of 2.3 and the corresponding element $u \in V \setminus \{0\}$ the associated *eigenvector*. If b corresponds to the scalar product of H, as is often the case in applications, λ is simply called an *eigenvalue* of 2.3.

To find a solution to Problem 2, we analyze it in terms of the well-understood spectral theory for compact operators. To this end, we consider the solution operator $T : H \to H$. By the Lax-Milgram theorem and our assumptions, given any $f \in H$ there is a unique $Tf \in V$ fulfilling

$$a(Tf, v) = b(f, v) \quad \text{for all } v \in V.$$

Due to our assumption that $V \subset H$ compactly, making T a compact operator, we can now apply results from spectral theory on compact, self-adjoint operators [Kat76, Alt12]. We can deduce that for Equation 2.3 there exists an infinite sequence of eigenvalues, $0 < \lambda_1 \leq \lambda_2 \leq \lambda_3 \leq \ldots$, and their assigned eigenfunctions $u_1, u_2, u_3, \ldots$ are orthogonal with respect to the bilinear forms a and b:

$$a(u_k, u_l) = b(u_k, u_l) = 0 \quad \text{if } k \neq l.$$

In case of multiple eigenvalues, the eigenfunctions can be chosen from the respective eigenspace, such that these orthogonalities hold as well.
Additionally, the eigenfunctions are considered normal in the sense that $b(u_k, u_k) = 1$. Note that this normalization is unique only up to its sign.
If b is the scalar product on H the eigenfunctions will thus constitute an orthonormal basis of H.

To pass to a finite-dimensional approximation of the eigenvalue problem, we consider the Rayleigh quotient $R : V \setminus \{0\} \longrightarrow \mathbb{R}$ for Problem 2:

$$R(v) := \frac{a(v, v)}{b(v, v)}$$

It can be shown, that the Rayleigh quotient is closely connected to the eigenproblem, as

$$\lambda_1 = \min_{v \in V \setminus \{0\}} R(v), \quad u_1 = \arg\min_{v \in V \setminus \{0\}} R(v)$$

and

$$\lambda_k = \min_{v \in U_{k-1}^{\perp} \setminus \{0\}} R(v), \quad u_1 = \arg\min_{v \in U_{k-1}^{\perp} \setminus \{0\}} R(v),$$

where $U_k = \operatorname{span}\{u_1, \ldots, u_k\}$ and the orthogonal complements are taken on V with respect to the scalar product induced by b.

For the Galerkin-discretization of problem (2.3) we choose a finite-dimensional subspace $V_h \subset V$ and consider the discrete problem:

Problem 3 (Discretized eigenvalue problem)**.**
Find $\lambda_h \in \mathbb{R}$ *and* $u_h \in V_h \setminus \{0\}$ *such that*

$$a(u_h, v_h) = \lambda_h b(u_h, v_h) \quad \text{for all } v_h \in V_h. \tag{2.4}$$

From here, we can take the same steps as before for the analysis of the problem: V_h is a Hilbert subspace of V, so we can define a discrete solution operator $T_h : H \longrightarrow H$, which is uniquely defined and compact. In particular, we can deduce that there exists a finite set of *discrete eigenvalues*, $0 < \lambda_{h,1} \leq \lambda_{h,2} \leq \ldots \leq \lambda_{h,N}$ with $N \in \mathbb{N}$ the dimension of V_h. The associated eigenvalues $u_{h,1}, u_{h,2}, \ldots, u_{h,N}$ fulfill the same orthogonalities as the continuous ones,

$$a(u_{h,k}, u_{h,l}) = b(u_{h,k}, u_{h,l}) = 0 \quad \text{if } k \neq l,$$

and analogous results hold for the discrete versions of the Rayleigh quotients.
The applicability of this Galerkin-method now depends on the relationship between the discrete and the original eigenvalues. And indeed, using the Rayleigh quotient and the *min-max-characterization* [Bof10],

$$\lambda_k = \min_{E \in V^k} \max_{v \in E} R(v), \quad \text{where } V^k = \{\, W \subset V \mid \dim(W) = k \,\},$$

which is also valid for the discrete problem, it can be shown that for a conforming approximation $V_h \subset V$ the exact eigenvalues are bounded from above by their discrete counterparts [Bof10],

$$\lambda_k \leq \lambda_{h,k} \quad \text{for all } k \in [1 : N].$$

Based on these results, one can safely apply Galerkin discretizations to approximate the eigenvalues of Equation 2.3. A classic choice of spaces are finite element spaces, for which convergence of the discrete eigenvalues to the exact ones can be proven for declining mesh widths.

Like for the Ritz-Galerkin methods for the solution of partial differential equations, we can then treat the computation of the eigenvalues and eigenfunctions of Problem 3 as a classic matrix eigenvalue problem: We choose V_h as a finite-element space with nodal basis $\Phi = \{\phi_i\}_{i=1}^N$. Then we can represent the eigenfunctions $u_h \in V_h$ via their coefficients in that basis representation, $u_h = \sum_{i=1}^N c_i \phi_i$ and test against this basis Φ, representing all $v_h \in V_h$ in Equation 2.4. Thus one obtains the generalized matrix eigenvalue problem

$$\mathbf{A}\mathbf{c} = \lambda_h \mathbf{B}\mathbf{c},$$

with the matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{N \times N}$, $a_{ij} = a(\phi_i, \phi_j)$ and $b_{ij} = b(\phi_i, \phi_j)$ discrete versions of the bilinear forms a and b. This problem then may be solved with, e.g., one of the methods introduced in the previous section.

In this thesis, we will need the applicability of the Galerkin method for eigenvalue problems to apply the Opticom method to eigenvalue problems, as in Section 3.3, and to provide a numerical derivation of our algorithm in Chapter 5.

3. Sparse Grids

In this chapter we will present the foundations of the sparse grid method. We will begin with a brief introduction of the idea and technique itself and then proceed to the description of combination techniques. This class of techniques will in some cases allow for a more efficient computation, while still being comparably accurate. For this thesis, the focus will be on an application of the optimized combination technique (Opticom), which will therefore be presented in more detail.
To conclude the chapter, we will show previous applications of Opticom and the classic combination technique for eigenvalue problems as well as for machine learning, since some of the challenges encountered in these scopes carry over to the setting of this thesis.

3.1. Overview

The sparse grid technique has been developed by Zenger [Zen91] in 1990 to efficiently discretize function spaces in high dimensions. Originally designed in the context of solving partial differential equations, the method now is applied in various different numerical settings, from physics simulations over machine learning applications to financial mathematics.
The technique is based on an idea by Smolyak [Smo63] aiming to counter the curse of dimensionality in numerical integration. Key idea is to use sparse tensor products of a hierarchical basis of piecewise linear functions. The classical nodal basis of hat functions can be replaced by such a hierarchical basis spanning the same function space and can then be reduced by omitting those basis functions with a small contribution to the overall approximation. In this section, we will mostly follow the derivation of sparse grids provided by Jochen Garcke in his survey [Gar13]. His approach is designed to lead to the introduction of the sparse grid combination techniques, which will be applied in this thesis.

For simplicity, we consider the d-dimensional unit cube $\Omega = [0,1]^d$. All results and definitions are applicable for any bounded rectangular domain by suitable rescaling.
We will start the construction of sparse grids with the classical full grid discretization approach for anisotropic grids. This discretization is defined via a level vector $\mathbf{l} \in \mathbb{N}^d$, yielding the grid

$$\Omega_{\mathbf{l}} := \left\{ \mathbf{x} \in \Omega \;\middle|\; x_i = j \cdot 2^{-l_i} \text{ for all } i \in [1:d], j \in \left[0 : 2^{l_i}\right] \right\}$$

and corresponding multilinear nodal basis functions,

$$\Phi_{\mathbf{l},\mathbf{j}}(x) := \prod_{i=1}^{d} \phi_{l_i,j_i}(x_i)$$

with $\mathbf{j}$ the multivariate position index, $j_i \in \left[0 : 2^{l_i}\right]$ for all $i \in [1:d]$, and $\phi_{l,j}$ the well-known one-dimensional hat functions,

$$\phi_{l,j}(x) := \begin{cases} 1 - \left|x/2^{-l} - j\right|, & x \in \left[(j-1)2^{-l}, (j+1)2^{-l}\right] \cap [0,1]^d \\ 0, & \text{otherwise.} \end{cases}$$

As usual, the mesh width in the i-th coordinate direction is defined by $h_i = 2^{-l_i}$. We denote the resulting full grid space corresponding to the grid $\Omega_{\mathbf{l}}$ by

$$V_{\mathbf{l}} := \operatorname{span}\left\{ \Phi_{\mathbf{l},\mathbf{j}} \;\middle|\; j_i \in \left[0 : 2^{l_i}\right] \text{ for all } i \in [1:d] \right\}.$$

For an isotropic full grid, we will in our notation replace the vector $\mathbf{l} \in \mathbb{N}^d$, $l_i = k$ constant for all $i \in [1:d]$, with the respective scalar $k \in \mathbb{N}$.

To now construct sparse grids, let us consider a classic isotropic full grid Ω_k, k fixed, and switch to a hierarchical basis of its associated function space V_k. This basis is built starting with the basis functions of the coarsest subgrid V_0 and then repetitively extended by adding basis functions from the next finer level as follows. Let

$$\mathcal{I}_l := \begin{cases} \left\{ j \in \left[0 : 2^l\right] \;\middle|\; j \text{ odd} \right\}, & l \geq 1 \\ \{0,1\}, & l = 0 \end{cases}$$

be a one-dimensional index set and let the Cartesian product $\mathcal{I}_{\mathbf{l}} = \prod_{i=1}^{d} \mathcal{I}_{l_i}$ be the corresponding multi-dimensional index set. For every $\mathbf{l}$, $\mathcal{I}_{\mathbf{l}}$ denotes the indices that were not present in any of the smaller levels.
In Figure 3.1, for each level l the respective indices of $\mathcal{I}_l$ in one dimension are marked in bold font.
Then $W_{\mathbf{l}} := \operatorname{span}(\{ \Phi_{\mathbf{l},\mathbf{j}} \mid \mathbf{j} \in \mathcal{I}_{\mathbf{l}} \})$ is a hierarchical increment space, as

$$W_{\mathbf{l}} = V_{\mathbf{l}} \setminus \bigoplus_{i=1}^{d} V_{\mathbf{l}-\mathbf{e}_i},$$

where $\mathbf{e}_i$ denotes the i-th unit vector and $V_{\mathbf{l}} := 0$ if $l_i < 0$ for any $i \in [1:d]$. Thus we can write the isotropic full grid space V_k as a direct sum of these increment spaces:

$$V_k = V_{(k,\ldots,k)} = \bigoplus_{|\mathbf{l}|_\infty \leq k} W_{\mathbf{l}}$$

The first one-dimensional spaces W_i and a comparison of the hierarchical and the nodal bases are depicted in Figure 3.2 [FPR+10].

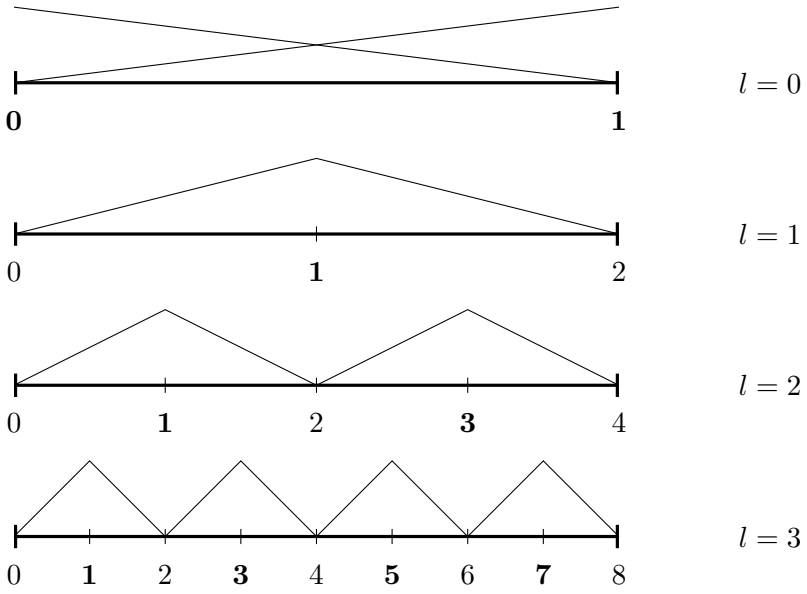


Figure 3.1.: The one-dimensional spaces $W_l = \operatorname{span}\{\, \phi_{l,j} \mid j \in \mathcal{I}_l \,\}$ for $l \leq 3$. The nodes from the index set $\mathcal{I}_l$ are marked in bold for each level.

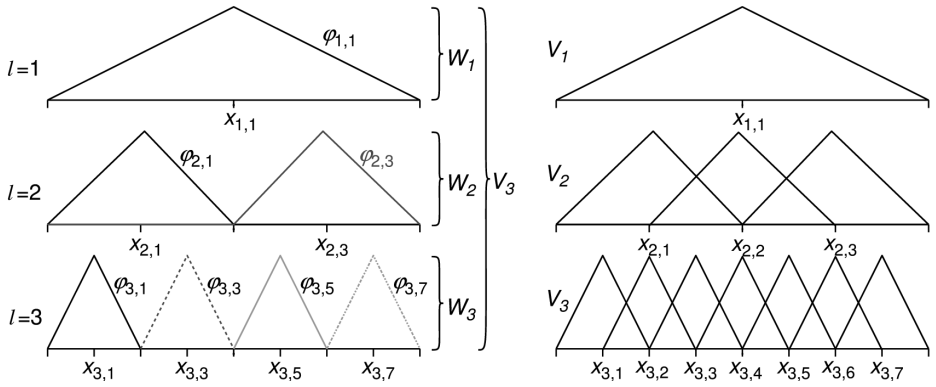


Figure 3.2.: On the left, the one-dimensional space V_3 represented by its hierarchical basis as the sum of the increment spaces W_1, W_2 and W_3, on the right hand side the classic nodal basis representation for all spaces V_i, $i \leq 3$. Note that the basis functions of the hierarchical increment spaces have disjoint supports. Source of figure: [FPR+10].

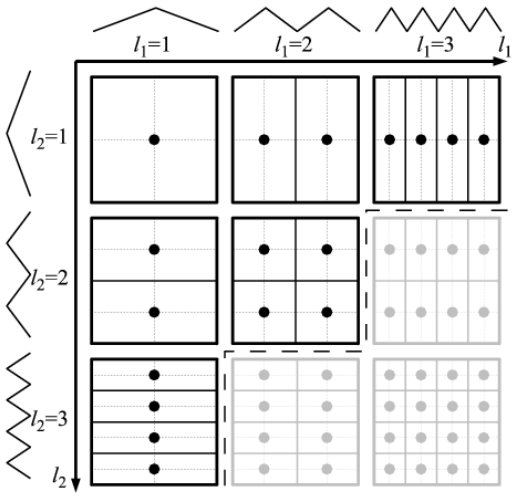


Figure 3.3.: The two-dimensional hierarchical subspaces of V_3: $W_\mathbf{l}$ with $0 < l_1 \leq 3$, $0 < l_2 \leq 3$. The basis functions are formed as tensor products of the one-dimensional ones. Using only the subspaces marked in black, one obtains the somewhat optimal choice of basis functions for the sparse grid space $V^s_{0,2}$ as defined in definition 4. Source of figure: [FPR+10].

Using the hierarchical basis, we can now write every piecewise linear function $f \in V_k$ as

$$f(x) = \sum_{|\mathbf{l}|_\infty < k} \sum_{\mathbf{j} \in \mathcal{I}_\mathbf{l}} \alpha_{\mathbf{l},\mathbf{j}} \Phi_{\mathbf{l},\mathbf{j}}(x) = \sum_{|\mathbf{l}|_\infty < k} f_\mathbf{l}(x),$$

where the $f_\mathbf{l} \in W_\mathbf{l}$ denote the so-called *hierarchical component functions* of f. As our aim is to reduce the complexity of the grid-based approximation while keeping accuracy high, we can now determine which of the hierarchical component spaces contribute the least to the approximation. If we measure the error both in the L^2 or L^∞ norm, the optimal (error versus number of grid nodes) choice of basis functions is to use the spaces $W_\mathbf{l}$ with $|\mathbf{l}|_1 \leq k + d - 1$. A detailed proof and examples can be found e.g. in the extensive survey by Bungartz and Griebel, [BG04]. Simply put, the importance of the basis functions diminishes with the size of their respective supports, so we only keep the hierarchical increment spaces with relatively large supports, cf. Figure 3.3.

The described cut-off now leads to the following definition of sparse grid spaces, as provided by Zenger [Zen91]:

Definition 4 (Sparse grid function spaces)**.**

$$V^s_{0,k} := \bigoplus_{|\mathbf{l}|_1 \leq k+d-1} W_\mathbf{l}, \quad l_i > 0 \text{ for all } i \in [1:d]$$

Note that due to the positivity condition this definition does not contain any degrees of freedom on the boundaries and is thus particularly useful for problems

with Dirichlet boundary conditions. An alternative definition, used e.g. in [Gar13] is

$$V_k^s := \bigoplus_{|\mathbf{l}|_1 \leq k} W_\mathbf{l},$$

including degrees of freedom on the boundary. The respective grids will analogously be denoted by $\Omega_{0,k}^s$ and Ω_k^s.

To precisely describe the approximation quality, we will consider functions from the function space $H^2_{\text{mix}}([0,1]^d) = H^2([0,1]) \otimes \cdots \otimes H^2([0,1])$, the *Sobolev space with dominating mixed derivatives*, which proves to be a very natural space working with sparse grid functions, as it mirrors the tensor product structure of the hierarchical basis functions. For details and the original definition see [Gar13, BG04].

Theorem 5 (Approximation quality of sparse grid functions)**.**
In the L^2- and L^∞-norms, the approximation error for a function $f \in H^2_{mix}$ in the sparse grid space $V_{0,k}^s$ amounts to:

$$||f - f_k^s||_2 = \mathcal{O}(h_k^2 \cdot k^{d-1}), \tag{3.1}$$
$$||f - f_k^s||_\infty = \mathcal{O}(h_k^2 \cdot k^{d-1}). \tag{3.2}$$

Proof and approximation errors in further norms can be found in [BG04].
To finish the consideration of the properties of sparse grid spaces, we now look at the number of degrees of freedom:

Theorem 6 (Number of inner grid points, [BG04])**.**
The dimension of the space $V_{0,k}^s$, i.e., the number of degrees of freedom or inner grid points, is given by

$$\left|V_{0,k}^s\right| = \mathcal{O}(2^k \cdot k^{d-1}).$$

Note that this complexity makes sparse grids a lot cheaper than the classic full grid space V_k with size $\mathcal{O}(2^{k \cdot d})$, while the approximation quality is only slightly worse.

As the sparse grids approach dampens, but not fully overcomes the curse of dimensionality, refinement techniques have been developed to further improve dealing with higher dimensions, notably spatial adaptivity approaches [Pfl10] and dimension-adaptive approaches [Heg02].

In the following sections, we will now consider combination techniques, a method to use sparse grids without relying on the hierachical basis, but instead considering them as the combination of several coarser, anisotropic grids.

3.2. Sparse Grid Combination Technique

To ease computation of the sparse grid method, the sparse grid combination technique [GSZ92] was introduced in 1992. Key idea is to consider a sequence of

finite-element grids with their nodal bases, solve the problem on these spaces using efficient standard algorithms and then combine these solutions. To this end, we represent the sparse grid Ω_k^s as the superposition of "classical", coarser anisotropic grids $\Omega^{(i)}$ with their respective function spaces $V^{(i)}$,

$$\Omega_k^s = \Omega^{(1)} \cup \ldots \cup \Omega^{(m)}. \tag{3.3}$$

A function f can then be discretized as a linear combination of the respective discretizations on these subspaces,

$$f_k^c = \sum_i c_i f_i, \tag{3.4}$$

with $f_i \in V^{(i)}$ and thus all the f_i represented in a "simple" nodal basis structure. This approach has several advantages: First, as mentioned above, for these well-studied spaces, efficient algorithms to solve PDEs already exist. These make use of the sparse structure of the standard FE matrices, a sparsity not given when we use the hierarchical basis of sparse grids, due to the large supports of the low-level basis functions intersecting with supports of many finer-level basis functions. Secondly, by splitting the problem into independent subproblems, the computation is trivially parallelizable and thus can be implemented very efficiently.

The partial spaces considered follow a simplicial structure: We use those grids $\Omega_{\mathbf{l}}$ whose levels $\mathbf{l}$ fulfill $|\mathbf{l}|_1 = k - q$ for all $q \in [0 : d-1]$, or, to neglect the degrees of freedom on the boundary, $|\mathbf{l}|_1 = k + (d-1) - q$. The simplified indices from Equation 3.3 are obtained by suitable numbering of the subspace levels.
In the following, we will use the term *layers*, where for each $q \in [0 : d-1]$ the respective layer is defined as the set of all grids with $|\mathbf{l}|_1 = k - q$, see also Figure 3.4.
As several nodes of such a combined grid will appear in more than one of the subgrids considered, we will need to apply the inclusion-exclusion principle from combinatorics, yielding the following formula for the approximation of a function $f \in H$, with H some suitable function space:

$$f_k^c(\mathbf{x}) := \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\mathbf{l}|_1 = k-q} f_{\mathbf{l}}(\mathbf{x}),$$

or, with no degrees of freedom on the boundary (the natural case for PDEs with Dirichlet boundary conditions),

$$f_k^c(\mathbf{x}) := \sum_{q=0}^{d-1} (-1)^q \binom{d-1}{q} \sum_{|\mathbf{l}|_1 = n+(d-1)-q} f_{\mathbf{l}}(\mathbf{x}). \tag{3.5}$$

Here, the $f_{\mathbf{l}} \in V_{\mathbf{l}}$ are the approximations of f in the standard nodal basis of the regular anisotropic finite element space $V_{\mathbf{l}}$ as defined in the section before,

$$f_{\mathbf{l}} = \sum_{j_1=0}^{2^{l_1}} \ldots \sum_{j_d=0}^{2^{l_d}} \alpha_{\mathbf{l},\mathbf{j}} \Phi_{\mathbf{l},\mathbf{j}}.$$

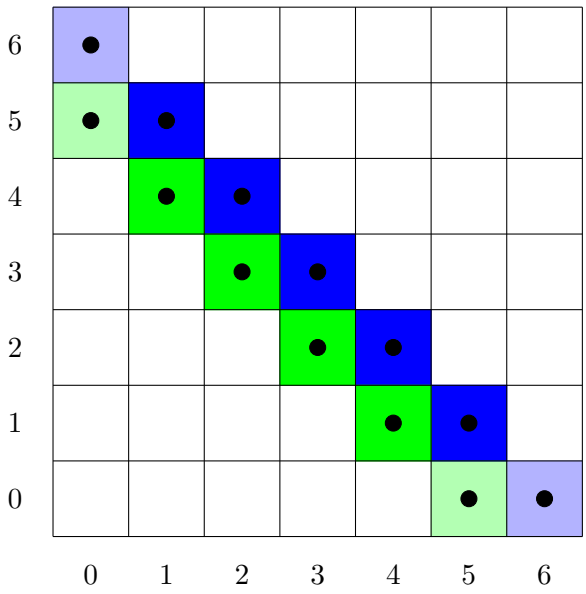


Figure 3.4.: The simplicial structure of the regular anisotropic grids that form the combined sparse grid, for level $l = 6$. The upper diagoal row, marked in blue (the *first layer* of partial grids are these grids with $|\mathbf{l}|_1 = 6$, the green row marks the *second layer* with $|\mathbf{l}|_1 = 5$. Often, one might wish to leave out highly anistropical grids. As an example in the figure the grids featuring only a resolution of 0 in one coordinate direction are shaded lighter and might be left out in the method.

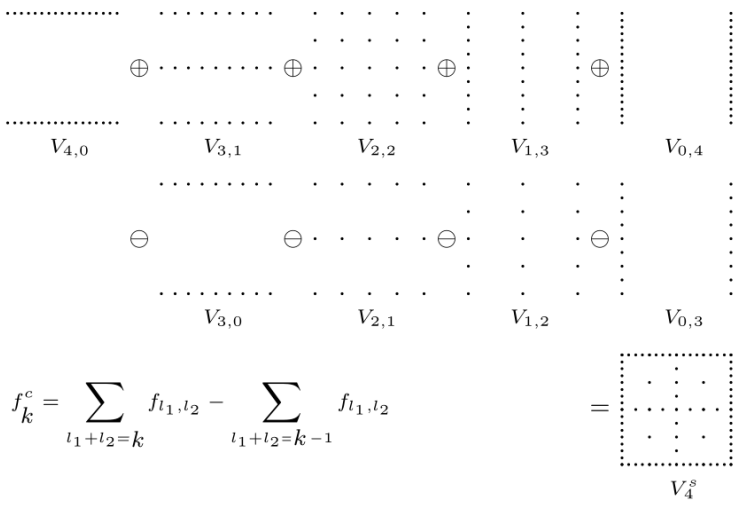


Figure 3.5.: The combination technique for two dimensions with level $k = 4$. The dotted grids in the first row are those belonging to the spaces $V_\mathbf{l}$ with $|\mathbf{l}|_1 = 4$ (layer 1), the grids in the second row fulfill $|\mathbf{l}|_1 = 3$. Combined according to the combination formula, they form the sparse grid space V_4^s, with the corresponding grid in the lower right corner. Source of figure: [Gar13].

Note that in the two-dimensional case, which will often be considered in this thesis, the binomial coefficients $\binom{d-1}{q}$ in the equations simplify to 1. An example for $k = 4$ is shown in Figure 3.5.

It can easily be shown that for function interpolation the combination technique yields the same interpolant as the one obtained with the classical sparse grid approach $f_k^c = f_k^s$, [GSZ92]. For the treatment of PDEs however, the solution obtained with the combination technique does live on the sparse grid space V_k^s, but does not correspond to the solution yielded by the sparse grid technique with hierarchical basis functions. However, if a certain pointwise error expansion holds, both approximation errors are of the same order $\mathcal{O}(h_k^2 \log(h_k^{-1}))$ due to the cancellation of some high-order terms, [GSZ92].
As we will not consider the solution of PDEs in this thesis, we abstain from diving deeper into detail here. Complete and concise explanations and proofs can be found in the surveys [GSZ92, Gar13].

An alternative point of view to the combination technique is to consider the approximations on both the sparse grid space and the partial anisotropic full grid spaces as projections onto those spaces. It can be shown that the approximation performed by the classical combination technique matches the sparse grid approximation exactly, if and only if the projections on the partial spaces, P_{V_i}, commute [HGC07]. As mentioned before, the identity holds true for interpolation problems [GSZ92]. In applications, where this is not the case, numerical instabilities may arise, as has been empirically observed for regression problems by Garcke in his dissertation, [Gar04]. For such cases, an alternative approach has been developed, which is based on the concept of projections: the opticom technique, which will be presented in the next subsection.

3.3. Optimized Combination Technique

While the combination technique has often been successfully applied in the context of partial differential equations, it has been observed to fail in some machine learning applications such as regression, especially when there exist redundancies in the data [Gar04]. Therefore, an alternate combination technique has been developed, that does not choose the coefficients c_i in Equation 3.4 based on the grid structure, but also takes into account the problem itself: the so-called *optimized combination technique (Opticom)* [HGC07].

Basic idea is the projection context, shortly introduced at the end of the past section. For a function $f \in H$, with H some suitable, problem-dependent function space, we consider the projections onto the partial grids, $P_{V^{(i)}} : H \to V^{(i)}$ for all i, and aim to combine those partial projections in a way minimizing the difference to the projection onto the target sparse grid space, $P : H \to V_k^s$. As in the past section we assume a suitable numbering of the partial grids.

To identify the optimal combination coefficients c_i, we aim to minimize the functional $J(c_1, \ldots, c_m) = ||Pf - \sum_{l=1}^{m} c_l P_{V^{(l)}} f||^2$. A simple extension and use of the projectivity property yield

$$J(c_1, \ldots, c_m) = ||Pf||^2 - 2 \sum_{l=1}^{m} c_l \langle Pf, P_{V^{(l)}} f \rangle + \sum_{i=1}^{m} \sum_{j=1}^{m} c_i c_j \langle P_{V^{(i)}} f, P_{V^{(j)}} f \rangle .$$

We note that $\langle Pf, P_{V_l} f \rangle = \langle P_{V_l} f, P_{V_l} f \rangle = ||P_{V_l} f||^2$. Deriving with respect to the c_k and setting the derivative to zero to determine the optimum now yields the system (cf. [HGC07]):

$$\begin{pmatrix} ||P_{V^{(1)}} f||^2 & \ldots & \langle P_{V^{(l)}} f, P_{V^{(m)}} f \rangle \\ \langle P_{V^{(2)}} f, P_{V^{(1)}} f \rangle & \ldots & \langle P_{V^{(2)}} f, P_{V^{(m)}} f \rangle \\ \vdots & \ddots & \vdots \\ \langle P_{V^{(m)}} f, P_{V^{(1)}} f \rangle & \ldots & ||P_{V^{(m)}} f||^2 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} = \begin{pmatrix} ||P_{V^{(1)}} f||^2 \\ ||P_{V^{(2)}} f||^2 \\ \vdots \\ ||P_{V^{(m)}} f||^2 \end{pmatrix} .$$

The solution vector $\mathbf{c}$ of this system contains the so called *optimized combination coefficients*, yielding an optimal combined solution

$$f_k^{oc}(\mathbf{x}) = \sum_{q=0}^{d-1} \sum_{|\mathbf{l}|_1 = k-q} c_{\mathbf{l}} f_{\mathbf{l}}(\mathbf{x}),$$

where the indices have been replaced with the accordingly sorted anisotropic grid levels.

Computationally, as this additional linear equation system is quite small, the solution of it does not come in expensive. The calculation of the scalar products is rather costly, though, as the projected functions do not live in the same spaces $V^{(i)}$. Therefore they first need to be interpolated into a larger space which embeds both relevant partial spaces and in the worst case is one order of magnitude larger than the partial spaces.

It is important to note that the derivation of the optimal coefficients has been developed from a projection point of view, which is suitable in the common case of the Galerkin solution approach to partial differential equations due to the Galerkin orthogonality. In that case, the scalar products and norms are taken with respect to the bilinear form $a(\cdot, \cdot)$ defining the Galerkin equation of the problem: $a(P^{(l)} f, v) = R(v)$ for all $v \in V^{(l)}$.
In many other contexts though, the solutions on the partial grids cannot be interpreted as projections. The approach can nevertheless be applied in these cases by observing that the equation system's right hand side may also considered to be a functional on the right-hand side of said Galerkin formulation, $||P_{V_i} f||^2 = R(P_{V_i} f)$, when choosing the partial solutions as the ansatz functions. Therefore, the opticom approach can also be transferred to other variationally posed problems, when choosing such partial solutions as ansatz functions – problems such as the eigenproblems, which will be considered in this thesis.

The next two sections will show briefly, how both opticom and the classic combination technique have been employed to eigenvalue and machine learning problems, and which challenges have been faced. Our approach in this thesis is tightly connected to such applications.

3.4. Combination Techniques and Eigenvalue Problems

As we will apply both the classic and the optimized combination technique for a certain eigenvalue problem in this thesis, we now present some classes of eigenvalue problems where these methods have been applied successfully.
One of the first applications has been presented by Garcke and Griebel in 2000 for quantum mechanics, namely to calculate the eigenvalues of the Born-Oppenheimer approximation for the Schrödinger equation [GG00]. In that paper as well as in the underlying thesis [Gar98] the combination technique is applied to the three-dimensional Schrödinger equation for hydrogen and the six-dimensional Schrödinger equation for helium under the influence of both magnetic and electric fields.
Using the Born-Oppenheimer approximation [BO27] and some suitable simplifications, one obtains the problem

$$Hu = Eu$$

with H the electronic Hamiltonian operator, E the targeted eigenvalue and u the corresponding wave function to be found [GG00]. In weak form, this problem can be stated as:

$$\langle Hu, \psi \rangle = E \langle u, \psi \rangle \quad \text{for all } \psi. \tag{3.6}$$

This problem could be tackled using conventional finite element methods. Unfortunately those are not applicable due to the high dimensions of the problem, leading to a need for storage and computational power that exceeds the capacities of most modern computers. Therefore, a sparse-grids combination technique with d-linear test and ansatz functions is applied. Classic finite elements eigensolvers (cf. Chapter 2) can be used on the smaller anisotropic grids, and these partial solutions will then be normalized and combined according to the combination formula in Equation 3.5.

However, some challenges arose, and as they will also carry over for the problems treated in this thesis, we will now present them in more detail, as well as present the methods developed to work around them.

3.4.1. Challenges

A first challenge consists in the fact, that in comparison to PDE applications with a single solution, eigenvalue problems have several eigenvalues that need not have a consistent ordering. The eigenvalue λ_i on one of the anisotropic partial grids does not need to (and mostly will not) correspond to λ_i on a grid with another level, or the true i-th eigenvalue of the problem. As we need to identify the eigenfunctions to combine them in a useful way, we have to find the "matching" eigenfunctions on each of the grids. Here, Garcke suggested an algorithm

that orders the grids in a distinct way and for neighboring grids tries to identify "matching" eigenfunctions by projecting the candidate functions onto a common supergrid and match those with the lowest Euclidean distance. For the means of their application, this method proved to work and enable good solutions [Gar98]. However, the authors emphazise, that it does not need to work, especially in the presence of high anisotropies in the grids [GG00].

A second difficulty to face is the possible presence of multiple eigenvalues. In that case, the corresponding eigenfunctions are not unique but may rotate in the eigenspace. This problem can be intercepted by very slightly (in the order of machine precision) modifying the size of the domain in the coordinate direction, which lead to distinct eigenvalues while at the same time not influencing the error order [Gar98].

3.4.2. Opticom for Eigenvalue Problems

Some years later, after the successful application of the combination technique for the Schrödinger equation, Garcke introduced a new algorithm [Gar08] that tackles the same problem, but this time uses Opticom instead of the combination technique.
As mentioned in Section 3.3, the Opticom approach is applicable here even without a projection context, as the core problem 3.6 can be trivially stated as a Galerkin problem. Thus we can use the partial solutions on the anisotropic grids as Galerkin ansatz functions and determine the Opticom coefficients as solutions of said Galerkin problem. Compared to the combination approach, this alternative method was shown to reduce the error results obtained in [GG00] by a factor of 14. While the challenges mentioned for the combination technique remain unchanged (and were tackled with the same methods), Opticom has the additional advantage that a scaling of the eigenfunctions on the partial grids for the combination is not necessary prior to the combination as it is implicitly included in the calculation of the combination coefficients.
So far, further studies to compare these methods for eigenvalue problems have not been conducted. We note, however, that these first results indicate an advantage of Opticom compared to the classic technique.

For the sake of completeness, let us shortly mention another, very different application of Opticom for eigenvalue problems: Kowitz and Hegland [KH14] utilized a variant of the technique to approximate the eigenvectors of the five-dimensional linear gyrokinetic operator. Here however, a reformulation of Opticom was necessary as this problem is neither based on a Galerkin formulation nor symmetric. We will not go into detail on their method, as the problems considered in this thesis are not closely related to the gyrokinetic problem – fortunately in our case, Opticom can be used in its standard form. It should nevertheless be noted, that in first tests the modified Opticom method proved to be very promising in approximating the solutions of the gyrokinetc problem.

3.5. Combination Techniques and Machine Learning

For our work, another successful application of the sparse grids combination techniques is important: the employment in problems of machine learning. In fact, the optimized combination technique was even introduced [Heg02] as an instrument to solve regression problems, a classic task in machine learning.

As some of the ideas and tools applicable in regression will have related concepts in our proposed method, we will now introduce the regression (also known as *data fitting*) problem and its sparse grid handling as described in [GGT01, Gar06, GH09] in more depth.

3.5.1. The Regression Problem

The classic regression problem is the following. Given is a data set of (possibly high dimensional) data points with associated real values, $\mathcal{S} := \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ with $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$ for all $i \in [1\,{:}\,m]$. We assume, that the y_i are function values of an unknown, continuous function $h : \mathbb{R}^d \longrightarrow \mathbb{R}$, $h(\mathbf{x}_i) = y_i$, and now aim to reconstruct this function from the given data set in an optimal fashion.

To that end, we determine a function space V of functions over $\mathbb{R}^d$ and introduce the functional $J : V \longrightarrow \mathbb{R}$,

$$J(f) := \frac{1}{m}\sum_{i=1}^{m}(f(\mathbf{x}_i) - y_i)^2 + \lambda\,||R(f)||_2^2\,.$$

The first term measures the residuals as a description of the closeness of f to the known data, the second term is a regularisation functional $R : V \longrightarrow \mathbb{R}$ that measures smoothness of the function f, mostly chosen to be a differential operator. The inclusion of this term is necessary to obtain a well-posed problem [EHN00]. Finally, the regularization parameter $\lambda \in \mathbb{R}$ balances the two terms.

An optimal choice as the regression target function will now be the minimizer of the functional, $f^* \in V$ with

$$f^* = \arg\min_{f \in V} J(f). \tag{3.7}$$

To apply our grid based method, we reconsider the problem as a Galerkin problem, yielding the equation

$$\frac{1}{m}\sum_{i=1}^{m} f(\mathbf{x}_i)g(\mathbf{x}_i) + \lambda\,\langle R(f), R(g)\rangle_2 = \frac{1}{m}\sum_{i=1}^{m} g(\mathbf{x}_i)y_i, \tag{3.8}$$

which will be fulfilled by the minimizer of J, f^*, for all $g \in V$ [Gar04]. In a next step, we choose the space V as a space where the matching bilinear form a constitutes a scalar product:

$$a(f, g) := \frac{1}{m}\sum_{i=1}^{m} f(\mathbf{x}_i)g(\mathbf{x}_i) + \lambda\,\langle R(f), R(g)\rangle_2\,.$$

In that case, the minimization 3.7 will be an orthogonal projection of the unknown target function h into V [Heg02].
For the sparse-grid-based solution we will choose a finite-dimensional subspace $V_N \subset V$ with a basis $\{\psi_i\}_{i=1}^N$. A function in V_N can thus be represented as $f_N(x) = \sum_{i=1}^N \alpha_i \psi_i$. Using the basis functions as ansatz functions, from Equation 3.8 we can now directly obtain the linear equation system

$$(\mathbf{B}^T\mathbf{B} + \lambda m\mathbf{C})\boldsymbol{\alpha} = \mathbf{B}^T\boldsymbol{\alpha}a, \tag{3.9}$$

with $\mathbf{B} \in \mathbb{R}^{N\times m}$ the evaluation matrix of the basis functions at the data points, $b_{ij} = \psi_i(\mathbf{x}_j)$, and $\mathbf{C} \in \mathbb{R}^{N\times N}$ the matrix containing the L^2-scalar products $c_{ij} = \langle R(\psi_i), R(\psi_j)\rangle_2$. We can then solve this equation for the vector $\boldsymbol{\alpha} \in \mathbb{R}^N$ and thus obtain the optimal solution f_N of Equation 3.7 in V_N.

3.5.2. Regression with the Combination Techniques

For the sparse grid combination technique, we will discretize the space V via a sparse grid space V_n^s. As described in the previous sections, the solution is then combined as the sum $\sum_{i=1}^k c_i f_i$ of the solutions f_i obtained from discretizing the problem on a sequence of k anisotropic subgrids. Note that if we use the gradient operator ∇ as the regularisation operator, the matrix $\mathbf{C}$ will be the standard stiffness matrix on the regular grids.

In his PhD thesis [Gar04], Garcke used the classic combination technique (coefficients c_i as in Equation 3.5) for the regression problem and was able to show empirically that while this technique performed well for some model problems, there were some problems where the combination technique was unstable and even diverged for large data sets and small regularization parameters λ. This certain problem seemed to appear particularly often in applications from data mining, where the data features many redundancies. In an extensive follow-up paper [HGC07] it could be shown that this behavior results from the fact, that the angles between the involved partial grid spaces were not orthogonal and thus the projections on these spaces did not commute. For the exact definition of these terms and the proofs, we refer to the original paper. The Opticom method however does not suffer from these problems, as the coefficients here are chosen adaptively. Further experiments conducted for regression problems with Opticom showed its competitivity compared to other, standard solution approaches [Gar06].

We could thus see that using optimized and problem-dependent combination coefficients has shown promising results before, both for the eigenvalue problems introduced in the last section and the regression application presented in this section. This motivated us to study Opticom for our problem, which roots in both eigenvalue and machine learning applications: the out-of-sample extension of Spectral Embedding and Spectral Clustering methods. These two classes of machine learning problems will be presented in the next chapter, before introducing our algorithm.

4. Spectral Methods in Machine Learning

This chapter offers an overview over two closely related fields of machine learning: *spectral embedding* and *spectral clustering* methods. Given a sample of high-dimensional data as an input, spectral embedding algorithms will provide a similarity-preserving lower-dimensional embedding. Spectral clustering methods will add a second step and perform a clustering of the embedded points in the lower-dimensional space.
After a general introduction to both these fields, one of the algorithms is discussed in more detail: *Laplacian Eigenmaps* by Belkin and Niyogi [BN03], which is the embedding method used for the numerical experiments in this thesis, and the corresponding spectral clustering algorithm by Shi and Malik [SM00].
In the following we will provide a short survey on convergence results for the family of spectral embedding algorithms.
The chapter is then concluded by introducing the problem of *out-of-sample extension* for these methods and a brief presentation of the Nyström approach, a common data-based tool to perform this extension.

4.1. Preliminaries for Spectral Learning

In many areas where we encounter high-dimensional data, this data is not intrinsically high-dimensional, but can be interpreted as the result of a (unknown) function that depends on relatively few variable parameters. One common example is computer vision, where an image of the same scene – usually represented as a large set of pixels associated with color intensity values and thus very high dimensional – changes due to the angle of light incidence or the position of a camera. Our aim now will be to identify a smaller number of such parameters, which are still suited to describe the data via maintaining the local similarities between the data points in a lower dimensional space. The resulting general problem of *dimensionality reduction* can be formulated as follows.

Let $\mathcal{Y} = \{ y_1, \dots, y_m \} \subset \mathbb{R}^d$ be the finite data set. We are looking for a lower-dimensional embedding $f : \mathcal{Y} \longrightarrow \mathbb{R}^p$, $p \ll d$, which preserves local similarities between the high-dimensional data points: If points x and y are close in the original space (with respect to a suitable metric), $f(x)$ and $f(y)$ are expected to be close in the embedding space as well. Larger spatial distances are less important to preserve in the embedding.
If this lower-dimensional representation has been the aim of our approach, e.g. for visualization purposes (target dimensions of $p = 2$ or $p = 3$) or for reasons

of an easier storage of the now reduced data, we will refer to the method as an *embedding method*.

Often however, this dimensionality reduction step is performed as a preprocessing step before applying a clustering algorithm. Such a procedure takes the same high-dimensional data as input but aims at grouping the output data in a meaningful way: Given a number $k \in \mathbb{N}$ of target clusters, a clustering algorithm returns a relation $\mathcal{Y} \to \{1, \ldots, k\}$, labeling each data point with a cluster, so that the similarity within the clusters is maximized while the inter-cluster-similarities are minimized.

Motivated by *graph cuts*, researchers in the late 90's started to develop methods that interpret the data set as nodes of a graph. A pair of data points with high spatial proximity in the original space is connected by an edge with weight according to the similarity between these two points. One can use the spectral properties of the weighted adjacency matrix of the graph to find optimal bipartitionings (or, in a more general context, partitions into k groups). This concept will be introduced in the next section in more detail.
Another concept, that leads to very similar algorithms, originates from probability theory. Using the same notion of the data as a graph, one now formulates the edge weights as transition probabilities of a random walk on the graph. The eigenvalues and eigenvectors of the according transition matrix can then be used to identify and separate different regions of high density in the data.

As both points of view lead to closely related algorithms, we will now subsume the common structure of the resulting family of spectral, graph-based algorithms, before passing to more detailed background analysis:

General Scheme for Spectral Data Analysis (*)

1. For each data point, identify the nearest neighbors according to some distance function or similarity kernel.
2. Construct a graph on the data set with edges between these neighbors and edge weights according to the chosen kernel.
3. Construct a certain matrix that captures information on that graph.
4. Use the first or last (generalized) eigenvalues and eigenvectors of that matrix to produce a lower-dimensional embedding.

For the so-called *spectral embedding methods*, we can now return this embedding and use it for further analysis, such as visualization. If the next step consists of clustering to find a meaningful grouping of the data points, we pass to another, tightly related area of spectral data analysis: *spectral clustering methods*. This class of algorithms will follow up with a clustering step:

5. Use a suitable clustering algorithm to group the embedded data points $f(\mathcal{Y})$.

The classical choice of matrix in step 3 is the so called *graph Laplacian*. Due to

the crucial role this matrix plays for all the algorithms, in the next section this operator and its properties are introduced in more detail.

4.2. Graph Laplacians and their Properties

Essential to all spectral clustering and spectral embedding algorithms is the matrix used to capture the relationships between the data points in some way. This matrix is nearly always referred to as *graph Laplacian*. However, this term is not used consistently in the literature. One (common) definition, and the one that we will use subsequently, is:

Definition 7 (Unnormalized Graph Laplacian)**.**
Let $G = (V, E)$, $|V| = m$ be an undirected graph with non-negative edge weights $w(\{v_i, v_j\}) = w_{ij}$ stored in the symmetric matrix $\mathbf{W} \in \mathbb{R}^{m \times m}$. By $\mathbf{D} \in \mathbb{R}^{m \times m}$ we denote the diagonal degree matrix with entries $d_{ii} = \sum_{j=1}^{m} w_{ij}$.
Then the unnormalized graph Laplacian $\mathbf{L} \in \mathbb{R}^{m \times m}$ *is defined as:*

$$\mathbf{L} = \mathbf{D} - \mathbf{W}$$

Note. *In the following, we will denote the unnormalized graph Laplacian only by* graph Laplacian*. In other works, that term is sometimes used differently, but always based on adjacency and degree matrices and with closely related properties. Other common definitions for this operator are for example the normalized graph Laplacian $\mathbf{L}_{nm} = \mathbf{D}^{-1}\mathbf{W}$, the random-walk graph Laplacian $\mathbf{L}_{rw} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$ and the symmetric graph Laplacian $\mathbf{L}_{sym} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$ [Lux07]. We will use the definition above as we follow the paper by Belkin and Niyogi [BN03]. Note that the eigenpairs of all these matrices are in a one-to-one-relationship and can be obtained from each other via simple transformations.*

Let us shortly consider some properties of the graph Laplacian, as for example compiled in Ulrike von Luxburg's comprehensive survey [Lux07] on spectral clustering:

Theorem 8 (Properties of the graph Laplacian [Lux07])**.**
The matrix $\mathbf{L}$ satisfies the following properties:

1. *For every $\mathbf{f} \in \mathbb{R}^m$ it holds that*

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \frac{1}{2}\sum_{i,j=1}^{m} w_{ij}(f_i - f_j)^2. \tag{4.1}$$

2. *$\mathbf{L}$ is symmetric and positive semi-definite.*
3. *The smallest eigenvalue of $\mathbf{L}$ is 0, the corresponding eigenvector is the constant one vector $\mathbf{1}$.*
4. *$\mathbf{L}$ has m non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_m$.*

Note. *The first equation explains the term graph Laplacian: if we transform the weight function w to a distance d via $w(x, y) = 1/d(x, y)^2$, the formulation*

$\sum_j w_{ij}(f_i - f_j)$ looks like a difference quotient approximating a gradient on the graph, and the right hand side of Equation 4.1 thus can be considered a discrete, graph-based version of the Laplace operator.

The properties of the eigenvectors and eigenvalues of the graph Laplacian have been studied thoroughly since the works of Fiedler [Fie73] and Donath and Hoffman [DH73] in the 60's and 70's and are subject of an own theory: *spectral graph theory*. Comprehensive works on this topic are an extensive survey by Mohar [Moh91] and the textbook by Fan Chung [Chu97].
Two more results are worth noting for our purposes. First, the following property of the eigenvectors to the smallest eigenvalue zero:

Lemma 9 (Identification of connected components)**.**
For an undirected graph G as in Definition 7, the multiplicity k of the smallest eigenvalue 0 of $\mathbf{L}$ equals the number of connected components in the graph, and its respective eigenspace is spanned by the indicator vectors of these components.

In a connected graph, there is thus only one eigenvalue λ_1 that equals 0, and its eigenspace is the space of constant vectors in $\mathbb{R}^m$ (corresponding to the space of constant functions on the vertices).

The second important observation is that, for a connected graph, the eigenvector v_2 belonging to the second eigenvalue λ_2 of $\mathbf{L}$ (also known as the *Fiedler vector*) is heuristically very well suited for graph bipartitioning according to the *normalized cut problem*:

Definition 10 (Normalized cut (NCut) problem)**.**
Given a graph $G = (V, E)$, find a bisection $V = V_1 \dot\cup V_2$, such that

$$\sum_{v \in V_1, w \in V_2} \frac{w(v,w)}{|V_1||V_2|}$$

is minimized.

The NCut-problem itself is known to be NP-hard [SM00]. However, a nearly optimal partitioning can be found by simply grouping the vertices by the sign of the Fiedler vector's entries: $V = V_1 \dot\cup V_2$ with $V_1 = \{\, i \mid v_2(i) < 0 \,\}$ and $V_2 = V \setminus V_1$ [SM00]. This approach solves a relaxed version of the problem; detailed explanation and proof can be found in [Moh91] and [SM00]. The fact that this method yields good results and is remarkably simple forms the main motivation for spectral algorithms in machine learning. In the next section, it will shortly be explained, how one can build the underlying graph from the data points.

4.3. General approach to Spectral Learning

To put things together, we consider the data set $\mathcal{Y} = (y_1, \ldots, y_m) \subset \mathbb{R}^d$ and build an undirected graph $G = (V, E)$ by setting $V = [1\!:\!m]$, where each $v \in V$ represents the corresponding data point y_i. As we expect some structure guiding the

data generation, one of two assumptions can be made: The *manifold assumption*, stating that all the data has been drawn from a manifold $\mathcal{M} \subset \mathbb{R}^d$ with respect to some probability distribution $P_{\mathcal{M}}$ or a *density assumption*, implying that the data is sampled according to some probability density on the whole space.

To enhance this geometric or probabilistic structure of the data, the similarity function is chosen in a way that preserves the intrinsic geometry of the data set by considering spatially local similarities. The common choice for the edge weights is a symmetric, positive definite kernel $K : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$ combined with some cut-off-criterion. Based on a distance function $d : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$, possible criteria to add the edge $\{i, j\}$ to E are:

ε-nearest neighbors The edge $\{i, j\}$ is added, if $d(y_i, y_j) > \varepsilon$.
mutual k-nearest neighbors The edge $\{i, j\}$ is added, if y_j is among the k nearest neighbors of y_i **and** vice versa.
symmetric k-nearest neighbors The edge $\{i, j\}$ is added, if y_j is among the k nearest neighbors of y_i **or** vice versa.

The edge weights $w : E \longrightarrow \mathbb{R}_{\geq 0}$ are then usually determined via the trivial 0-1-kernel or the Gaussian (or heat) kernel K_σ with a parameter σ:

0-1-kernel For $\{i, j\} \in E$: $w(\{i, j\})) = 1$.
Gaussian kernel Fix a bandwidth $\sigma \in \mathbb{R}_{\geq 0}$. For $\{i, j\} \in E$:

$$w(\{i, j\}) = K_\sigma(y_i, y_j) := e^{-\frac{\left|\left|y_i - y_j\right|\right|^2}{\sigma}}.$$

Note. *There is no thorough mathematical theory on which graph setup to use in which cases, or on a suitable choice of graph parameters, for details see [MHL09]. We will introduce some of the established rules of thumb and our own choices in Chapter 6, where we describe our experiments.*

For the graph created in this manner, we will now use the spectral decomposition of the graph Laplacian matrix $\mathbf{L}$ defined in the previous section. The bipartitioning of the graph according to the sign of the second eigenvector of its Laplacian matrix then yields a good bipartitioning for the whole data set, as introduced and demonstrated for image segmentation by Meila and Shi in 2000 [MS01]. To perform partitioning into more than two groups, the eigenvectors $v_3, v_4, \ldots$ corresponding to the next significant eigenvalues $\lambda_3, \lambda_4, \ldots$ can be used for a finer subpartitioning of the two clusters, or more involved clustering rules than a trivial thresholding at zero may be applied.
Instead of clustering, one can also consider the entries of the eigenvectors to form an embedding and use this lower-dimensional embedding for further data processing and analysis.

Machine Learning methods based on this principle are for example the spectral embedding algorithms Diffusion Maps [CL06a], Isomap [TSL00] and Laplacian Eigenmaps [BN03], as well as the famous clustering algorithms by Ng et al. [NJW01],

Shi and Malik [SM00] and Meila and Shi [MS01].

We will now present Belkin and Niyogi's Laplacian Eigenmaps and the closely related Shi-Malik-algorithm for spectral clustering in full detail, before considering convergence results for this general family of algorithms.

4.4. Laplacian Eigenmaps and the Shi-Malik-algorithm

A popular example for a spectral embedding algorithm following the described scheme is Laplacian Eigenmaps, introduced in 2003 by Belkin and Niyogi [BN03]. Originally based rather on intuition, the authors later provided a theoretical background and were able to prove that under some assumptions the graph Laplacian converges to the Laplace-Beltrami operator on the manifold, and that the respective eigenfunctions converge as well [BN08b, BN08a]. These results and further convergence results in general will be outlined in the following section 4.5.

We will now state Belkin and Niyogi's algorithm in detail[1]:

Laplacian Eigenmaps

Input Data set $\mathcal{Y} = \{ y_1, \dots, y_m \} \subset \mathbb{R}^d$

Output Lower-dimensional embedding $\mathcal{Y} \longrightarrow \mathbb{R}^p$, $p \ll d$ fixed.

Step 1: Construct adjacency graph $G = (V, E)$**:** Set $V = [1, \dots, m]$, $E = \emptyset$.
Choose a parameter $k \in \mathbb{N}$. For all pairs $(i, j) \in V \times V$ add the edge $\{i, j\}$ to E, if y_i is among the k nearest neighbors of y_j or vice versa.

Step 2: Set edge weights: Choose a parameter $\sigma \in \mathbb{R}$. For all edges $\{i, j\}$ set entries of the weight matrix $\mathbf{W} \in \mathbb{R}^{m \times m}$ as

$$w_{ij} = e^{-\frac{||y_1 - y_j||^2}{\sigma}}.$$

Construct the diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{m \times m}$ and the unnormalized graph Laplacian $\mathbf{L} \in \mathbb{R}^{m \times m}$ as defined in Definition 7.

Step 3: Eigenmaps: Assume the graph G is connected. Otherwise do the following for each connected component.
Solve the generalized eigenproblem

$$\mathbf{L}f = \lambda \mathbf{D}f,$$

where we can consider $\mathbf{L}$ as an operator on functions defined on V. Let $(\lambda_1, f_1), \dots, (\lambda_m, f_m)$ be the solution pairs to the eigenproblem, ordered according to the size of eigenvalues:

$$0 = \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_m.$$

Return the embedding into p-dimensional Euclidean space:

$$y_i \mapsto (f_1(i), \dots, f_p(i))$$

[1] For simplicity we present just the (standard) version used in this thesis, with k-nearest-neighbors graph and heat kernel-based weights.

Some years before, Shi and Malik [SM00] had presented a closely related clustering algorithm. Their approach was motivated from image segmentation and is one of the first spectral clustering algorithms used in machine learning.
As in the rough overview in the previous section, simply adding one more step to the embedding algorithm yields their clustering algorithm:

Spectral Clustering according to Shi and Malik

Input Data set $\mathcal{Y} = \{ y_1, \dots, y_m \} \subset \mathbb{R}^d$, number k of clusters to construct
Output Set of Clusters $\{ C_i \}_{i=1}^k$.
Steps 1 – 3: As above.
Step 4: Clustering: Fix a parameter k and cluster the points $(f_1(i), \dots, f_p(i))$ calculated in step 3 according to the k-means algorithm[2] into clusters $\tilde{C}_1, \dots, \tilde{C}_k$. Return set of Clusters $C_1, \dots, C_k$ with $C_i = \left\{ y_j \;\middle|\; (f_1(j), \dots, f_p(j)) \in \tilde{C}_i \right\}$.

Note. *In the original paper, Shi and Malik do not solve a generalized eigenproblem, but directly consider the eigenvectors of the normalized graph Laplacian* $\mathbf{L}_{nm} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$*, the matrix that is also used in Diffusion Maps [CL06a]. The embeddings provided are however the same, as follows directly by multiplying the eigenvalue equation* $\mathbf{L}_{nm} f = \lambda f$ *with* $\mathbf{D}$ *from the right.*

We will thus consider these algorithms as a "normalized" approach to spectral learning, which will be important concerning the convergence properties discussed in the next subsection.

4.5. Convergence

As noted in the remark after Theorem 8, the equation $f^T \mathbf{L} f = \sum_{ij} w_{ij}(f_i - f_j)^2$ may be considered as a discrete version of the quadratic form associated to the standard Laplace operator Δ on $\mathbb{R}^n$, which satisfies

$$\langle g, \Delta g \rangle = \int ||\nabla g||^2 .$$

Based on this intuition it makes sense to study the interconnections between the discrete graph Laplacian and continuous Laplace operators on manifolds, especially concerning their spectral properties. Over the last years, several papers have been published, that consider this topic from different angles.

Based on results by Giné and Koltchinskii [GK06], Belkin and Niyogi [BN08a, BN08b] performed convergence studies for Laplacian Eigenmaps, which take a geometric approach based on the manifold assumption. They introduced the *point*

[2] The k-means algorithm is one of the most used clustering algorithms. It minimizes the distance of the points to their respective cluster center (thus minimizing the variance) and was first described by Lloyd in the 1950's [Llo82].

cloud Laplacian, a generalized continuous version of the Laplacian eigenmaps matrix, which depends on the sample size m. They proved that, if the data is sampled uniformly from some underlying manifold $\mathcal{M}$, the eigenfunctions of this operator converge for $m \to \infty$ to the eigenfunctions of the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ on said manifold.

Nadler *et al.* [NLCK06] analyzed the related Diffusion Maps algorithm for spectral embedding and considered the case of a manifold equipped with a non-uniform probability distribution $\rho(x) = \exp(-U(x))$. They were able to show that in this case the eigenvectors of their version of the graph Laplacian, $\mathbf{L}_{\text{rw}}$, converge to the eigenfunctions of the backward Fokker-Planck-operator on $\mathcal{M}$:

$$\mathcal{H}_{\mathcal{M}}\phi = \Delta_{\mathcal{M}}\phi - 2\nabla\phi \cdot \nabla U.$$

This operator is a generalization of the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$, where the term $2\nabla\phi \cdot \nabla U$ represents a drift towards areas with a higher probability density. A similar, but more general result for different classes of Laplacian matrices was proven by Hein *et al.* [HAL06, HAL07]. In their work, they showed that while all of the considered graph Laplacians $\mathbf{L}_{\text{rw}}, \mathbf{L}$ and $\mathbf{L}_{\text{sym}}$ converged pointwise to some continuous limit operator, indeed *only* for $\mathbf{L}_{\text{rw}}$ this limit operator is a (weighted) version of the Laplace-Beltrami operator on the manifold.

An alternative approach in the convergence analysis leaves out the geometric aspects and the manifold assumption and concentrates on a more general case of underlying integral operators, based on a probability density see e.g. [LBB08, RBD10]. In the cited publications, the authors prove that with a fixed weight function w under certain conditions (which remain rather weak for the class of normalized problems containing $\mathbf{L}_{\text{rw}}$ and $\mathbf{L}_{\text{sym}}$) the eigenvectors of the discrete problems converge to the eigenfunctions of certain limit operators on a Hilbert space.

Even though Laplacian Eigenmaps uses the unnormalized graph Laplacian in the problem formulation, the convergence results for the normalized $\mathbf{L}_{\text{rw}}$ are applicable here, as follows from a simple reformulation, see also the note in Section 4.4.

4.6. Out-of-Sample Extension for Spectral Learning Methods

Laplacian Eigenmaps, as all graph-based spectral embedding methods, only provides an embedding for the training data at hand. Thus, a new data point can only be incorporated into the embedding by a new run of the algorithm on the now slightly enlarged data set. Due to the high computational complexity of the eigenvalue solver, alternative methods to perform this so called *out-of-sample extension* without full re-calculation of the eigenvalue problem are being explored, for example in [BPV$^+$04, CL06b, GLMH12, BBH13].

4.6.1. Nyström Extension for Spectral Embedding

The most popular approach for this has been proposed by Bengio et al. [BPV+04] in 2003 and is based on the Nyström method[3]. The extension makes use of the fact that all popular spectral embedding algorithms can be expressed via a normalized version $\tilde{K}$ of the kernel function K employed for the edge weights of the underlying graph – thus, in case of Laplacian Eigenmaps the Gaussian kernel K_t or the nearest-neighbor-0-1-kernel.
The embedding of a new point $\xi \in \mathbb{R}^n$, $\bar{f}_k(\xi)$ with k the index of the corresponding eigenvector, can then be obtained via the formula [BPV+04]:

$$\bar{f}_k(\xi) = \frac{1}{\lambda_k} \sum_{i=0}^{m} f_k(i) \tilde{K}(\xi, y_i),$$

with $\tilde{K}$ a normalized version of the kernel and f_k as defined in the output of the algorithm.
In our case of Laplacian Eigenmaps, such a normalized kernel $\tilde{K}$ can be defined via the kernel K used for the edge weights in the graph,

$$\tilde{K}(a,b) := \frac{1}{m} \frac{K(a,b)}{\sqrt{E_y[K(a,y)] E_{y'}[(K(b,y')]}},$$

where the expectations in the denominator are taken over the complete data set $\mathcal{Y}$ [BPV+04].

Intuitively, the denominator corresponds to the normalization matrix $\mathbf{D}^{-1}$ in the sense that $d_{ii} = \sum_{j=1}^{m} k(y_i, y_j) = m E_{y_i}[k(y_i, y_j)]$. Thus, the kernel $\tilde{K}$ does not exactly correspond to the problem considered in Laplacian Eigenmaps $\mathbf{Lu} = \lambda \mathbf{Du}$, but to the problem $\mathbf{L}_{nm}\mathbf{u} = \lambda \mathbf{u}$ using a different normalized Laplacian $\mathbf{L}_{nm} := \mathbf{D}^{-\frac{1}{2}} \mathbf{W} \mathbf{D}^{-\frac{1}{2}}$. However, the eigenvalues and eigenvectors can, as mentioned before, be obtained from this related formulation via simple transformations (see also [Wei99]).

4.6.2. Nyström Extension for Spectral Clustering

The same approach for out-of-sample-extension can be used for spectral clustering algorithms, as here, too, the computationally expensive part lies in calculating the embedding. After the additional out-of-sample point has been embedded using the Nyström method, a new (cheap) run of the k-means-clustering algorithm can be performed on $f(\mathcal{Y}) \cup \{f(\xi)\}$, which works in $\mathcal{O}(mkp)$, with k the targeted number of clusters and p the dimension of the embedding space. Alternatively, if k-means clustering has already been performed on $f(\mathcal{Y})$ one can perform k-means-out-of-sample extension and assign ξ to the cluster, whose center is the closest to $f(\xi)$.

[3] Originally the Nyström method stems from problems with large matrix computations and consists of generating a lower-rank approximation to a large matrix by subsampling from its columns.

4.6.3. Grid-based Out-of-Sample Extension

This method has been proven to work very well and is a standard choice for out-of-sample-extension of spectral data analysis methods. However, it is not very natural. A more intuitive approach to the extension problem would be to use spectral embedding to not only learn a discrete embedding $f : \mathcal{Y} \to \mathbb{R}^p$, but a full embedding function $\bar{f} : \mathbb{R}^n \to \mathbb{R}^p$. As Belkin and Niyogi were able to prove, the eigenvectors their algorithms provides will converge (as the number of data points – interpreted as samples drawn from the underlying manifold $\mathcal{M}$ – grows) to the eigenfunctions of the Laplace-Beltrami operator $\Delta_{\mathcal{M}}$ on that manifold. Therefore, learning a continuous embedding function should be feasible. However, a "useful" finite-dimensional representation of this function would need to be grid-based, and as we are working in high dimensions the curse of dimensionality would hold.

Based on a proposal by Peherstorfer, Pflüger and Bungartz [PPB11], the aim of this thesis is to present, adapt and analyze a method for this grid-based out-of-sample-extension using the sparse grid approach introduced in Chapter 3 and thus containing the inherent curse of dimensionality up to some point.
The next chapter will present the algorithms needed to combine the spectral techniques presented in this chapter with the sparse grid method and the challenges we face in this approach.

5. Out-of-Sample-Extension using Opticom

In this chapter, we will establish the central concept of this thesis: a grid-based out-of-sample extension for Laplacian Eigenmaps using the optimized sparse grid combination technique.

First, we will present the general approach for a sparse-grids-based extension as introduced by Peherstorfer, Pflüger and Bungartz in 2011 [PPB11]. While they applied hierarchical sparse grid basis functions, we will instead use the optimized combination technique. The proposed algorithm will be presented in Section 5.2 in detail.
Some of the challenges mentioned in the previous chapters will carry over to this application, others can be effectively countered. We will give an overview and a short discussion of these aspects in Section 5.3. In the following, we will provide an alternative point of view on the partial problems considered in the algorithm.

5.1. Sparse-Grid-based Out-of-Sample Extension

The idea of using sparse grids in the context of out-of-sample extensions of Laplacian Eigenmaps was first published in a paper by Peherstorfer, Pflüger and Bungartz in 2011 [PPB11]. Instead of using an extension based on the data points (as the Nyström method, cf. Section 4.6), their idea is the following:
It is known that under certain conditions for an increasing number of samples $\mathcal{Y} = \{ y_1, \dots, y_M \}$, $M \to \infty$ drawn from a manifold $\mathcal{M} \subset \mathbb{R}^d$, the eigenvectors $\mathbf{u}_i \in \mathbb{R}^M$ recovered by Laplacian Eigenmaps will converge to the eigenfunctions u_i of the Laplace Beltrami operator $\Delta_{\mathcal{M}}$ on that manifold [BN08a], cf. Section 4.5. Thus, it may be possible to try and recover a discretized version of these functions $u_{h,i}$, and for out-of-sample points $\xi \in \mathbb{R}^d$ simply evaluate these functions at ξ to provide a p-dimensional embedding $\{u_{h,i}(\xi)\}_{i=2}^{p+1}$.

A naive approach to perform such a calculation on a grid would fail, as the problems are often high-dimensional and grid methods would suffer from the curse of dimensionality. To prevent this, the authors suggest to use the sparse grid technique, as presented in Chapter 3.
We denote the discrete sparse grid space by V^s and its hierarchical basis by $\Phi = \{\phi_i\}_{i=1}^N$. For every function $f \in V^s$ we now have a representation

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

with a coefficient vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_N)$. We set up a new discrete problem as follows: We consider the i-th embedding vector $\mathbf{v}_i \in \mathbb{R}^M$ resulting from the classic Laplacian eigenmaps as the vector $\tilde{\mathbf{f}}_i \in \mathbb{R}^M$ containing function evaluations of f_i at the data points, $\tilde{\mathbf{f}}_i = (f_i(y_k))_{k=1}^M$ and reformulate the problem to:
Find $\lambda \in \mathbb{R}, \tilde{\mathbf{f}} \in \mathbb{R}^N$ such that

$$\mathbf{L}\tilde{\mathbf{f}} = \lambda \mathbf{D}\tilde{\mathbf{f}}$$

A variational approach, using the representation of f in the sparse grids formulation leads to:
Find $\lambda \in \mathbb{R}$, coefficients $\boldsymbol{\alpha} \in \mathbb{R}^N$, such that

$$\mathbf{B}^T\mathbf{L}\mathbf{B}\boldsymbol{\alpha} = \lambda \mathbf{B}^T\mathbf{D}\mathbf{B}\boldsymbol{\alpha},$$

with $\mathbf{B} \in \mathbb{R}^{M \times N}$ being the evaluation matrix of the basis Φ at the data set $\mathcal{Y}$, $b_{ij} = \phi_j(y_i)$.
To provide a better solvabilily, Peherstorfer et al. add a regularization term,

$$(\gamma \mathbf{C} + \mathbf{B}^T\mathbf{L}\mathbf{B})\boldsymbol{\alpha} = \lambda \mathbf{B}^T\mathbf{D}\mathbf{B}\boldsymbol{\alpha},$$

where $\mathbf{C} \in \mathbb{R}^{N \times N}$ may for example be the identity matrix, and $\gamma \in \mathbb{R}$ is a suitably chosen regularization parameter.
The eigenvectors $\boldsymbol{\alpha}_i$ obtained from the solution of this problem define our target functions $f_i \in V^s$. Note that, analogously to the previous chapter, $\boldsymbol{\alpha}_i$ defines the eigenvector associated to the eigenvalue λ_i, and the eigenvalues are ordered increasingly, $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_N$.
An embedding for an out-of-sample-point $\xi \in \mathbb{R}^d$ into $\mathbb{R}^p$ can then be provided by $\xi \mapsto (f_i(\xi))_{i=2}^{p+1}$. Note that we leave out the function f_1 corresponding to the smallest eigenvalue λ_1, as we know that $\lambda_1 = 0$ and the associated eigenvector is constant and thus contains no information.

The authors implemented and tested their approach successfully on synthetic and real world data, using the hierarchical sparse grid basis functions. The aim of this thesis is now, to implement, test and analyze their proposed method with the Opticom technique instead of the – harder to handle – hierarchical basis functions. This idea has already worked out well for regression problems (cf. Section 3.5.2) and we will now consider if it may also prove to be an alternative for the out-of-sample-extension of spectral embedding.

5.2. Algorithm

5.2.1. Core Algorithm

Before considering the theoretical details and challenges, we will show how the method presented in the previous section can be modified, formulated and implemented to provide an Opticom-based out-of-sample-extension for Laplacian Eigenmaps.
To this end, we will apply the method described above to a sequence of anisotropic,

Algorithm 1: OOS-extension of Laplacian Eigenmaps with Opticom

Input: data set $\mathcal{Y} = \{ y_1, \ldots, y_M \} \subset \Omega \subset \mathbb{R}^d$, *sparse grid parameters*: level $L \in \mathbb{N}$, margin $a_{\text{mar}} \in \mathbb{N}$, *graph setup parameters*: number of nearest neighbors $n_{\text{nn}} \in \mathbb{N}$, kernel bandwidth $\sigma \in \mathbb{R}$, *regularization parameter* $\gamma \in \mathbb{R}$

Output: embedding function $f : \Omega \longrightarrow \mathbb{R}$

read input data $\mathcal{Y}$;
build similarity graph on data using graph parameters n_{nn}, σ (cf. Section 4.4);
calculate graph Laplacian $\mathbf{L} \in \mathbb{R}^{M \times M}$ and diagonal degree matrix $\mathbf{D} \in \mathbb{R}^{M \times M}$ (cf. Section 4.4);
determine $K \in \mathbb{N}$, number of partial grids to be considered;
initialize opticom matrices $\bar{\mathbf{L}} \in \mathbb{R}^{K \times K}$, $\bar{\mathbf{D}} \in \mathbb{R}^{K \times K}$;
initialize level counter $k := 1$;
for *levels* $\mathbf{l}$ *with* $\sum_i l_i = L$ *and* $\forall i : l_i \leq a_{mar}$ **do**
  setup regular anisotropic grid of level $\mathbf{l}$ on Ω, using d-linear basis functions $\mathcal{B}^{(k)} = \left\{ \phi_1^{(k)}, \ldots, \phi_{N_k}^{(k)} \right\}$;
  calculate grid stiffness matrix $\mathbf{C}^{(k)} \in \mathbb{R}^{N_k \times N_k}$, $(c^{(k)})_{ij} = \int \nabla \phi_i \nabla \phi_j$;
  calculate grid-Laplacian $\mathbf{L}^{(k)} = (\mathbf{B}^{(k)})^T \mathbf{L} \mathbf{B}^{(k)} \in \mathbb{R}^{N_k \times N_k}$ and grid-degree matrix $\mathbf{D}^{(k)} = (\mathbf{B}^{(k)})^T \mathbf{D} \mathbf{B}^{(k)} \in \mathbb{R}^{N_k \times N_k}$, where $b_{ij}^{(k)} = \phi_j^{(k)}(y_i)$;
  solve the generalized eigenproblem

$$(\mathbf{L}^{(k)} + \gamma \mathbf{C}^{(k)}) \boldsymbol{\alpha}^{(k)} = \lambda^{(k)} \mathbf{D}^{(k)} \boldsymbol{\alpha}^{(k)}$$

  and use generalized eigenvector $\boldsymbol{\alpha}_2^{(k)}$ corresponding to second smallest eigenvalue $\lambda_2^{(k)}$ to obtain and store partial solution $f^{(k)} = \sum_{i=1}^{N_k} \alpha_{2,i}^{(k)} \phi_i^{(k)}$;
  for $\tilde{k} = 1$ **to** k **do**
    determine level $\mathbf{l}^*$ of smallest grid containing grids k and $\tilde{k}$;
    setup regular anisotropic grid of level $\mathbf{l}^*$ and corresponding stiffness matrix $\mathbf{C}^{(k^*)}$, grid-Laplacian $\mathbf{L}^{(k^*)}$ and grid-degree matrix $\mathbf{D}^{(k^*)}$ as above;
    project functions $f^{(k)}$, $f^{(\tilde{k})}$ onto functions $\overline{f^{(k)}}$, $\overline{f^{(\tilde{k})}}$ on grid $k*$;
    set $\bar{l}_{ij} = \left\langle \overline{f^{(k)}}, (\mathbf{L}^{(k^*)} + \gamma \mathbf{C}^{(k^*)}) \overline{f^{(\tilde{k})}} \right\rangle$;
    set $\bar{d}_{ij} = \left\langle \overline{f^{(k)}}, \mathbf{D}^{(k^*)} \overline{f^{(\tilde{k})}} \right\rangle$;
  end
  k = k+1;
end
solve opticom generalized eigenvector problem

$$\bar{\mathbf{L}} \mathbf{u} = \lambda \bar{\mathbf{D}} \mathbf{u}$$

for eigenvector $\mathbf{u}_1$ according to smallest eigenvector λ_1 and store in coefficient vector $\mathbf{c}$;
return embedding function $f := \sum_{k=1}^{K} c_k f^{(k)}$

but regular grids (instead of using the hierarchical basis of one sparse grid), and combine these solutions in a useful manner determined by the additional Opticom step (cf. Section 3.3). In contrast to the previously presented method, we will need to be a bit more careful with the regularization step, in order to deal with the anisotropies on the partial grids. Where Peherstorfer et al. could apply the identity matrix, we need to use a matrix taking into account the shape of our, potentially highly anisotropic, grids instead. In the presented approach, this role is taken by the stiffness matrix, however also a weighted version of the mass matrix (representing the identity operator) is be a possible candidate.

The proposed core algorithm without post-processing is stated in detail in Algorithm 1. For simplicity, we give a full version of the algorithm only for a one-dimensional embedding. The necessary extensions (and decisions on implementation and design) for a more-dimensional embedding will be presented and discussed in the subsequent sections, especially in Section 5.4.

Note that we used only the top layer of anisotropic grids here; if we were to use the standard combination technique we would need d layers. In this case, we replace the outer `for`-loop in the algorithm by the two nested loops "**for** q in [0:d] **do**" and "**for** levels $\mathbf{l}$ with $\sum_i l_i = L - q$ and $\forall i : l_i \leq a_{\text{mar}}$ **do**".

For the following analysis, we will partition the algorithm into the following core steps:

Data management Read data, build graph and matrices.

Step 1: Compute partial solutions Solve grid-dependent eigenproblems on K regular anisotropic grids.

Step 2: Optimized Combination Solve an eigenproblem set up from scalar products of the partial solution.

Post-processing and function evaluation Evaluate the obtained Opticom solution at out-of-sample points $\xi \in \Omega$. If applicable, perform clustering on the embedding $f(\mathcal{Y} \cup \{\, \xi \,\})$ (cf. Subsection 5.2.2)

5.2.2. Postprocessing

Depending on the application context, different aims are pursued by embedding methods. Sometimes the goal is to allow for a lower-dimensional representation of originally high-dimensional data, that can then be visualized in two or three dimensions and adapted to human perception. Often, the embedding is indeed only a preprocessing step before clustering is applied on the now lower dimensional space.

Analogously; the reasons for out-of-sample-extensions vary: In some situationS not all data points are known at calculation time, and one wishes to quickly analyse new data as it becomes available. In others, training of the machine learning algorithm on the data set grows so quickly in M, that one splits the data into a training set and one evaluation set that is processed with the newly learned embedding function. In the following, we will focus on the latter application.

For the post-processing step we have different implementation options to achieve the respective aim as efficiently as possible.

Embedding

As our aim is to speed up the learning process and thus all out-of-sample data is given at once, we now present the short algorithm for batch evaluation of the new data set in Algorithm 2. Note that we assume that the learned target function $f : \Omega \to \mathbb{R}$ is stored internally via its coefficient vector $\mathbf{c} \in \mathbb{R}^K$ as well as the number of partial grids, $K \in \mathbb{N}$.

Algorithm 2: Batch evaluation of out-of-sample points for embedding

Input: out-of-sample data $\mathcal{S} = \{ \xi_1, \ldots, \xi_M \} \subset \Omega \subset \mathbb{R}^d$
Output: out-of-sample embedding $f(\mathcal{S})$

initialize output vector $v \in \mathbb{R}^M$;
for $k < K$ **do**
  load partial grid k;
  load grid coefficient vector $\boldsymbol{\alpha}^{(k)}$;
  for $i < M$ **do**
    $v_i = v_i + \sum_j \alpha_j^{(k)} \phi_j^{(k)}(\xi_i)$;
  end
end
return $f(\mathcal{S}) = \{v_1, \ldots, v_M\}$

As it is possible to evaluate the basis functions of the partial grids very efficiently, this batch evaluation is very fast, which is also confirmed by experiments in the upcoming Chapter 6.

Clustering

If we only wish to obtain a simple bi-clustering into clusters $\{0, 1\}$, it is sufficient to consider the one-dimensional embedding function and threshold it at zero. For the new data point $\xi \in \Omega$ the cluster assignment is then defined as

$$c(\xi) = \begin{cases} 0, & \text{if } f(\xi) < 0, \\ 1, & \text{if } f(\xi) \geq 0, \end{cases}$$

with $f : \Omega \to \mathbb{R}$ the embedding function learned in Algorithm 1. Especially for embedding dimensions $p > 1$ (for the extension of Algorithm 1 to this case see Section 5.4), one may wish to use a more sophisticated clustering than simple thresholding. In that case, one needs to decide depending on the complexity of the clustering algorithm and the size of the out-of-sample-data set, if a full run of the clustering algorithm on training and out-of-sample set is necessary. Otherwise, a out-of-sample-extension of the clustering algorithm itself may be used.

5.2.3. Complexity

There are two main drivers of the complexity in our approach: the solutions of the grid based eigenvalue problems on the $\mathcal{O}(dn^{d-1})$ partial grids, and for every partial grid the loop over all previously considered grids with the projections on a common grid and the calculation of the scalar products $\left\langle \overline{f^{(k)}}, (\mathbf{L}^{(k^*)} + \gamma \mathbf{C}^{(k^*)}) \overline{f^{(\tilde{k})}} \right\rangle$ on these common grids. The preprocessing steps, as well as the reasonably small final eigenproblem, are negligible in comparison.

5.3. Challenges

To combine the spectral embedding approach with Opticom gives rise to several challenges, some of these known from the application of the combination techniques to eigenvalue problems, cf. Section 3.4, others arising due to the origins of the operator in question.

5.3.1. Sorting of the Eigenvalues

This problem has already been described shortly in Section 3.4: Depending on the different discretizations on the partial grids, different features of the data are enhanced, and thus the eigenvalues capturing these features will vary from grid to grid. Especially for the less significant eigenvalues, that this will also influence the ordering of the eigenvalues and the effect of *eigenvalue crossings* will occur. For the combination however we will need to combine the "matching" eigenvalues instead of those at the same position. In [Gar98] Garcke suggests (for the classic combination technique) to, for each eigenfunction, loop over all possible matching functions on other grids, project them onto the coarsest grid containing both function spaces and compute the L^2-distance on that grid. The "closest" functions in this sense are then considered to be approximations to the same continuous eigenfunction and will be combined in the combination step. As this heuristic approach is not guaranteed to work [GG00, Gar98], other possible solutions may be evaluated.
For the problems considered in this thesis, we assume the first non-zero eigenvalue λ_1 to be so dominant in the problem that it will be recovered on all grids in the same extreme position. Concerning the further eigenvectors, which are needed for a higher-dimensional embedding, we will follow an alternative approach, introduced in Section 5.4, where we collect all eigenvectors on the respective spaces into one, larger Opticom system and use the existing orthogonalities between the eigenfunctions.
Alternatively, the intra-grid alignment of the eigenvectors might be modeled similar to the projection methods used in model order reduction. An example application can be found in [AF11].

5.3.2. Multiple Eigenvalues

If an eigenvalue has a multiplicity larger than one, the corresponding eigenvectors are not uniquely determined (up to multiplication with -1), but may rotate in the eigenspace they span. Thus in this case we would need to perform some (computationally very expensive) inter-grid alignment of these eigenvectors. For the Born-Oppenheimer-problem Garcke [GG00] successfully evaded this problem by slightly perturbing the spaces in the different coordinate directions.
Luckily, this problem will is unlikely to occur in our case, where the eigenvalue problem stems from a graph based problem on the data, which has been drawn randomly without any symmetries. If the graph is connected, we know that exactly one of the eigenvalues, namely λ_1, is equal to zero, and due to the random structure of the graph the other eigenvalues will almost surely not be multiple.

5.3.3. Anisotropies of Grids

It is known from perturbation theory that the stability of eigenvalues holds only locally [Kat76]. The inclusion of highly anisotropic grids might thus be unfeasible for eigenvalue problems and hinder any possible convergence. For the small range of grid levels and dimensions considered in the course of this thesis, we will try to counter potentially negative effects of highly anisotropic grids via the margin parameter $a_{\text{mar}} \in \mathbb{N}$. For future, larger setups an automatic choice of a_{mar} will need to be considered.

5.4. Variants for two-dimensional Embeddings

Most of the above-mentioned concepts, as well as the presented algorithm, treat only the one-dimensional embedding provided by the second eigenvector. As indicated, in case of a graph Laplacian, this vector (also known as Fiedler vector) indeed carries most of the information on the graph. In general, to obtain a higher-dimensional embedding (and, at the same time, a more sophisticated clustering), the next $p-1$ eigenvalues must be considered for the embedding analogously.

In the classic Laplacian eigenmaps and in the method on only one grid (e.g. the hierarchical organized sparse grid considered in the approrach by [PPB11]), these eigenvectors can be obtained directly from the method by computing, storing and processing more generalized eigenvectors of the matrix $\mathbf{L}$ or $\gamma\mathbf{C}+\mathbf{B}^T\mathbf{L}\mathbf{B}$. For our approach, however, one needs to decide on the design of the algorithm. For each partial grid (step 1) we can easily determine as many eigenvectors as we want – but how do we combine them in the Opticom step (step 2)?
Trivially, two options come to mind:

1. To determine the i-th component of the embedding, use the respective $i+1$st partial solution functions provided by the partial grids and for every component set up a separate Opticom system, from which the first eigenvector will be used (in short: several Opticom systems, repeat step 2).

2. Use all $k \cdot p$ partial solution functions obtained from the partial grids, $f_2^{(1)}, \ldots, f_{p+1}^{(1)}, f_2^{(2)}, \ldots, f_{p+1}^{(2)}, \ldots, f_2^{(k)}, \ldots f_{p+1}^{(k)}$ as ansatz functions to form a large Opiticom system, from which we use the first p eigenvectors in the next step (in short: one Opticom system, enlarge step 2).

Note that only the first approach can be used analogously to the classic Combination technique.
In the next two subsections, we will present the theoretical advantages and drawbacks; numerical experiments and results for both approaches will be shown in chapter 6.

5.4.1. More Opticom Systems

An obvious drawback of this approach is the necessary sorting of the eigenvalues, as described in the previous section. A heuristic algorihm to cope with this problem does exist, as introduced in the previous subsection. However, the overhead of projecting the solutions onto the finer, common space is large, and there are no results on the expected quality of this processing. Especially the choice of the L^2-difference as a matching criterion might not be optimal.
A possible alternative could be to consider the bilinar forms defined by the matrix $\mathbf{A} := \gamma\mathbf{C} + \mathbf{B}^T\mathbf{L}\mathbf{B}$ on the common grid. As the originial spaces are embedded in the common grid space, it is known from classic spectral theory that the eigenvectors caluculated on the partial grids will also be $\mathbf{A}$-orthogonal on the common grid. It follows, that pairs of partial solutions approximating the same continuous eigenfunction will have scalar products close to one, while those pairs of eigenfunctions on the partial grids approximating different – and thus orthogonal with respect to the continuous operator – eigenfunctions will exhibit scalar products close to zero.

5.4.2. Larger Opticom System

The option, where we just use all $k \cdot p$ solution functions from the partial grids is a lot simpler to implement, as there is no overhead from comparing the functions across different partial grids. On the other hand, the complexity is – analogously to above – dominated by the necessary projections on the subspaces, where we calculate the bilinear forms. Here, too, we encounter the $\mathbf{A}$- and $\mathbf{M}$- orthogonalites, where $\mathbf{M} := \mathbf{B}^T\mathbf{D}\mathbf{B}$ is the matrix on the right-hand side. This leads to a very clear distinction between the approximations of different eigenfunctions on the partial grids. Thus, the optimal combination coefficients for target eigenfunction u_i indeed will nearly only give weight to partial solution functions that are "matching", without the necessity of a prior sorting.

5.5. Graph-based Derivation

A possible point of view towards the eigenvalue problem $(\gamma\mathbf{C} + \mathbf{B}^T\mathbf{L}\mathbf{B})\boldsymbol{\alpha} = \lambda\mathbf{B}^T\mathbf{L}\mathbf{B}\boldsymbol{\alpha}$ is to consider it as a new sort of graph problem on a grid.

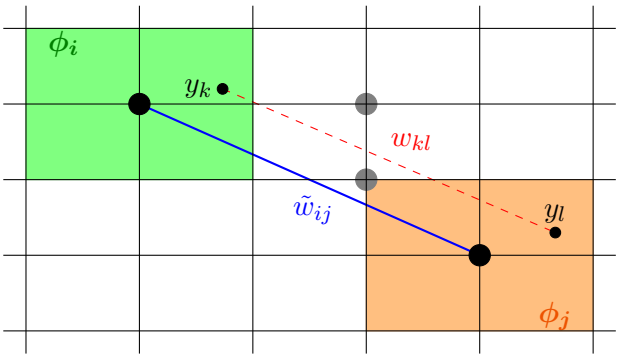


Figure 5.1.: Calculation of the edge weight $\tilde{w}_{ij}$, describing data-based "similarity" between the nodal basis functions ϕ_i (support marked in green) and ϕ_j (support marked in orange) on the grid graph: The edge between the data points y_k and y_l on the data graph (red dashed line) adds its weight, which is then again weighted by the respective function values at the data points: $\tilde{w}_{ij} = \phi_i(y_k)\phi_j(y_l)w_{kl}$. Note that on the grid graph there would be no edge between the gray nodes, as none of their respective nodal basis functions supports a data point.

Whereas the graph $G = (V, E)$ defining the graph Laplacian matrix $\mathbf{L}$ was defined over the data set $\mathcal{Y} = \{ y_1, \dots y_M \}$, we now consider another graph, $\tilde{G} = (\tilde{V}, \tilde{E})$, which is defined over the nodes of the anisotropic grid. For distinction, we will denote the former by *data graph* and the latter by *grid graph*. We define the grid graph as follows: the vertices are formed by the indices of the grid's basis functions, $\tilde{V} = \{ i \mid \phi_i \in \Phi \}$ and $\{i, j\} \in E$, if there is a positive edge weight, according to the weight function $\tilde{w} : \tilde{V} \times \tilde{V} \to \mathbb{R}$,

$$\tilde{w}(i, j) = \sum_{k,l=1}^{M} \phi_i(y_l)\phi_j(y_k)w_{kl},$$

where w_{kl} is the edge weight assigned to $\{k, l\}$ (or $\{y_k, y_l\}$) in the data graph. This setup is depicted in Figure 5.1.
As this weight function is non-negative, all the results on graph Laplacians from Section 4.2 are also applicable for the *grid Laplacian* $\tilde{\mathbf{L}} = \tilde{\mathbf{D}} - \tilde{\mathbf{W}} = \mathbf{B}^T\mathbf{D}\mathbf{B} - \mathbf{B}^T\mathbf{W}\mathbf{B} = \mathbf{B}^T\mathbf{L}\mathbf{B}$.

In this setup, the regularization can be explained by the addition of information on interconnections between the grid points. In Figure 5.1 for example, the nodes marked by the gray circles would not be connected in the graph, even though they are neighboring, and intuitively also affected by the similarity between the data points y_k and y_l. Adding some weights via a grid-based matrix such as mass or stiffness matrix would at least strengthen the idea of a similarity of neighboring grid points and the transfer of similarity not only via data points, but also via a sort of "diffusion" of similarity across the domain.

Note. *While in the analysis of Laplacian eigenmaps the solution of our generalized*

problem $\mathbf{Lv} = \lambda\mathbf{Dv}$ *corresponded to that of* $\mathbf{L}_{rw}\mathbf{v} = \lambda\mathbf{v}$*, this does not hold any more for the problems on the grid, as the right-hand-side matrix* $\tilde{\mathbf{D}} = \mathbf{B}^T\mathbf{DB}$ *is not invertible.*

5.6. Attempt of a Numerical Derivation

For a technical analysis of our approach, we may want to consider our problem setup as the empirical and functionally discretized version of an unknown continuous operator. This point of view allows us to reconsider our problem in the light of the variational numerical methods for the solution of eigenvalue problems, introduced in Chapter 2. In the following, we will provide a heuristic approach to this reformulation.

Going backwards, we start with some unknown operator $\mathcal{L}$ and its associated bilinear form a. A variational approach (cf. Chapter 2) would start from the eigenvalue equation

$$a(u,v) = \int_\Omega \mathcal{L}uv \,\mathrm{d}x = \lambda \int_\Omega \mathcal{D}uv \,\mathrm{d}x = \lambda b(u,v) \quad \text{for all } v \in V, \tag{5.1}$$

where we choose the necessary spaces as $V = H^1(\Omega)$ and $H = L^2(\Omega)$. The operator $\mathcal{L}$ is known only partially. For a start, we heuristically assume it is an integral operator on a manifold $\mathcal{M}$ equipped with a probability distribution ν and that it can be written in the form

$$\mathcal{L}u(x) = \int_{\mathcal{M}} k(x,s)u(s) \,\mathrm{d}\nu(s).$$

The underlying manifold and the density function are yet unknown, but can be empirically estimated by the sample $\mathcal{Y}$ that we assume has been drawn accordingly. Thus, we will model $\mathcal{L}$ by an empirical operator $\mathcal{L}^* : V \to V$,

$$\mathcal{L}^*u(x) = \frac{1}{m}\sum_{k=1}^{m} u(y_k)k(x,y_k).$$

Performing a Monte Carlo integration over the grid points, we can thus express the LHS integral as the sum

$$\int_\Omega \mathcal{L}uv \,\mathrm{d}x \approx \int_\Omega \mathcal{L}^*uv \,\mathrm{d}x \approx \frac{1}{m^2}\sum_{k,l=1}^{m} u(y_k)v(y_l)k(y_k,y_l).$$

For $\mathcal{D}$ on the other hand we make the assumption that it is a simple scaling operator, weighing $u(x)$ with the expectation value of $k(x,\cdot)$ taken with respect to the data-generating density function ν:

$$\mathcal{D}u(x) = u(x)\int_\Omega k(x,s) \,\mathrm{d}\nu(s).$$

This can also be approximated with the empirical density given by the sample $\mathcal{Y}$:

$$\mathcal{D}u(x) \approx \mathcal{D}^* u(x) = u(x)\frac{1}{m}\sum_{k=1}^{m} k(x, y_k).$$

As this scaling does not depend on u, we can define the weights using the function $d : \mathbb{R}^d \to \mathbb{R}$, $d(x) = \frac{1}{m}\sum_{k=1}^{m} k(x, y_k)$, representing the empirical expectation of $k(x, \cdot)$. As before, we also use a Monte-Carlo method to calculate the right-hand-side integral of Equation 5.1 and obtain

$$\int_\Omega \mathcal{D}uv \,\mathrm{d}x \approx \frac{1}{m}\sum_{j=1}^{m} u(y_j)d(y_j)v(y_j).$$

Using a grid-based finite element space $V_h \subset V$ with basis $\Phi = \{\phi_i\}_{i=1}^N$, we can then use a standard Galerkin method by approximating u in V_h by a discretized $u_h = \sum_{j=1}^N \alpha_j \phi_j$ and obtain the finite element problem:
Find $\boldsymbol{\alpha} \in \mathbb{R}^N$ such that

$$\sum_{i=1}^{N} \alpha_i \sum_{k,l=1}^{m} \phi_i(y_l)\phi_j(y_k)k(y_l, y_k) = \lambda \sum_{i=1}^{N} \alpha_i \sum_{k=1}^{m} \phi_i(y_k)\phi_j(y_k)d(y_k) \quad \text{for all } \phi_j \in \Phi.$$

This problem may be represented in matrix-vector notation as

$$\mathbf{B}^T\mathbf{L}\mathbf{B}\boldsymbol{\alpha} = \lambda \mathbf{B}^T\mathbf{D}\mathbf{B}\boldsymbol{\alpha}, \tag{5.2}$$

with evaluation matrices $\mathbf{B} \in \mathbb{R}^{m\times N}$, $b_{ij} = \phi_j(y_i)$.

So, in conclusion we used discretization in the following steps:

- Discretization of unknown operator $\mathcal{L}$ via known empirical distribution (data points),
- Monte-Carlo approximation of both integrals,
- Galerkin finite element approach for eigenvalue problem using the same basis as ansatz and test functions.

Regularization can be included in this derivation by describing the unknown target operator as the combination of the standard Laplacian operator and an (unknown) integral operator, $\mathcal{L} = \gamma\Delta + \tilde{\mathcal{L}}$. This can be justified not only by the technical need for regularization, but also by our assumption, that the graph Laplacian corresponds to a discrete version of the Laplace operator.

As the Laplacian is known, we can then rewrite the integral $\int \gamma\Delta uv \,\mathrm{d}x$ into $\gamma\int \nabla u\nabla v \,\mathrm{d}x$ without the need for a data-based reformulation as it was performed for $\tilde{\mathcal{L}}$. The discretization takes place only in the Galerkin step, adding an additional term to Equation 5.2:

$$\gamma m^2\mathbf{C} + \mathbf{B}^T\mathbf{L}\mathbf{B}\boldsymbol{\alpha} = \lambda \mathbf{B}^T\mathbf{D}\mathbf{B}\boldsymbol{\alpha}, \tag{5.3}$$

where $\mathbf{C} \in \mathbb{R}^{N \times N}$ is the standard finite element stiffness matrix, $c_{ij} = \int \nabla \phi_i \nabla \phi_j$.

We emphasize that this derivation is only heuristic. While it helps to provide a better intuition of the discretization step, the operator $\mathcal{L}$ as we chose it, does not resemble the Laplace Beltrami operator Δ, but rather its inverse, Δ^{-1}. Additionally, the matrix $\mathbf{L}$ as known from Laplacian Eigenmaps does not exactly correspond to a kernel k, in the way we used it to derive Equation 5.2.

Still, this heuristic attempt is useful to understand both main steps of our algorithm as a Galerkin formulation. We know that on the partial grids our basis $\Phi = \{\phi_i\}_{i=1}^{N}$ is a standard nodal finite element basis. However, also in the second step, the Opticom step, we can interpret the space spanned by the partial solutions as another finite-dimensional ansatz space, on which we will consider the exact same operators.

6. Numerical Experiments

As a main component of this thesis, the proposed Algorithm 1 has been implemented in an extensive C++-project, enclosed on CD-ROM.
In this section, we will put the algorithm and the implementation to the test by performing some parameter studies and tests on classic toy data sets from machine learning. We will show, that even though the algorithm is meant to be an extension for Laplacian Eigenmaps it sometimes outperforms it concerning the information contained in its output and the stability towards some parameters. However, this gain comes at the cost of a higher complexity.

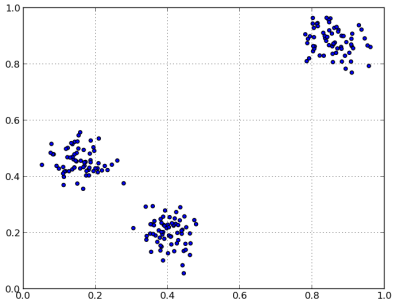
6.1. Implementation

The algorithm presented in section 5 has been implemented as a C++ program for the purpose of this thesis. Some of the functionalities have been adapted from C-code written by this thesis' supervisor Jochen Garcke for a presentation at SGA in Stuttgart [Gar14]. Pre- and postprocessing, as well as operating the program binary was performed using python scripts.
A short overview of the program's class structure and a documentation can be found in appendix A.
All of the relevant source code and documentation are also added to this thesis on an enclosed CD-ROM.

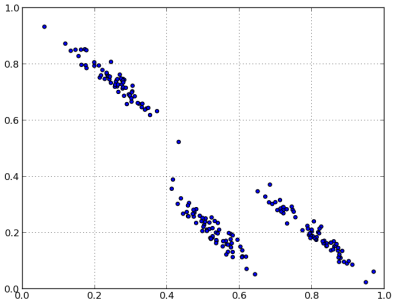
The computationally most complex part of the code, the solution of the several partial and final opticom generalized eigenvalue problems, is performed by using the FEAST eigenvalue solvers [Pol09], shortly introduced in Section 2.1. Further external libraries used were the Eigen library for its sparse matrix format [GJ$^+$10] and a kd-tree implementation to perform the k-nearest-neighbors search needed to set up the graph Laplacian on the nodes.
For the experiments the program binary was called from an adaptive python script, the plots used in this thesis were also generated in python using the matplotlib package [Hun07]. Data generation, preprocessing (normalizing the data, deciding on some parameters) and postprocessing (clustering, quality measuring) were also implemented in python, using several functionalities offered by the scikit-learn package [PVG$^+$11].

6.2. Data Sets

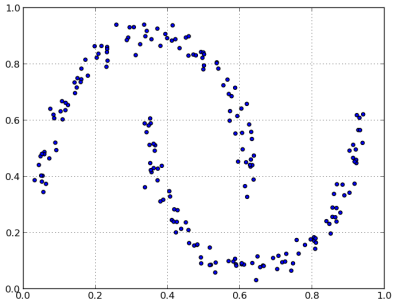
In the following, we present some of the synthetic data sets used for the experiments. Unless otherwise specified, the data sets have support on the unit square



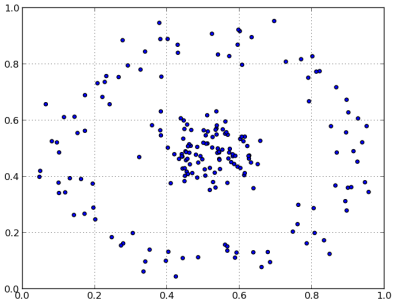
(a) Isotropic Blobs, number of centers $n = 3$.



(b) Anisotropic blobs, number of centers $n = 3$.



(c) Two Moons, added Gaussian noise with standard derivation $\sigma = 0.05$



(d) Two Circles, added Gaussian noise with standard derivation $\sigma = 0.1$

Figure 6.1.: Plots of example data sets

$[0, 1]^d$ and we have drawn 200 samples for the training and 2000 samples for test sets.

(Anisotropic) Blobs

The n-blobs data sets are generated using the `make_blobs()`-function provided by the Python package scikit-learn [PVG+11]. Here, samples are drawn from isotropic Gaussians around n different, randomly determined centers.
For the anisotropic blobs – a data set that is very interesting, because the classic k-means clustering has some difficulties with it – n blobs were created with `make_blobs()` as above and then linearly transformed with deterministic parameters. In the end, both data sets are fitted to the unit square. Plots of data sets generated this way are shown in Figures 6.1a and 6.1b.

Two Moons and Two Circles

Both Two Moons and Two Circles are classic and simple data sets to visualize and compare (bi-)clustering algorithms. Two Moons consists of two half circles that

are not linearly separable but interleaving; Two Circles is formed by a smaller and a larger circle around a common center. For the purpose of this thesis, both have been created with some added Gaussian noise, using the scikit-learn functions `make_moons()` and `make_circles()`. Here, too, the sets are fitted to the unit square, cf. Figures 6.1c and 6.1d.
These data sets are commonly used to test nonlinear learning algorithms, as classic clustering algorithms that work either by finding hypersurfaces to linearly separate clusters, or by finding concentric "blobs" in the data (such as k-means) fail to properly segment them. Two Circles is especially interesting, as the data points in the central circle lay much denser than those in the outer circle, adding an additional level of complexity to the clustering problem.

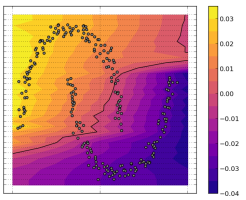
6.3. Embedding on Partial Grids

As a first impression of the functionality of our proposed algorithm, the final and partial results for a run on the Two Moons data set are depicted in Figure 6.2 for grid level $L = 6$ and margin $a_{\text{mar}} = 1$. The first 5 subfigures, 6.2a – 6.2e, show the upper layer (for the notion of layers, cf. 3.4) of partial grids ($\sum_d l_d = 6$), the next 4 plots, 6.2f – 6.2i, represent the second layer ($\sum_d l_d = 5$). In the last row, we see the combined result for the classic combination technique, Opticom on both layers and Opticom using only the top layer.

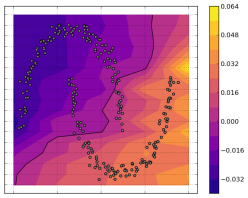
All embeddings here and in the remaining sections have been plotted in the following manner: The plot shows the embedding function applied to the unit square. Function values are visualized as colors, dyed according to a colormap that is optimized for human perception, as the colors increase their brightness as the function values increase. In the plot itself, a black line marks the zero contour, the contour along which a very basic bi-clustering algorithm (cf. Section 5.2.2) would split the clusters. The axis ticks in the plots indicate the refinement level of the respective grid. In case of a combined solution, the solution function has been evaluated at every grid point of the smallest full grid containing all relevant partial grids and then linearly interpolated. The data points used for training are depicted as gray circles.

6.4. Bi-Clustering Results

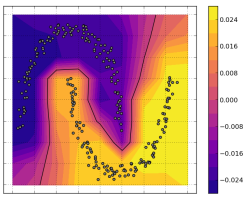
In this section, we will extensively consider our proposed method in the light of clustering of two dimensional not linearly separable data sets.
The potential use case we test our method for is the clustering of a large data set, where we wish to train an efficient embedding function on a small subset of the data (the training set), which will then be used embed the rest of the data (the out-of-sample set) in a consistent way.



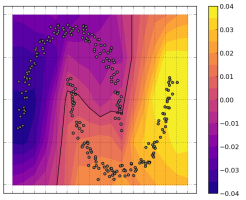
(a) Partial solution, grid level $\begin{pmatrix}1 & 5\end{pmatrix}^T$.



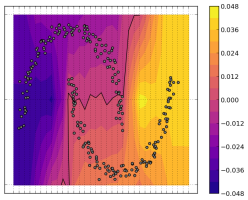
(b) Partial solution, grid level $\begin{pmatrix}2 & 4\end{pmatrix}^T$.



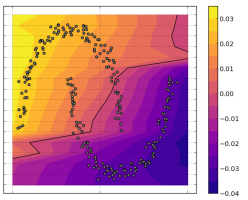
(c) Partial solution, grid level $\begin{pmatrix}3 & 3\end{pmatrix}^T$.



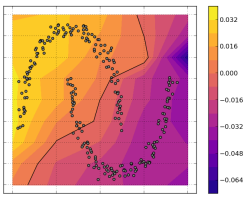
(d) Partial solution, grid level $\begin{pmatrix}4 & 2\end{pmatrix}^T$.



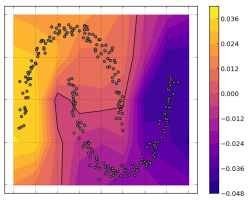
(e) Partial solution, grid level $\begin{pmatrix}5 & 1\end{pmatrix}^T$.



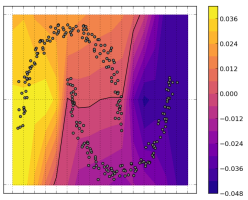
(f) Partial solution, grid level $\begin{pmatrix}1 & 4\end{pmatrix}^T$.



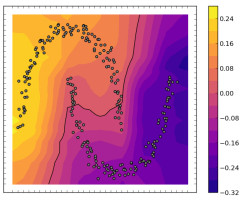
(g) Partial solution, grid level $\begin{pmatrix}2 & 3\end{pmatrix}^T$.



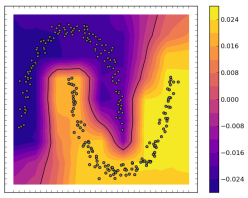
(h) Partial solution, grid level $\begin{pmatrix}3 & 2\end{pmatrix}^T$.



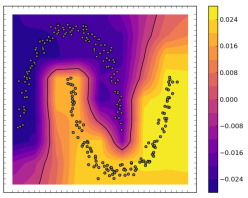
(i) Partial solution, grid level $\begin{pmatrix}4 & 1\end{pmatrix}^T$.



(j) Classic combination technique solution.



(k) Opticom solution using both layers.



(l) Opticom solution using one layer only.

Figure 6.2.: Partial and combined solutions of example run on Two Moons data set with $m = 250$ data points. Parameters: sparse grid level $L = 6$, $\delta = 8.0$, $\bar{\gamma} = 0.0001$, $n_{\text{nn}} = 8$.

6.4.1. Quality Measure

To determine the quality of out-of-sample-extensions, no clear formula has evolved. Different researchers use different quality measurements depending on their approaches. Most rely on some sort of cross validation, some compare the cross validation result then to the variability inside the training set. Possible measurements include:

- Simple k-fold cross validation[1], commonly 10-fold, as performed for out-of-sample extension for spectral embedding in [GLMH12]. Note the slight abuse of notation - classically, for cross validation the prediction error is compared to the actual (known) embedding. As in our case of unsupervised learning we do not have a "correct" embedding, we compare the results to those that are calculated when the cross validation set is itself part of the full training set.
- Comparison of a $(m-1)$-fold cross validation (*leave-one-out-cross validation*, LLOCV) with the so-called "training set variability", meaning the difference between the error calculated in LLOCV and the error on a subset $S \subset \mathcal{Y}$ when splitting $\mathcal{Y} \setminus S = R_1 \cup R_2$ and training with $S \cup R_1$ versus $S \cup R_2$. This method is used by Bengio et al. in their paper on Nyström-based out-of-sample-extension, [BPV+04].
- Usage of an intermediate clustering step, as performed by Peherstorfer et al. in the paper that motivated this thesis, [PPB11], and then applying a measure of cluster similarity: the so-called ARI (*adjusted Rand index*). The classic Rand Index constitutes a similarity measure between two clusterings by considering all pairs of samples and counting those pairs that are assigned in the same or different clusters in the predicted and true clusterings [HA85]: If we have a ground truth cluster assignment C and want to compare a new clustering K, we define a as the number of pairs of elements that are in the same set in C and in the same set in K, and b the number of pairs of elements that are in different sets in C as well as in different sets in K. Then the Rand index is calculates as:

$$\mathrm{RI}(C, K) := \frac{a+b}{C_2^M},$$

  with $C_2^M = \binom{M}{2}$ the total number of possible pairs.
  The adjusted rand index includes a normalization, yielding a score of -1.0 for fully dissimilar, 0.0 for random and 1.0 for identical clusterings. This normalization is performed via taking into account the expectation value, $E[\mathrm{RI}]$ as follows:

$$\mathrm{ARI}(C, K) := \frac{\mathrm{RI}(C, K) - E[\mathrm{RI}]}{\max(\mathrm{RI}) - E[\mathrm{RI}]}$$

[1] Cross validation: Split the data set $\mathcal{Y}$ randomly into k subsets of the same size. Then for each subset Y_i train the method with the $k-1$ other subsets, evaluate the obtained target function f for every $y \in Y_i$ and calculate the error compared to the embedding obtained when training with the full set. Calculate the mean over all subsets.

For our analysis, that focuses mostly on the clustering quality of the out-of-sample-extension, we apply the ARI score. As a ground truth, unless otherwise specified, we use the cluster assignment obtained with the corresponding Shi-Malik-clustering on the union of training and out-of-sample data points. This maps the potential use case: training with a small subset of the full data, while aiming for a clustering that features a comparable quality to one obtained directly from the full data. In a next step, we will then need to set this quality in relation to the potential gain in computational efficiency, cf. Section 6.4.5.

6.4.2. Parameter Studies

In the following, we will analyze the sensitivity of our proposed algorithm to the variation of its parameters. For our consideration, we sort the parameters according to the part of the algorithm, that is influenced by them: First, the parameters that determine the setup of the graph, namely the number $n_{\mathrm{nn}} \in \mathbb{N}$ that determines the number of edges present in the graph, and the bandwidth $\sigma \in \mathbb{R}$ of the Gaussian kernel defining the edge weights. Second a parameter influencing the partial solutions on the grids (and the Opticom system): the regularization parameter γ, weighing the grid-based regularization term. Third, the Opticom parameters, determining which grids we are going to consider: the total grid level $L \in \mathbb{N}$ and the margin $a_{\mathrm{mar}} \in \mathbb{N}$ describing how many of the highly anisotropic grids on the borders of our simplicial level structure will be ignored in the calculation.
The Opticom parameters will however not be treated separately. As to be expected, a growing refinement level L will lead to more elaborate results and a greater robustness towards parameter variations. For the simple data sets used in this section, already rather coarse levels will be enough to yield perfect ARI scores of 1.0. This effect can be observed in several of the figures supporting the studies of the other parameters, e.g. in Figure 6.7, where for each level the ARI score is plotted with respect to the regularization parameter.

Data Graph Parameters: n_{nn} and σ

The choice of parameters for the set up of the data graph, the number of nearest neighbors $n_{\mathrm{nn}} \in \mathbb{N}$ to be connected with an edge and the bandwidth for the Gaussian kernel $\sigma \in \mathbb{R}$, is quite tricky and there are few to no theoretic studies on this matter. At the same time, the results of spectral algorithms may be very sensitive to variations in the graph set up [Lux07].

Some rules of thumb, however, have been established and will also be applied in this thesis. For the bandwidth parameter σ it is common to use the (scaled) length of the longest edge in a minimum spanning tree over the data, or the mean distance of a point to its $(\log(m) + 1)$-nearest neighbor [Lux07]. We will follow the approach by Lafon [Laf04] for Diffusion Maps and use the mean difference to

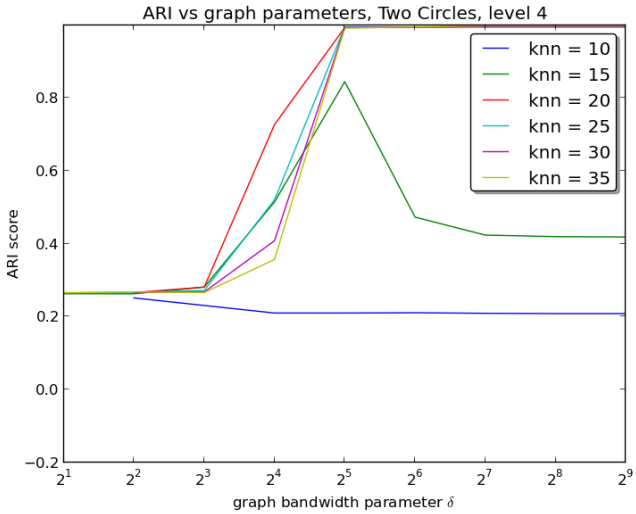
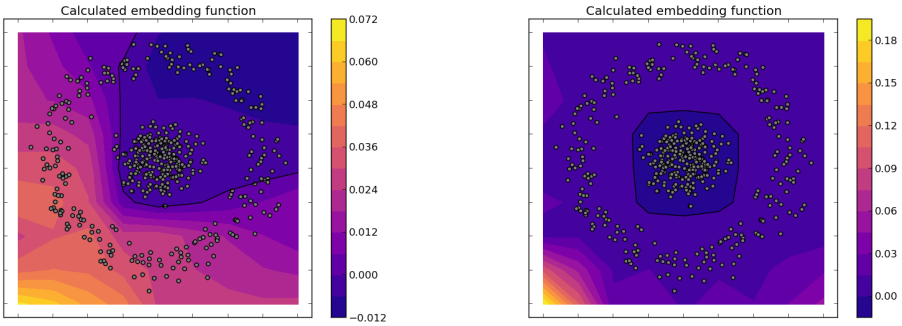


Figure 6.3.: The ARI score on the Two Circles data set for varying graph parameters. Training set size was 500, test set size was 2500. Note the sudden quality improvement, indicating a sensitivity towards the bandwidth parameter stemming from the existence of different density regions in the data set. Regularization parameter (fixed): $\bar{\gamma} = 10^{-9}$, level $L = 4$.

its closest neighbor, scaling by a parameter δ:

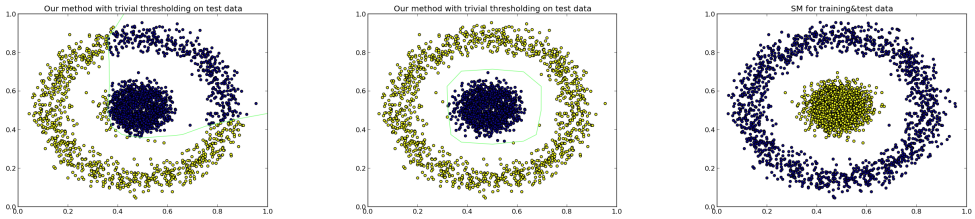
$$\sigma := \frac{\delta}{m} \sum_{x \in \mathcal{Y}} \min_{y \in \mathcal{Y}} \{\, d(x,y) \mid d(x,y) \neq 0 \,\}$$

Heuristically, a lower choice of σ (or δ, respectively) emphasizes local similarities higher, as the edge weights between spatially close points will be higher in contrast to those between spatially further apart points. At the same time, such a choice makes it more difficult to cluster outliers and to deal with data sets with a underlying density that varies in different regions.
An example is depicted in Figure 6.3, which shows the ARI for the Two Circles data set dependent on the bandwidth scaling parameter δ for varying choices of the number of nearest neighbors in the graph. It is clearly visible, that for all choices of "reasonable" n_{nn} there happens a sudden improvement, where the quality jumps from a arguably bad embedding (ARI score of ≈ 0.4) to a near perfect match to our target clustering (ARI score of ≈ 0.99) when scaling with $\delta = 32$. To make this difference more visible, consider Figure6.4, showing the embedding functions calculated from the training data for different δ, and Figure 6.5 showing the clustering on the out-of-sample data set next to the Shi-Malik clustering determining the ground truth.
For the choice of the number of nearest neighbors, one important factor to take into account is the efficiency of the algorithm. A complete graph ($n_{\text{nn}} = m$) will contain the most information, even though due to the Gaussian weight function far apart pairs of points will have negligible edge weights. Unfortunately, this graph's



(a) $\delta = 8.0$ (b) $\delta = 32.0$

Figure 6.4.: Embeddings for Two Circles data set, training set size 500. Parameters: $L = 4$, $n_{\text{nn}} = 25$, $\bar{\gamma} = 10^{-9}$



(a) $\delta = 8.0$ (b) $\delta = 32.0$ (c) Shi-Malik Clustering

Figure 6.5.: Clusterings of 2500 out-of-sample-data points for Two Circles data set, training set size 500. The green line shows the zero contour learned in the embedding, see Figure 6.4. On the right, we see our ground truth, the clustering obtained with the Shi-Malik-algorithm working directly on all training and out-of-sample data points. The different colors follow from the fact, that uniqueness of the eigenvectors is only given up to sign. Parameters: $L = 4$, $n_{\text{nn}} = 25$, $\bar{\gamma} = 10^{-9}$

matrices are not sparse and thus make the algorithm's operations computationally very expensive.
On the other hand, the size of n_{nn} must be chosen in a way, that will ideally lead to a connected graph. Otherwise, any clustering algorithm will simply return the connected components and all information on inter-cluster-similarity will be lost. For larger data sets, von Luxburg in her survey suggests a thumb rule of $n_{\text{nn}} = \log(m)$ [Lux07]. As the data sets we use in this thesis are not "large" in their sense, we will vary this parameter by hand, while keeping this rule in mind. Note that in our setting, even if the data graph is not connected, the graph on the grids (as introduced in Section 5.5) will always be, due to the addition of the regularization term. The resulting difference in clustering quality is especially notable when clustering into more than 2 clusters see Section 6.5.

In Figure 6.6 we show, that a careful choice of the neighborhood parameters n_{nn} and δ is crucial especially for small sparse grid levels as $L = 5$, where in two dimensions the largest of the partial grid features as little as 51 grid nodes and, due to the margin parameter $a_{\text{mar}} = 1$, only 4 grids are considered in total. In the second subplot, it can be seen that even for level $L = 4$ (largest grid with 27 nodes and a total of 3 grids) with a suitable parameter choice ARI scores of more than 0.94 can be achieved.
Analogous tests with grid level $L = 6$ on the same data set performed well, with ARI scores of more that 0.98 for all parameters in the tested range.

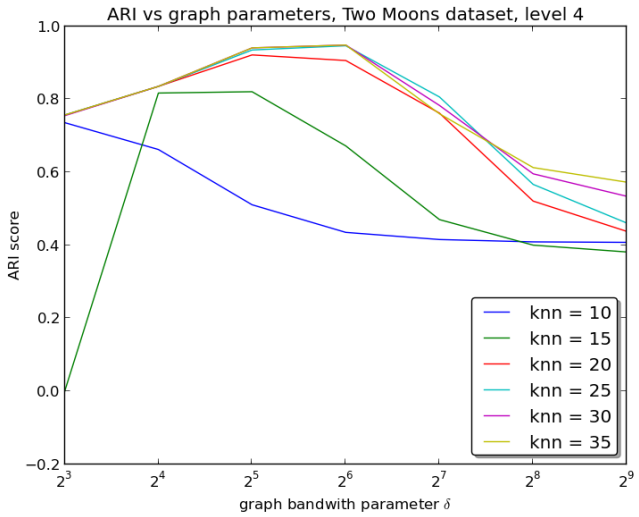
It can be noted that the optimal scaling parameter δ for the band width of the Gaussian kernel is rather high, considering the fact that is was originally suggested to be $\delta = 1$. A plausible explanation is the fact, that for our approach we do not use the weights between the data points directly, but rather distribute them to find weights for the interconnections between grid nodes. Thus, the finer our resolution gets and the less data points will be on the support of a single function, the more important it will get to still have significant weight between the respective data point and its neighbors. A suggestion for future investigations could thus be to choose the graph for Laplacian Eigenmaps already with the grids in mind and include grid properties such as mesh width into the choice of the kernel bandwidth.

Regularization Parameter γ

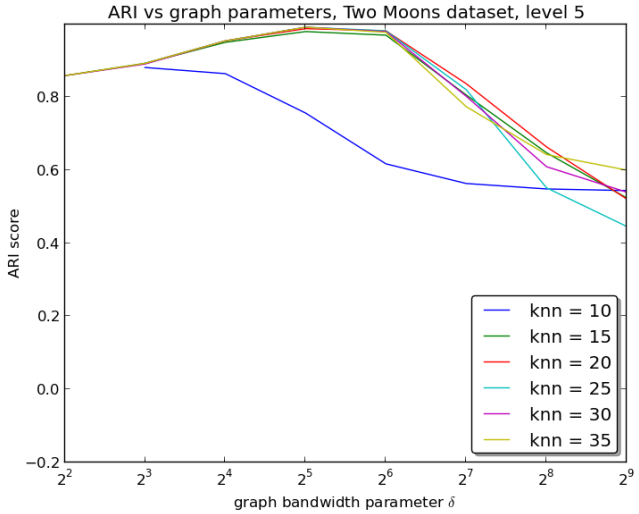
We scaled our regularization parameter $\gamma \in \mathbb{R}$ dependent on the data set size,

$$\gamma := \bar{\gamma} m^2,$$

where $\bar{\gamma} \in \mathbb{R}$ is a parameter passed by the user. The quadratic dependency of the data set size m was chosen heuristically due to the numerical derivation in Section 5.6, where the factor $\frac{1}{m}$ appears twice in the discretized data term on the left hand side and might thus need to be canceled in the regularization term. It remains to be empirically tested, if such an interconnection between regularization parameter and training data set size is justified, or if other descriptors of the problem are better suited for this task.



(a) ARI scores for Two Moons, level 4



(b) ARI scores for Two Moons, level 5

Figure 6.6.: ARI scores plotted against the graph parameters. We used the Two Moons data set with a training set size of 200 and an out-of-sample test set of 2000 and clustered according to trivial thresholding at zero. The regularization parameter was fixed at $\bar{\gamma} = 10^{-10}$. Ground truth is the clustering obtained with Shi-Malik spectral clustering on the combined set of 2200 data points.

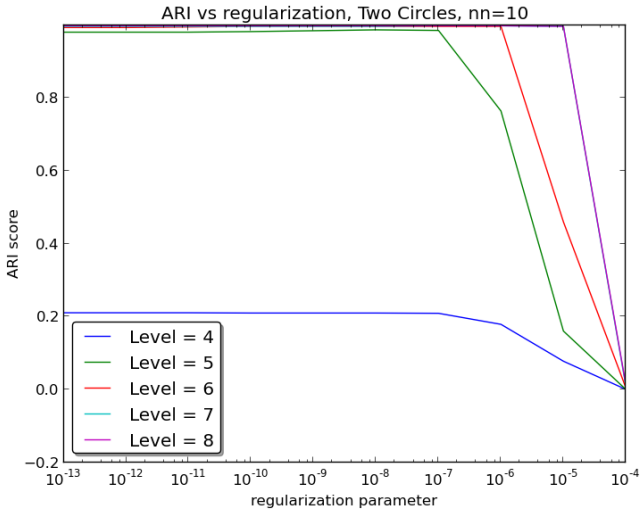


Figure 6.7.: ARI scores plotted against the regularization parameter for varying levels on the Two Circles data set. Further parameters: $n_{\mathrm{nn}} = 10$, $\delta = 32$.

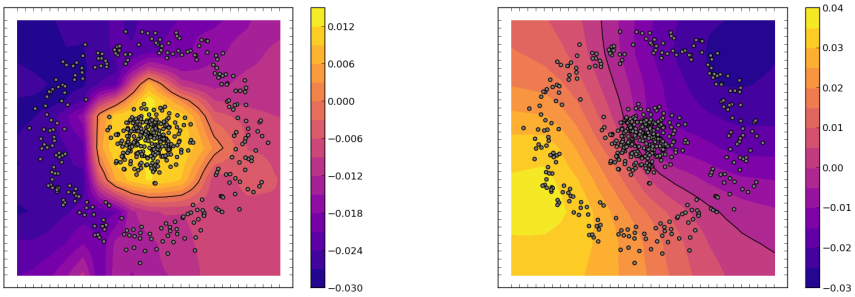
As for our experiments, we did not vary the data set size within one set of experiments, so m^2 may just be considered a scaling parameter.

The variation of $\bar{\gamma}$ played the most important role for grids with many "empty" cells, where several nodal basis functions had no data points on their support. This notion was also introduced shortly in the intuitive graph-based explanation in Section 5.5: If nodal basis functions are neighboring, even if they are not connected via data points on their respective supports, there should still be a notion of similarity.

On the other hand, if there are only data points, a high regularization parameter overshadows the connectivity stemming from the data points. In these cases, the embedding function will not stay true to the data. This can be seen in the sudden decrease of quality for rising $\bar{\gamma}$ in the plot in Figure 6.7 and in the exemplary embeddings in Figure 6.8, which depict the effect of too large regularization.

6.4.3. Choice of Clustering Function

As shortly noted in Section 5.2.2 for the bi-clustering we pursued in the Two Moons and Two Circles examples, a thresholding at zero can be applied as a trivial clustering algorithm. In these experiments, we additionally ran a k-means algorithm on the one-dimensional embedding, to take into account proximity to the means of the two clusters. The results obtained were mostly very similar concerning the ARI scores, as can be seen in the runtime tables, Table 6.1 and Table 6.2, and there is no obvious tendency towards one choice. Interestingly however, some outliers were present in both ways. Often, if both results are good (ARI Score of 0.95



(a) $\bar{\gamma} = 10^{-8}$ (b) $\bar{\gamma} = 10^{-4}$

Figure 6.8.: The influence of a growing regularization parameter on the Two Circles data set ($m = 500$). On the second plot, the term with the stiffness matrix dominates the problem and any data dependency is lost. Further parameters: $L = 6$, $n_{\text{nn}} = 10$, $\delta = 32$

or better), the k-means clustering scores slightly better, due to its more tolerant view towards the decision boundary at zero. This is especially notable in cases with a large regularization parameter.

The discussed comparability of the clustering naturally holds only for the biclusterings considered in this section. For higher-dimensional embeddings, where the number of clusters exceeds the number of target dimensions (which is the standard case for real-world-applications), the advantages of the k-means algorithm naturally predominate. However, also in these cases a thresholding of the coordinate axes at zero can be meaningful, cf. the figures in Section 6.5.

6.4.4. Use of the Classic Combination Technique

A natural and closely related choice of method to compare the Opticom eigenvalue approach to is the classical Combination technique. However, the following aspects need to be noted:

- The combination technique needs the second layer of grids (for the notion of layers, cf. Figure 3.4 in Section 3.2), thus almost twice as many partial solutions needs to be calculated.
- For more than one target dimension of the embedding (or, in clustering, the lower dimensional space in which the clustering algorithm is performed) we need to use the first variant introduced in Section 5.4, where we first attempt to match the pairs of the eigenvectors across the grids.

Our experiments showed that for the eigenvalue problems considered in this thesis the classic combination technique is not competitive. While it reacted a lot more sensitive towards the parameters, notably the regularization parameter $\bar{\gamma}$ in the sense that the results would slightly improve for larger values of $\bar{\gamma}$, the results were never found to be of high quality. A representative example is depicted in

$M_{\mathbf{train}}$	**RT Training**	**RT OOSE**	**ARI (k-m.)**	**ARI (thr.)**
50	1.19 s	0.06 s	0.119	0.000
100	1.5 s	0.06 s	0.008	0.008
150	1.86 s	0.05 s	0.810	0.940
200	2.41 s	0.08 s	0.945	0.990
250	2.65 s	0.07 s	1.0	1.0
300	3.28 s	0.05 s	1.0	1.0

Table 6.1.: Runtime and ARI score of Two Circle, independent out-of-sample test set with $M_{\text{oos}} = 2000$. ARI score calculated on ground truth created at data generation. Parameters: $n_{\text{nn}} = 20$, $\delta = 32$, $L = 5$, $\bar{\gamma} = 10^{-7}$.

Figure 6.2j. Here, the Opticom solution manages to balance the coefficients of the partial solution in a meaningful way, while the non-adaptive classic combination coefficients can not differentiate.

6.4.5. Runtime

Some empiric runtimes of both the Two Moons and the Two Circles data set are given in Tables 6.2 and 6.1. In both cases, we varied the size M_{train} of the training set between 2.5 and 15% of the size of the out-of-sample, M_{oos}. The sets were drawn independently.
In comparison, the full Laplacian Eigenmaps algorithm on both training and out-of-sample data runs about 5.5 seconds. Thus for both toy data sets, we succeeded in our approach to achieve high-quality approximations (ARI score of 0.95 or higher), while taking considerably less time than the full algorithm. As discussed before, the better results for Two Moons stem from the fact, that here the data density is more consistent. The Two Circles data set features a reason of lower density (the outer circle), so more samples of the data are needed to obtain a meaningful graph.

It needs to be noted that the measured runtimes to some scale depend on a fine-tuning of the FEAST solver. For example, a larger regularization parameter leads to the existence of more eigenvalues close to zero, and as the speed of the solver depends on the number of eigenvalues in a determined interval, the regularization parameter thus influences the runtime.

6.5. Clustering via n-dimensional Embeddings

For comparability and ease of inspection, the above-mentioned experiments were performed for biclustering, and using an embedding into $\mathbb{R}$.
As discussed in Section 5.4, a strength of Opticom for eigenvalue problems is the "automatic" sorting it will perform in the case of several eigenfunctions coming from each partial grids. This was put to the test using the anistropic blobs data

$M_{\mathbf{train}}$	**RT Training**	**RT OOSE**	**ARI (k-m.)**	**ARI (thr.)**
50	1.06 s	0.07 s	0.540	0.540
100	1.22 s	0.05 s	0.660	0.664
150	1.62 s	0.07 s	1.0	1.0
200	2.11 s	0.07 s	0.988	0.988
250	2.42 s	0.06 s	0.980	0.980
300	2.64 s	0.05 s	0.962	0.962

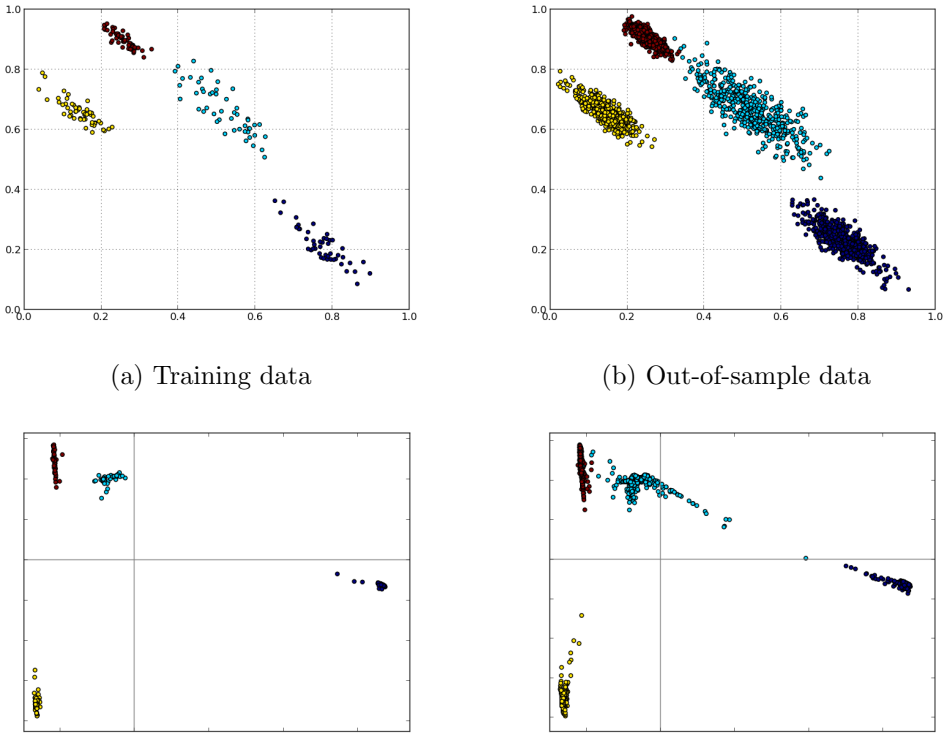
Table 6.2.: Runtime and ARI score of Two Moons, independent out-of-sample test set with $M_{\text{oos}} = 2000$. ARI score calculated on ground truth created at data generation. Parameters: $n_{\text{nn}} = 20$, $\delta = 32$, $L = 5$, $\bar{\gamma} = 10^{-7}$.

set, with different data densities for every blob. We mapped this two-dimensional data set again into the $\mathbb{R}^2$, while combining the solution function from *all* $2k$ eigenfunctions, $\left\{f_2^{(i)}, f_2^{(i)}\right\}_{i=1}^{k}$ coming from the k partial grids. For this problem, the proposed algorithm worked remarkably well. An example data set and its embedding is depicted in Figure 6.9. Note, that already an embedding onto $\mathbb{R}$ via the x-coordinate would yield quite an accurate one-dimensional embedding. While a trivial zero-thresholding clustering could not separate the yellow and light blue points, the k-means algorithm on the embedding had no problems splitting these two data set, yielding a perfect ARI score of 1.0 for the embedding of the training data, and a score of 0.988 for the formerly unknown out-of-sample data.

Notably, the application of the original Laplacian Eigenmaps for the 2-dimensional embedding gave worse results and found only three of the clusters, as its underlying graph was not connected, and thus each of the eigenvectors was piecewise constant. In the calculation of the ARI score above we therefore used the ground truth provided by the data generating function.

6.6. Embedding Results

While in the past section we concentrated on the clustering capacities of our proposed algorithm for two-dimensional data sets, while still having the underlying embedding functions in mind, we will now present the results for a standard, true embedding problem, the unrolling of the Swiss Roll dataset.
This classic toy dataset for machine learning features a two-dimensional submanifold embedded very nonlinear ("rolled") in $\mathbb{R}^3$, and it is the aim of the algorithm to recover the original two-dimensional structure. In Figure 6.10 we show the original data set, where the colors mark the x-direction in the originating manifold. Our proposed method, like the closely related method by Peherstorfer et. al., managed to find an embedding for the structure, as shown in Figure 6.11. Note that this embedding, even though it does still feature some curvature, is indeed "better" than the embedding obtained with Laplacian Eigenmaps on the training data only, as the distances between points are recovered more accurately. A plot



(a) Training data

(b) Out-of-sample data

(c) Embedding of training data

(d) Embedding of out-of-sample data

Figure 6.9.: Anistropic Blob data set with varying density, split in training data ($m = 200$) and test data ($m = 2000$), and the 2 dimensional embedding obtained when training on the training data. The colors represent ground truth labels, the gray lines in the embedding plots mark the coordinate axes. Parameters: $n_{nn} = 15$, $\delta = 32$, $\lambda = 10^{-5}$, $a_{\text{mar}} = 1$, $L = 5$.

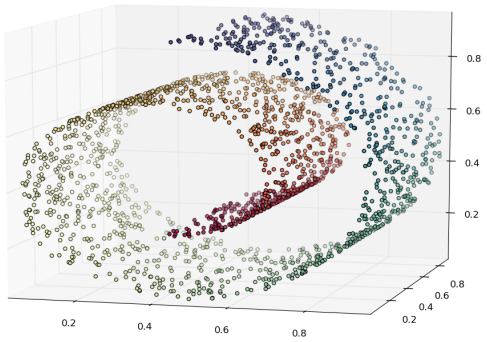


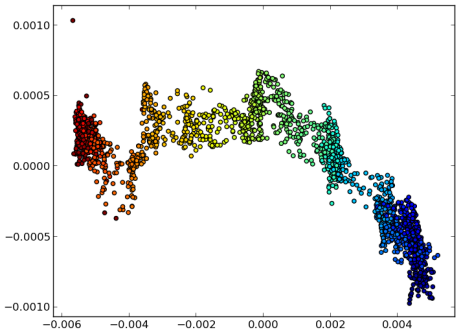
Figure 6.10.: The Swiss Roll data set with 2000 data points.

of the latter is depicted in the last subfigure of 6.11.

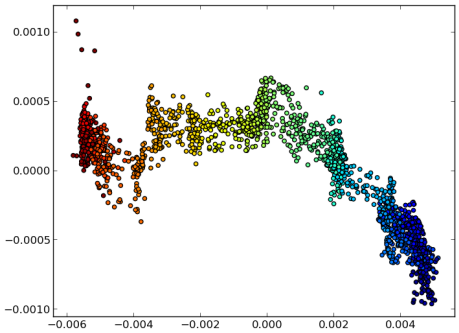
It needs to be noted, that already for this 3-dimensional case with a level as low as 5, the slow-down due to the projection products on the high-dimensional spaces was notable. A list of empiric runtimes of distinct steps of the algorithms is given in Table 6.3.

6.7. Limitations

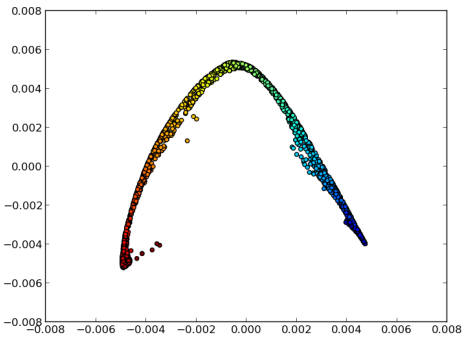
In addition to the influence on the runtime, some problems occurred during this experiments due to the FEAST solver. While it is fast and works well on sparse matrices, to be able to use it at its full power a lot of fine tuning is necessary, adjusting parameters such as the search interval and the number of eigenvalues to be found in this interval, adaptively and carefully. Still, for some matrices (often for very large and sparse ones, with high level L and low regularization parameter $\bar{\gamma}$) the solver did not converge, and it could not become clear, why this behavior appeared in these situations. For this reasons, our intended experiments to compare the results to those on a highly-refined full grid were not possible, as the solver aborted.
As our main use case will be situations with more data points than grid points, these effects may not occur in other setups.
Still, these problems will need to be investigated in the future, and it might even be useful to consider another solver, especially for use cases that can be run single-threaded and do not need the computational power of FEAST.



(a) Embedding for training data



(b) Embedding for test data



(c) LE embedding for test and training data

Figure 6.11.: The two-dimensional embeddings for the Swiss Roll data set found by our algorithm, for training and out-of-sample data set, both with a size of 2000, and with the original Laplacian Eigenmaps on the data points. The colors are chosen as in Figure 6.10 and denote the original position on the manifold. Parameters: level $L = 5$, $n_{nn} = 40$, $\delta = 64$, $\bar{\gamma} = 10^{-10}$.

Step	Runtime	(matrix size)
Grid eigenproblem level $\begin{pmatrix}1 & 1 & 3\end{pmatrix}^T$	4.07 s	81
Scalar products with previous solutions	0.0 s	
Grid eigenproblem level $\begin{pmatrix}1 & 2 & 2\end{pmatrix}^T$	4.16 s	75
Scalar products with previous solutions	12.62 s	
Grid eigenproblem level $\begin{pmatrix}2 & 1 & 2\end{pmatrix}^T$	2.96 s	75
Scalar products with previous solutions	24.10 s	
Grid eigenproblem level $\begin{pmatrix}1 & 3 & 1\end{pmatrix}^T$	2.96 s	81
Scalar products with previous solutions	35.89 s	
Grid eigenproblem level $\begin{pmatrix}2 & 2 & 1\end{pmatrix}^T$	2.96 s	75
Scalar products with previous solutions	47.65 s	
Grid eigenproblem level $\begin{pmatrix}3 & 1 & 1\end{pmatrix}^T$	2.98 s	81
Scalar products with previous solutions	59.33 s	
Final eigenproblem	0.01 s	12
Out of sample batch evaluation	0.16 s	

Table 6.3.: Detailed empiric runtimes for calculation of grid based embedding function for Swiss Roll. Measurement was performed on a simple workstation.

6.8. Open Questions

In the following, we summarize the open questions that appeared during the theoretical and experimental analysis of our method and that may be worth some further consideration.

Parameter Choices While some heuristic rules of thumb have been established for the classic spectral learning algorithms, it remains to be investigated, if these can just be copied for a grid-based approach. Our experiments indicated, that this might not be sufficient, but it might be necessary to include values like the grid's mesh width already into the graph setup.

Choice of Spectral Learning Algorithm So far, we based our algorithm on Laplacian Eigenmaps. While we noted in Chapter 4 that most of the data-based spectral learning algorithms are equivalent in some way, this does not automatically carry over to the grid-based reformulation. Intuitively, the random walk derivation of Diffusion Maps [CL06a], encoded in the random walk Laplacian $\mathbf{L}_{\mathrm{rw}}$, might be more suitable in the context of grids.

Convergence The existing results on the convergence of Laplacian Eigenmaps (cf. Section 4.5, [BN08a]) and on eigenvalue approximation on finite element meshes (cf. [Bof10]) could possibly be combined to derive convergence results for our proposed method. Unfortunately, a thorough mathematical investigation on this matter exceeded the scope of this thesis. Notable aspects to consider are the highly anisotropic grids that arise in the com-

bination technique and can strongly influence the local stability properties of the spectrum (cf. [Kat76]), as well as the exact properties of the spaces spanned by the partial solutions, which are the ansatz spaces for our second Galerkin step in the Opticom approach.

Computational Feasibility for higher-dimensional Problems The dimensions of both the ambient and the latent space influence our approach's runtime largely. The ambient space does this in a more obvious and well studied way which did in fact give rise to the sparse grid approach. But here, too, exist limitations up to which level sparse grids can be applied before suffering from the curse of dimensionality. Also the latent space's dimension $p \in \mathbb{N}$ has some influence to our algorithm's complexity, as in our suggested approach the final Opticom system is built from the scalar products of the kp partial solution functions (k the number of partial grids considered for the applied sparse grid level). As discussed before, these scalar products mean some computational overhead, as the considered functions are embedded into a more refined grid space, and the system matrix has to be constructed for this space as well. Efficient implementations for this need to be sought for, and depending on the effective computational loss the other options discussed in Section 5.4 may need to be reevaluated in this context.

Potential Adaptivity Considering the fact that in most of the considered problems (as usual in data analysis problems) the data points are very dense in some areas and rather sparse in others, a potential for adaptivity seems likely. In contrast to the hierarchical basis that is applied for the problem by Peherstorfer et al., for the Opticom approach there exists no adaptive refinement, as it draws its efficiency from the fact that the partial grids are regular. A sort of adaptivity might however be considered in future work by simply adding data dependent functions to the Galerkin ansatz space in the Opticom step.

7. Conclusion and Outlook

7.1. Conclusion

In this thesis, we investigated the possible application of the optimized sparse grids combination technique for eigenvalue problems in machine learning. To that end, we introduced and implemented a new framework for the grid-based out-of-sample-extension of the Laplacian Eigenmaps algorithm.
The diverse theoretic origins of the algorithm were collected, summarized and applied to interpret the new method from different points of view.

The proposed algorithm was then analyzed and tested with standard machine learning data sets. It has been shown, that for 2-dimensional clustering problems even for very few grid points a well-suited embedding function could be learned. Combined with an efficient batch evaluation, our method could deliver a quality compared to that of a full run of Laplacian Eigenmaps at a higher speed. This fulfilled one of the core quality assessments we defined for the algorithm.

Additionally, we evaluated the performance in a pure embedding setup. Here, the algorithm was able to recover the two-dimensional structure of the Swiss Roll data set. Remarkably, the provided embedding featured some qualities superior to the one obtained with Laplacian Eigenmaps. For this three dimensional problem, however, a loss of speed was notable.

7.2. Outlook

It needs to be evaluated, if the results also prove to be competitive when applied to higher-dimensional and real-world data. The extrapolation qualities observed in the tests are very promising, while the increase in computation time might be critical for the use in real-world-applications.

If this is feasible, several extensions and adaptions to the method are possible. In [BNS06] a two-fold regularization both in the ambient and in the latent space was proposed, which could positively affect our embedding quality. Additionally, for our proposed use case of training on a subsample and then cheaply extending the results to the rest of the sample, one could investigate the use of sophisticated subsampling methods,
So far, this thesis considered only the case of unsupervised learning, i.e. we worked with unlabeled training data. If some of the data is labeled, the clustering problem can be refined into a *classification* problem. For this semi-supervised learning ap-

proach, Sinha and Belkin suggested an extension of Laplacian Eigenmaps [SB10], which first finds a basis representation of the embedding and then adds the learning from labeled examples in a second step. It would be interesting to investigate, if a similar approach is also possible with our provided function representation.

A detailed list of more practical open questions following from the experimental results has additionally been compiled in section 6.8.

A. Appendix: Implementation Details

The code written for this thesis, including a python-script containing an example run, can be found on the enclosed CD-ROM.

A.1. Class structure

The main building blocks of the program are the following classes, ordered roughly according to their generality:

Experiment The most general class, created in `main.cpp`, built using all configuration parameters. Several experiments are implemented here, including cross validation, the general data-based Laplacian Eigenmaps algorithm and the functionalties for out-of-sample extension.

OCHandler This class manages the execution of Opticom for eigenvalue problems by performing all steps from Algorithm 1, from pre-processing to function evaluation. For the latter, OCHandler also stores all partial solutions internally.

OpticomEVSolver Virtual class to provide Solvers for generalized eigenvalue problems, for both sparse and dense problems.

FEASTOpticomEVSolver Wrapper for the FEAST implementation by Eric Polizzi, [Pol09]. If the solver does not converge, the wrapper will change the parameters accordingly and retry.

InputData Handles data management. Provides readers to load data stored in txt-format and can work directly on data to set up the data dependent Graph Laplacian and degree matrices.

RegularAnistropicGrid Building block reprensenting the partial grids. Provides all methods on function evaluation.

A detailed description of all member variables and method can be found in the documentation provided in Section A.2.

A.2. Documentation

On the following pages, find a detailed documentation of the program code, created with Doxygen.

Opticom Spectral Embedding Extension

Generated by Doxygen 1.7.6.1

Mon Apr 25 2016 22:38:26

Contents

1 Class Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

2 Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

3 File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

4 Class Documentation

4.1 CMatrix Class Reference

`#include <CMatrix.h>`

Public Member Functions

- **CMatrix** (int dim1, int dim2)
- **CMatrix** (const **CMatrix** &obj)
- void **Init** (int param_dim1, int param_dim2)
- int **GetDim1** () const
- int **GetDim2** () const
- int **GetSize** () const
- void **SetEntry** (const int &i, const int &j, real_t val)
- void **AddToEntry** (const int &i, const int &j, real_t val)
- void **regularise** (real_t gamma)
- void **regularise** (real_t gamma, real_cmat_t stiffnessMatrix)
- void **AddMatrix** (real_t coeff, **CMatrix** ∗mat)
- real_cvec_t **GetMatrixAsCVector** ()
- real_cvec_t **matrixVectorProduct** (real_cvec_t inputVector)
- real_t **scalarProduct** (real_cvec_t const &vec1, real_cvec_t const &vec2)
- real_t **operator()** (const int &i, const int &j)
- real_t **GetEntry** (const int &i, const int &j)
- void **print** ()

Protected Attributes

- int **dim1**
- int **dim2**
- int **size**
- real_cvec_t **asVector**
- bool **isAlloced**

4.1.1 Detailed Description

CMatrix.h (p. **??**)

Deprecated class, but still in use in code. Was originially be meant to inherit from some Matrix-Class to provide easy interchangability for Matrix formats. Internally, values are stored in a C-Array In the long to be replaced fully by Eigen matrices (EMatrix)

Author: Luisa Schwartz

4.1.2 Member Function Documentation

4.1.2.1 void CMatrix::AddMatrix (real_t *coeff*, CMatrix ∗ *mat*)

Add another **CMatrix** (p. 3)

4.1.2.2 void CMatrix::Init (int *param_dim1*, int *param_dim2*)

Allocated memory to store matrix

4.1.2.3 void CMatrix::regularise (real_t *gamma*)

Regularise by adding one value to all diagonal entries

4.1.2.4 void CMatrix::regularise (real_t *gamma*, real_cmat_t *stiffnessMatrix*)

Regularise by adding weighted stiffness matrix

The documentation for this class was generated from the following file:

- CMatrix.h

4.2 Experiment Class Reference

`#include <Experiment.h>`

Public Member Functions

- **Experiment** ()
- **Experiment** (const char ∗paramPath)
- virtual ∼**Experiment** ()

- void **Run** ()
- void **RunConsistency** ()
- void **RunBatch** (bool fullevaluation)
- real_cvec_t **CrossValidation** (int k)
- real_t **ConvergenceCheck** (int l, std::string filePathOriginal)
- void **CompareGridPureLE** ()
- void **TestBatchEvaluation** (bool fullevaluation)
- real_cvec_t **LaplacianEigenmapsPure** ()

4.2.1 Detailed Description

Experiment.h (p. **??**)

This class provides some experiments with the programs functions that go beyond simply running the algorithm (this is done in **OpticomHandler** (p. 12)).

So far, it includes pure Laplacian Eigenmaps, comparisons to Laplacian EM, k-fold cross validation and convergence experiments.

Author: Luisa Schwartz

4.2.2 Constructor & Destructor Documentation

4.2.2.1 Experiment::Experiment ( )

Default constructor.

4.2.2.2 Experiment::Experiment (const char * *paramPath*)

Standard constructor from config-file.

Parameters

paramPath	path to configuration file

4.2.2.3 virtual Experiment::~Experiment ( ) [virtual]

Destructor.

4.2.3 Member Function Documentation

4.2.3.1 void Experiment::CompareGridPureLE ( )

Run Laplacian Eigenmaps on data only (classic LE method) and store on disk, run -Opticom for grid-based embedding and store obtained embedding of data set on disk, compare both embeddings (only first eigenvector) and store difference vector on disk. All relevant filenames are generated from the parameter-struct.

4.2.3.2 real_t Experiment::ConvergenceCheck (int *l*, std::string *filePathOriginal*)

Run one instance of problem (determined via parameter member variable) and compare to result stored in a file (l2-distance).

Parameters

l	level of grid on which distance is to be calculated
filePathOriginal	path pointing to the file with the values on the to grid to be compared to

Returns

l2-distance between new and loaded solutions, calculated on grid points of level-l-grid

4.2.3.3 real_cvec_t Experiment::CrossValidation (int *k*)

Run k-fold cross validation (assumumption: data is given in random order), compare training and evaluating on full data set vs training on partial set and evaluating on remainder).

Parameters

k	parameter k for number of folds

Returns

average error over k folds when training on full set and evaluating on fold i vs training on all other folds and evaluating on fold i

4.2.3.4 real_cvec_t Experiment::LaplacianEigenmapsPure ( )

Run Laplacian Eigenmaps on data only (classic LE method) and return embedding

Returns

vector X, size nData∗numEV (the latter given via params, = targetDim), containing the first numEV eigenvalues found

4.2.3.5 void Experiment::Run ( )

A generic run function, so far managed by hand.

4.2.3.6 void Experiment::RunBatch (bool *fullevaluation*)

Setup data, then perform batch evaluation test: Run OOSE to obtain function, then perform batch OOSE on test data. All parameters will be taken from configuration file.

Parameters

fullevaluation	decides if solution function should additionally be evaluated on all grid points

4.2.3.7 void Experiment::RunConsistency ( )

Setup data, then run consistency check defined in **CompareGridPureLE()** (p. 5).

4.2.3.8 void Experiment::TestBatchEvaluation (bool *fullevaluation*)

Run OOSE to obtain function, then perform batch OOSE on test data. All parameters will be taken from configuration file.

Parameters

fullevaluation	decides if solution function should additionally be evaluated on all grid points

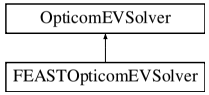
The documentation for this class was generated from the following file:

- Experiment.h

4.3 FEASTOpticomEVSolver Class Reference

`#include <FEASTOpticomEVSolver.h>`

Inheritance diagram for FEASTOpticomEVSolver:



Public Member Functions

- **FEASTOpticomEVSolver** ()
- virtual ∼**FEASTOpticomEVSolver** ()
- real_cvec_t **SolveFinalEigenproblem** (int numEV, int size, **CMatrix** ∗LHSMatrix, **CMatrix** ∗RHSMatrix)
- real_cvec_t **SolveSparseEigenproblem** (int numEV, int size, **feastSparseMatrix_t** ∗LHSFeastMatrix, **feastSparseMatrix_t** ∗RHSFeastMatrix, real_t EMax=1.0)

4.3.1 Detailed Description

OpticomEVSolver.h (p. **??**)

Wraps FEAST Solver for eigenvalue problem as it occurs in Opticom (generalized symmetric EV problem). Both sparse and dense are implemented.

Author: Luisa Schwartz

4.3.2 Constructor & Destructor Documentation

4.3.2.1 FEASTOpticomEVSolver::FEASTOpticomEVSolver ( )

Default constructor

4.3.2.2 virtual FEASTOpticomEVSolver::∼FEASTOpticomEVSolver ( ) [virtual]

Destructor

4.3.3 Member Function Documentation

4.3.3.1 real_cvec_t FEASTOpticomEVSolver::SolveFinalEigenproblem (int *numEV*, int *size*, **CMatrix** ∗ *LHSMatrix*, **CMatrix** ∗ *RHSMatrix*) [virtual]

Solves generalized eigenproblem given as CMatrices (wrapper for C array, deprecated!).

Parameters

numEV	Number of eigenvalues to be found
size	Outer dimensions of Matrix
LHSMatrix	Matrix for left hand side of generalized eigenproblem
RHSMatrix	Matrix for right hand side of generalized eigenproblem
X	Output vector, size size∗numEV, after finishing will contain the first numEV eigenvalues found

Implements **OpticomEVSolver** (p. 11).

4.3.3.2 real_cvec_t FEASTOpticomEVSolver::SolveSparseEigenproblem (int *numEV*, int *size*, **feastSparseMatrix_t** ∗ *LHSFeastMatrix*, **feastSparseMatrix_t** ∗ *RHSFeastMatrix*, real_t *EMax* = 1.0) [virtual]

Solves generalized eigenproblem given as **feastSparseMatrix_t** (p. 9) (sparse matrix format optimal for FEAST).

Parameters

numEV	Number of eigenvalues to be found
size	Outer dimensions of Matrix
LHSFeastMatrix	Matrix for left hand side of generalized eigenproblem
RHSFeastMatrix	Matrix for right hand side of generalized eigenproblem
EMax	upper limit of search interval for eigenvalues, problem-dependent all are in [0,1], but can be chosen smaller as we only want the smallest

Returns

vector X, size size∗numEV, containing the first numEV eigenvalues found

Implements **OpticomEVSolver** (p. 11).

The documentation for this class was generated from the following file:

- FEASTOpticomEVSolver.h

4.4 feastSparseMatrix_t Struct Reference

`#include <FeastHelper.h>`

Public Attributes

- int **innerSize**
- int **outerSize**
- real_cvec_t **valuePointer**
- int ∗ **outerIndexPointer**
- int ∗ **innerIndexPointer**

4.4.1 Detailed Description

FEAST needs peculiar, 1-based CSR-format for matrix storage. This struct stores all necessary pointers ans sizes in one. Compare FEAST documentation, section 3.3.2.

4.4.2 Member Data Documentation

4.4.2.1 int∗ feastSparseMatrix_t::innerIndexPointer

'js', pointer to array containing for each non-zero the column

4.4.2.2 int feastSparseMatrix_t::innerSize

number of non-zero elements, size of valuePointer and innerIndexPointer

4.4.2.3 int∗ feastSparseMatrix_t::outerIndexPointer

'is', positions in non-zero array, where new row starts

4.4.2.4 int feastSparseMatrix_t::outerSize

number of non-zero-rows+1, size of outerIndexPointer

4.4.2.5 real_cvec_t feastSparseMatrix_t::valuePointer

pointer to array holding all non-zero matrix elements

The documentation for this struct was generated from the following file:

- FeastHelper.h

4.5 InputData Class Reference

Public Member Functions

- **InputData** ()
- **InputData** (std::vector< real_cvec_t > ppoints, int pdim)
- **InputData** (const char ∗path, int pdim)
- ∼**InputData** ()
- int **GetNum** () const
- int **GetDim** () const
- bool **IsDataInitialized** () const
- bool **IsMatricesInitialized** () const
- real_cvec_t **GetDataPoint** (int dataPointIdx) const
- std::vector< real_cvec_t > **GetDataVec** () const
- real_cmat_t **GetPointerToData** ()
- real_t **GetLaplacianFull** (int i, int j)
- real_t **GetDegreeFull** (int i)
- EMatrix **GetLaplacianKNN** ()
- real_t **GetDegreeKNN** (int i)
- void **Read2dDataFromFile** (const char ∗datapath)
- void **Read3dDataFromFile** (const char ∗datapath)

4.5.1 Constructor & Destructor Documentation

4.5.1.1 InputData::InputData ( )

Default constructor.

4.5.1.2 InputData::InputData (std::vector< real_cvec_t > *ppoints*, int *pdim*)

Constructor from given point set.

Parameters

ppoints	point set $\mathscr{S} \subset \mathbb{R}^d$
pdim	dimension d of points in point set

4.5.1.3 InputData::InputData (const char ∗ *path*, int *pdim*)

Constructor with data path.

Parameters

path	path to txt-file containing data points
dim	dimension of the data

4.5.1.4 InputData::~InputData ( )

Destructor.

4.5.2 Member Function Documentation

4.5.2.1 void InputData::Read2dDataFromFile (const char * *datapath*)

Reads 2D data from file.

Reads 2D data from file, first two columns are considered as the data points, further columns are ignored.

Parameters

datapath	path to the data file

4.5.2.2 void InputData::Read3dDataFromFile (const char * *datapath*)

Reads 3D data from file.

Reads 3D data from file, first two columns are considered as the data points, further columns are ignored.

Parameters

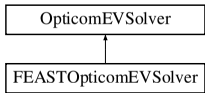
datapath	path to the data file

The documentation for this class was generated from the following file:

- InputData.h

4.6 OpticomEVSolver Class Reference

Inheritance diagram for OpticomEVSolver:



Public Member Functions

- virtual real_cvec_t **SolveFinalEigenproblem** (int numEV, int size, **CMatrix** *LHSMatrix, **CMatrix** *RHSMatrix)=0
- virtual real_cvec_t **SolveSparseEigenproblem** (int numEV, int size, **feastSparseMatrix_t** *LHSFeastMatrix, **feastSparseMatrix_t** *RHSFeastMatrix, real_t EMax=1.0)=0

The documentation for this class was generated from the following file:

- OpticomEVSolver.h

4.7 OpticomHandler Class Reference

Public Member Functions

- **OpticomHandler** ()
- **OpticomHandler** (const char *paramPath)
- **OpticomHandler** (**parameters_t** pparam)
- **OpticomHandler** (**parameters_t** pparam, **InputData** *pdata)
- virtual ~**OpticomHandler** ()
- void **Run** ()
- void **RunFullMax** ()
- void **EvaluateOpticomSolutionOnGrid** (int_cvec_t gridlevel)
- real_t **CompareDifferenceOpticomSolutionOnGrid** (std::string filePathOriginal, int_cvec_t gridlevel)
- real_cmat_t **EvaluateOpticomSolution** (real_cvec_t point)
- real_cmat_t **EvaluateOpticomSolutionOnPointSet** (std::vector< real_cvec_t > *pointSet)
- void **EvaluateCombiSolutionOnGrid** (int_cvec_t gridlevel)
- void **setGridEvaluation** (bool b)

4.7.1 Constructor & Destructor Documentation

4.7.1.1 OpticomHandler::OpticomHandler ( )

Default Constructor (Init needs to be called later).

4.7.1.2 OpticomHandler::OpticomHandler (const char * *paramPath*)

Constructor from parameter file.

Parameters

paramPath	location of configuration file holding the parameters

4.7.1.3 OpticomHandler::OpticomHandler (parameters_t *pparam*)

Constructor from parameter struct.

Parameters

pparam	parameters

4.7.1.4 OpticomHandler::OpticomHandler (parameters_t *pparam*, InputData * *pdata*)

Constructor from parameter struct and data.

Parameters

pparam	parameters
pdata	data already read and stored as **InputData** (p. 10)

4.7.1.5 virtual OpticomHandler::~OpticomHandler ( ) [virtual]

Destructor.

4.7.2 Member Function Documentation

4.7.2.1 real_t OpticomHandler::CompareDifferenceOpticomSolutionOnGrid (std::string *filePathOriginal*, int_cvec_t *gridlevel*)

Calculate difference of Opticom solution to another function on grid space, stored at <filePathOriginal>.

Parameters

filePath-Original	path to file storing grid based function
gridlevel	resolution of grid stored in <filePathOriginal>

4.7.2.2 void OpticomHandler::EvaluateCombiSolutionOnGrid (int_cvec_t *gridlevel*)

Evaluate ct solution on every grid point of full isotropic grid.

Evaluate combination technique solution on every grid point and save list of all function values for further processing in file specified via parameters.

Parameters

gridlevel	level of the (full isotropic) grid on that function shall be evaluated

4.7.2.3 real_cmat_t OpticomHandler::EvaluateOpticomSolution (real_cvec_t *point*)

Evaluate Solution on a single point.

Evaluate Solution on a single point, via creating a point set with single point and calling point set evaluation function.

Parameters

point	point to be evaluated

Returns

cMatrix with dim (targetDim x 1) containing the embedding of point

4.7.2.4 void OpticomHandler::EvaluateOpticomSolutionOnGrid (int_cvec_t *gridlevel*)

Evaluate Opticom Solution on every grid point of full isotropic grid.

Evaluate Opticom Solution on every grid point and save list of all function values for further processing in file specified via parameters.

Parameters

gridlevel	level of the (full isotropic) grid on that function shall be evaluated

4.7.2.5 real_cmat_t OpticomHandler::EvaluateOpticomSolutionOnPointSet (std::vector< real_cvec_t > * *pointSet*)

Evaluate Solution on point set.

Parameters

pointSet	vector of points to be evaluated

Returns

cMatrix with dim (targetDim x sizePointSet) containing the embeddings of all points

4.7.2.6 void OpticomHandler::Run ( )

Full run.

4.7.2.7 void OpticomHandler::RunFullMax ( )

Run on full grid of Maximum Level.

4.7.2.8 void OpticomHandler::setGridEvaluation (bool *b*) [inline]

Decide, if function should be evaluated and grid and output to be stored. If only evaluations at data points or other defined pointset is needed, and function can be forgotten afterwards, set false.

Parameters

b	true, if target function should be evaluated and grid and output to be stored. false, otherwise.

The documentation for this class was generated from the following file:

- OpticomHandler.h

4.8 parameters_t Struct Reference

`#include <OpticomHandler.h>`

Public Attributes

- int **dim**
- int **targetDim**
- int **sgLevel**
- real_vec_t **start**
- real_vec_t **length**
- int **numLayers**
- int **margin**
- int **knn**
- real_t **delta**
- real_t **lambda**
- real_t **gamma**
- std::string **datapath**
- std::string **outputpath**
- std::string **testdatapath**

4.8.1 Detailed Description

Struct containing all parameters to set up Opticom problem

4.8.2 Member Data Documentation

4.8.2.1 std::string parameters_t::datapath

path to training data file (see **InputData** (p. 10) class)

4.8.2.2 real_t parameters_t::delta

parameter for Gaussian ($\sigma = \delta \sigma_{\text{opt}}$)

4.8.2.3 int parameters_t::dim

dimension of problem

4.8.2.4 real_t parameters_t::gamma

parameter for regularization: $\gamma = \lambda n_{\text{data}}$

4.8.2.5 int parameters_t::knn

number of nearest neighbours to build neighborhood graph

4.8.2.6 real_t parameters_t::lambda

(user input) parameter for regularization

4.8.2.7 real_vec_t parameters_t::length

extension of the domain in all directions

4.8.2.8 int parameters_t::margin

how many highly anisotropic grids will be ignored

4.8.2.9 int parameters_t::numLayers

number of layers to be used (opticom usually one or two, combi technique uses two)

4.8.2.10 std::string parameters_t::outputpath

path where to write output files

4.8.2.11 int parameters_t::sgLevel

maximum sparse grid level in one coord direction

4.8.2.12 real_vec_t parameters_t::start

coordinates of the origin of the domain

4.8.2.13 int parameters_t::targetDim

(target) dimension of embedding

4.8.2.14 std::string parameters_t::testdatapath

path to test data file (see **InputData** (p. 10) class)

The documentation for this struct was generated from the following file:

- OpticomHandler.h

4.9 partial_solution_t Struct Reference

`#include <OpticomHandler.h>`

Public Attributes

- **RegularAnisotropicGrid grid**
- real_cvec_t **solution_vector**
- int **size**

4.9.1 Detailed Description

OpticomHandler.h (p. **??**)

Provides class to manage the Opticom Sparse Grid Eigenvalue Problem. Will create grid objects to solve problems on grids, then collect and manage results to set up system for Opticom Step and solve. Also provides functions to evaluate obtained Opticom solution functions, on grids, point sets and single points.

Author: Luisa Schwartz Struct containing one partial solution on an anisotropic grid (solution vector in R^size) and the grid itself

4.9.2 Member Data Documentation

4.9.2.1 RegularAnisotropicGrid partial_solution_t::grid

grid on which solution is defined

4.9.2.2 int partial_solution_t::size

size of grid

4.9.2.3 real_cvec_t partial_solution_t::solution_vector

coefficient vector defining solution on grid, $f = \sum_i c_i \phi_i$

The documentation for this struct was generated from the following file:

- OpticomHandler.h

4.10 RegularAnisotropicGrid Class Reference

`#include <RegularAnisotropicGrid.h>`

Public Member Functions

- **RegularAnisotropicGrid** ()
- **RegularAnisotropicGrid** (int dim, int_cvec_t level, real_vec_t start, real_vec_t length)
- **RegularAnisotropicGrid** (int dim, int_cvec_t level)
- **RegularAnisotropicGrid** (const **RegularAnisotropicGrid** &obj)
- ∼**RegularAnisotropicGrid** ()
- int **GetDim** () const
- int_cvec_t **GetLevel** () const
- int **GetSize** () const
- real_t **GetH** (int dir) const
- int **GetNumPointsInDir** (int dir) const
- int **GetNextInDir** (int dir) const
- real_t **GetGridCoor** (int dir, int idx) const
- real_vec_t **GetStart** () const
- real_vec_t **GetLength** () const
- real_cvec_t **CoordFromIdx** (int nodeIdx) const
- real_t **EvaluateFunction** (real_cvec_t coefficients, real_cvec_t xi) const
- real_t **EvalMultiLinNodalBasisFunction** (int node_idx, real_cvec_t xi) const
- bool **isDataPointOnSupportOfFunction** (real_cvec_t xi, int nodeIdx) const
- bool **isDataPointOnSupportOfFunction** (real_cvec_t xi, real_cvec_t node) const
- int **GetDistancesAndQuadIdxForPoint** (real_cvec_t xi, real_cvec_t distance) const
- void **regularizeSparseMatrixOnGrid** (real_t gamma, std::vector< T > ∗sparseMatrixTripletList)
- void **InitLocalMatrices** ()

Protected Member Functions

- void **Init** (int dim, int_cvec_t level, real_vec_t start, real_vec_t length)
- int **GridCoordToIdx** (int_vec_t multiindex) const
- int **IdxLocalToGlobal** (int localIdx, int elementCornerIdx) const
- bool **nodeDefinesElement** (int nodeIdx)
- int_vec_t **GridCoordFromIdx** (int nodeIdx) const

Protected Attributes

- int **dim**
- int_cvec_t **level**
- real_vec_t **start**
- real_vec_t **length**
- real_vec_t **h**
- int_vec_t **pts_per_dir**
- int_vec_t **next**
- int **size**
- std::vector< real_vec_t > **pt_positions**
- real_cmat_t **pt_positions_C**
- int **numElementCorners**
- real_cmat_t **stiff1D**
- real_cmat_t **mass1D**
- real_cmat_t **massLocal**
- real_cmat_t **stiffLocal**
- bool **localMatricesInititalized**

4.10.1 Detailed Description

RegularAnisotropicGrid.h (p. **??**)

This class represents a regular anisotropic grid for finite element methods on a rectangular domain. Offers several evaluation functions, internally has access to mass and stiffness matrices.

Possible future extensions: Provide scalar products, output mass and stiffness matrices

Author: Luisa Schwartz

4.10.2 Constructor & Destructor Documentation

4.10.2.1 RegularAnisotropicGrid::RegularAnisotropicGrid ( )

Default Constructor, only called by program. Does nothing.

4.10.2.2 RegularAnisotropicGrid::RegularAnisotropicGrid (int *dim*, int_cvec_t *level*, real_vec_t *start*, real_vec_t *length*)

Full Constructor.

Parameters

dim	dimension of the problem
level	discretization level of the grid
start	coordinates of origin of domain / grid ("lower left corner")
length	extension of domain into every direction

4.10.2.3 RegularAnisotropicGrid::RegularAnisotropicGrid (int *dim*, int_cvec_t *level*)

Constructor for unit square. Fixes domain automatically.

Parameters

dim	dimension of the problem
level	discretization level of the grid

4.10.2.4 RegularAnisotropicGrid::RegularAnisotropicGrid (const RegularAnisotropicGrid & *obj*)

Copy Constructor. Provides deep copy of obj.

Parameters

obj	instance to be copied

4.10.2.5 RegularAnisotropicGrid::~RegularAnisotropicGrid ( )

Destructor.

4.10.3 Member Function Documentation

4.10.3.1 real_cvec_t RegularAnisotropicGrid::CoordFromIdx (int *nodeIdx*) const

Translate index of a node to its spatial oordinates.

Parameters

nodeIdx	index of nodal basis function in question

Returns

coordinates of that node

4.10.3.2 real_t RegularAnisotropicGrid::EvalMultiLinNodalBasisFunction (int *node_idx*, real_cvec_t *xi*) const

Evaluation function for basis functions on grid.

Evaluate a multilinear basis function of the grid identified via its node index at a point xi (DEPRECETED, use distances instead!).

Parameters

nodeIdx	node identified by index *i* which nodal basis function shall be evaluated
xi	evaluation point $\xi \in \mathbb{R}^d$

Returns

$\phi_i(\xi)$

4.10.3.3 real_t RegularAnisotropicGrid::EvaluateFunction (real_cvec_t *coefficients*, real_cvec_t *xi*) const

Evaluation function given via coefficient on grid.

Parameters

coefficients	coefficient vector $c \in \mathbb{R}^N$ for function defined on grid
xi	evaluation point $\xi \in \mathbb{R}^d$

Returns

$\sum_{i=0}^{N-1}(c_i \phi_i(\xi))$

4.10.3.4 int RegularAnisotropicGrid::GetDistancesAndQuadIdxForPoint (real_cvec_t *xi*, real_cvec_t *distance*) const

Return supporting quad and normalized distances to all corners.

Return index of supporting quad and normalized (to unit square) distances to all corners for given point (corresponding to 1D basis function evaluations); easier to handle than standard evaluation function (normalized according to basis function evaluation).

Parameters

xi	point of evaluation $\xi \in \mathbb{R}^d$
distance	(output) array with distances to "origin corner" in all coord directions

Returns

index of origin node of that quad

4.10.3.5 int_vec_t RegularAnisotropicGrid::GridCoordFromIdx (int *nodeIdx*) const [protected]

Convert node index to integer grid coordinates.

Parameters

nodeIdx	node index

Returns

grid coordinates ($\nu \in \mathbb{N}^d$)

4.10.3.6 int RegularAnisotropicGrid::GridCoordToIdx (int_vec_t *multiindex*) const [protected]

Translate (integral) grid coordinates of node into index of that node.

Parameters

multiindex	grid coordinate $\nu \in \mathbb{N}^d$ of nodal basis function ϕ_i in question

Returns

index *i* of that node

4.10.3.7 int RegularAnisotropicGrid::IdxLocalToGlobal (int *localIdx*, int *elementCornerIdx*) const [protected]

For an element defined by its lowest corner index, return global index corresponding to a local index on that element.

Parameters

localIdx	local index on element
elementCornerIdx	element identifier (global index of lowest corner)

Returns

global index corresponding to local index on element

4.10.3.8 void RegularAnisotropicGrid::Init (int *dim*, int_cvec_t *level*, real_vec_t *start*, real_vec_t *length*) [protected]

Initialization Function called by all constructors (does not initialize local Matrices).

Parameters

dim	Dimesion of the problem
level	Discretization level of the grid
start	Coordinates of origin of domain / grid ("lower left corner")
length	Extension of domain into every direction

4.10.3.9 void RegularAnisotropicGrid::InitLocalMatrices ( )

Initialize local mass and stiffness matrix (member variables).

4.10.3.10 bool RegularAnisotropicGrid::isDataPointOnSupportOfFunction (real_cvec_t *xi*, int *nodeIdx*) const

Checks if some point is on support of basis function.

Parameters

xi	point of evaluation $\xi \in \mathbb{R}^d$
nodeIdx	index i of nodal basis function in question

Returns

true if $\xi \in \text{supp}(\phi_i)$, false if not.

4.10.3.11 bool RegularAnisotropicGrid::isDataPointOnSupportOfFunction (real_cvec_t *xi*, real_cvec_t *node*) const

Checks if some point is on support of basis function.

Checks if some point is on support of basis function, needs less total lookups than the one with (real_cvec_t, int).

Parameters

xi	point of evaluation $\xi \in \mathbb{R}^d$
node	coordinates of node x_i defining nodal basis function ϕ_i in question

Returns

true if $\xi \in \text{supp}(\phi_i)$, false if not.

4.10.3.12 bool RegularAnisotropicGrid::nodeDefinesElement (int *nodeIdx*) [protected]

Check if node defines element (node is not on right boundary in any coordinate direction).

Parameters

nodeIdx	global index of node in question

Returns

true if node defines element, otherwise false

4.10.3.13 void RegularAnisotropicGrid::regularizeSparseMatrixOnGrid (real_t *gamma*, std::vector< T > * *sparseMatrixTripletList*)

Add gamma*(grid stiffness matrix) to Eigen Triplet list.

Add entries of γC , with C grid stiffness matrix to Eigen Triplet list, from which as next step a sparse matrix can be built in Eigen format.

Parameters

gamma	regularisation parameter $\gamma \in \mathbb{R}$
sparseMatrixTripletList	triplet list representing Eigen sparse matrix, will be extended by function call

The documentation for this class was generated from the following file:

- RegularAnisotropicGrid.h

5 File Documentation

5.1 EigenMatrixHelper.h File Reference

`#include <Eigen/Sparse>` `#include "MatVecHelper.h"` `#include "FeastHelper.h"`

Typedefs

- typedef Eigen::Triplet< real_t > **T**
- typedef Eigen::SparseMatrix < real_t, Eigen::RowMajor > **EMatrix**

Functions

- void **addToDiagonalSparse** (EMatrix mat, real_t val)
- void **printSparseMatrix** (EMatrix mat)
- void **printSparseMatrixCompact** (EMatrix mat)
- void **sparseEigenToFEAST** (EMatrix *mat, real_cvec_t valuePtrFeast, FEAST_INT *innerIndexPtrFeast, FEAST_INT *outerIndexPtrFeast)
- void **sparseEigenToSparseFEAST** (EMatrix *mat, **feastSparseMatrix_t** *feastmat)

5.1.1 Detailed Description

Contains some typedefs and functions to work with Eigen sparse matrices.

Author: Luisa Schwartz

5.1.2 Function Documentation

5.1.2.1 void addToDiagonalSparse (EMatrix *mat*, real_t *val*)

Add some value to matrix diagonal.

Add some value to matrix diagonal (e.g. for regularisation).

Parameters

mat	sparse matrix to be changed
val	value to add to each diagonal entry

5.1.2.2 void printSparseMatrix (EMatrix *mat*)

Print an Eigen Sparse Matrix.

Print an Eigen Sparse Matrix to the console via transformation to dense matrix.

Parameters

mat	sparse matrix to be printed

5.1.2.3 void printSparseMatrixCompact (EMatrix *mat*)

Print an Eigen Sparse Matrix in Compact form.

Print an Eigen Sparse Matrix to the console in sparse form, giving just nonzero entries for each row and that value's respective column index.

Parameters

mat	sparse matrix to be printed

5.1.2.4 void sparseEigenToFEAST (EMatrix * *mat*, real_cvec_t *valuePtrFeast*, FEAST_INT * *innerIndexPtrFeast*, FEAST_INT * *outerIndexPtrFeast*)

Cast Eigen Sparse Matrix to FEAST format (in pieces).

Cast Eigen Sparse Matrix (0-based indices) to FEAST format (1-based indices).

Parameters

mat	Eigen sparse matrix
valuePtr	FEAST value pointer (target)
innerIndex-Pointer	FEAST j-index pointer (target)
outerIndex-Pointer	FEAST i-index pointer (target)

5.1.2.5 void sparseEigenToSparseFEAST (EMatrix * *mat*, feastSparseMatrix_t * *feastmat*)

Transfer Eigen Sparse Matrix to FEAST format (into one **feastSparseMatrix_t** (p. 9) struct).

Transfer Eigen Sparse Matrix (0-based indices) to FEAST format (1-based indices, one **feastSparseMatrix_t** (p. 9) struct).

Parameters

mat	Eigen sparse matrix
feastmat	Feast sparse Matrix

5.2 GridProlongation.h File Reference

`#include "MatVecHelper.h"` `#include "RegularAnisotropicGrid.h"` `#include "InputData.h"` `#include "EigenMatrixHelper.h"`

Functions

- void **SetupGridMatricesKNN** (**RegularAnisotropicGrid** *grid, **InputData** *data, EMatrix *eigenLHS, EMatrix *eigenRHS, real_t gamma=0.)
- real_vec_t **calcProjectionScalarProduct** (real_cvec_t vec1, **RegularAnisotropicGrid** *grid1, real_cvec_t vec2, **RegularAnisotropicGrid** *grid2, **InputData** *data, real_t gamma=0.)
- real_cvec_t **vectorProlongation** (real_cvec_t vector, **RegularAnisotropicGrid** *originGrid, **RegularAnisotropicGrid** *targetGrid)
- bool **gridlineIterator** (**RegularAnisotropicGrid** *originGrid, int *originPosition, int *originIndex, **RegularAnisotropicGrid** *targetGrid, int *targetPosition, int *targetIndex, int *scale_diff, int transitionLevel, int dim, int face)

5.2.1 Detailed Description

Defines some helper function to work with Regular Anisotropic Grids in Opticom; including grid prolongation, grid iteration and the setup of the grid-based Laplacians given the Input data.

Author: Luisa Schwartz

5.2.2 Function Documentation

5.2.2.1 real_vec_t calcProjectionScalarProduct (real_cvec_t *vec1*, **RegularAnisotropicGrid** * *grid1*, real_cvec_t *vec2*, **RegularAnisotropicGrid** * *grid2*, **InputData** * *data*, real_t *gamma* = `0.`)

Calculates the bilinear forms needed for the Opticom matrices.

Calculates the bilinear form needed for the Opticom matrix, vB^TLBw and vB^TDBw , where v and w may be on different grids and the product is calculated on the smallest regular anisotropic grid containing both the grids.

Parameters

vec1	vector v
grid1	grid that defines the basis functions belonging to v
vec2	vector w

grid2	grid that defines the basis functions belonging to w
data	input data needed to define the grid Laplacians
gamma	regularisation parameter, default is 0

Returns

2-dimensional array containing in ret[0] vB^TLBw and in ret[1] vB^TDBw

5.2.2.2 bool gridlineIterator (RegularAnisotropicGrid * *originGrid*, int * *originPosition*, int * *originIndex*, RegularAnisotropicGrid * *targetGrid*, int * *targetPosition*, int * *targetIndex*, int * *scale_diff*, int *transitionLevel*, int *dim*, int *face*)

Iterator over Points on grid faces.

Iterates over Points on grid faces, as needed for tensor interpolation. Necessary: originGrid.level <= targetGrid.level in every dimension WARNING: This function had a bug previously, concerning the return value of originIndex when using transition grids and going in a dimension not yet matching the target level. It is hopefully fixed with the construction "if (transitionLevel >= dim) .... " but if problems appear, check that first!

Parameters

originGrid	coarser grid
originPosition	position on coarser grid in each dimension, needs to be set in application
originIndex	index of origin point of relevant gridline (in targetGrids dimensions!)
targetGrid	finer grid
targetPosition	position on finer grid in each dimension, needs to be set in application
targetIndex	starting index of gridline that will in the end contain points to be set next
scale_diff	difference in scaling between coarser and finer grid
transitionLevel	if transition grids are used: level up to which transition grid equals target grid (set to dim if no transition grids)
dim	dimension into which we are moving (when calling externally usually 0)
face	drection of considered gridline

Returns

true, if we are still on grid, false if not

5.2.2.3 void SetupGridMatricesKNN (RegularAnisotropicGrid * *grid*, InputData * *data*, EMatrix * *eigenLHS*, EMatrix * *eigenRHS*, real_t *gamma* = 0.)

Setup of matrices for one grid.

Setting up the grid based matrices B^TLB and B^TDB in Eigen-Format, taking into account the sparse structure in L due to using knn-graph. $B_{ij} = \phi_j(y_i)$.

Parameters

grid	grid we want to calculate on
data	data defining the data-based graph Laplacian L
eigenLHS	output target for left hand side, B^TLB , in Eigen format
eigenRHS	output target for right hand side, B^TDB , in Eigen format
gamma	regularisation parameter, default is 0

5.2.2.4 real_cvec_t vectorProlongation (real_cvec_t *vector*, RegularAnisotropicGrid * *originGrid*, RegularAnisotropicGrid * *targetGrid*)

Prolongate vector from coarser to finer grid.

Interpolates the given vector (representing a multilinear function) on a larger grid via linear interpolation along the grid lines.

Parameters

vector	vector (function) to be extended
originGrid	grid the vector is originally defined on
targetGrid	(finer) grid the vector should be projected on

Returns

extended vector on targetGrid, memory is allocated here

5.3 MatVecHelper.h File Reference

`#include <vector> #include <fstream>`

Typedefs

- typedef double **real_t**
- typedef std::vector< int > **int_vec_t**
- typedef std::vector< real_t > **real_vec_t**
- typedef real_t ** **real_cmat_t**
- typedef real_t * **real_cvec_t**
- typedef int * **int_cvec_t**

Functions

- real_t **CVectorDistance** (real_cvec_t vec1, real_cvec_t vec2, int vecDim)
- void **printCVec** (real_cvec_t vec, int dim)
- void **printCVecPlain** (real_cvec_t vec, int dim)
- void **fprintCVecPlain** (std::ostream &out, real_cvec_t vec, int dim)
- void **fprintCMatPlain** (std::ostream &out, real_cmat_t mat, int dim1, int dim2)
- void **fprintCMatTransposedPlain** (std::ostream &out, real_cmat_t mat, int dim1, int dim2)

- void **printIntCVec** (int_cvec_t vec, int dim)
- void **printCMat2D** (real_cmat_t mat, int dim1, int dim2)

5.3.1 Detailed Description

Contains some typedefs and functions to work with standard matrices and vectors, especially based on C-arrays and those from the std::vector library

Author: Luisa Schwartz

5.3.2 Function Documentation

5.3.2.1 real_t CVectorDistance (real_cvec_t *vec1*, real_cvec_t *vec2*, int *vecDim*)

Returns euclidean distance of ctype vectors (arrays) with same dimension.

Parameters

vec1	$v_1 \in \mathbb{R}^d$
vec2	$v_2 \in \mathbb{R}^d$
vecDim	dimension d of vectors

Returns

$d(v_1, v_2)$ l_2 - distance of input vectors

5.3.2.2 void fprintCMatPlain (std::ostream & *out*, real_cmat_t *mat*, int *dim1*, int *dim2*)

Prints (real) CMat elementwise to file (no further formatting).

Parameters

out	stream to be printed to (usually filestream)
mat	$M \in \mathbb{R}^{m \times n}$, matrix to be printed
dim1	dimension 1 of matrix (m)
dim2	dimension 2 of matrix (n)

5.3.2.3 void fprintCMatTransposedPlain (std::ostream & *out*, real_cmat_t *mat*, int *dim1*, int *dim2*)

Prints transpose of (real) CMat elementwise to file (no further formatting).

Parameters

out	stream to be printed to (usually filestream)
mat	$M \in \mathbb{R}^{m \times n}$, matrix to be printed
dim1	dimension 1 of matrix (m)
dim2	dimension 2 of matrix (n)

5.3.2.4 void fprintCVecPlain (std::ostream & *out*, real_cvec_t *vec*, int *dim*)

Prints (real) CVec elementwise to file (no further formatting)

Parameters

vec	$v \in \mathbb{R}^d$, vector to be printed
vecDim	dimension d of vector

5.3.2.5 void printCMat2D (real_cmat_t *mat*, int *dim1*, int *dim2*)

Prints 2D C-Type Matrix to standard output with some formatting.

Parameters

mat	$M \in \mathbb{R}^{m \times n}$, matrix to be printed
dim1	dimension 1 of matrix (m)
dim2	dimension 2 of matrix (n)

5.3.2.6 void printCVec (real_cvec_t *vec*, int *dim*)

Prints (real) CVec to standard output with some formatting.

Parameters

vec	$v \in \mathbb{R}^d$, vector to be printed
vecDim	dimension d of vector

5.3.2.7 void printCVecPlain (real_cvec_t *vec*, int *dim*)

Prints (real) CVec elementwise to standard output (no further formatting).

Parameters

vec	$v \in \mathbb{R}^d$, vector to be printed
vecDim	dimension d of vector

5.3.2.8 void printIntCVec (int_cvec_t *vec*, int *dim*)

Prints (integer) CVec to standard output with some formatting.

Parameters

vec	$v \in \mathbb{N}^d$, vector to be printed
vecDim	dimension d of vector

Bibliography

[AF11] David Amsallem and Charbel Farhat. An online method for interpolating linear parametric reduced-order-models. *SIAM Journal on Scientific Computing*, 33(5):2169–2198, 2011.

[Alt12] Hans Wilhelm Alt. *Lineare Funktionalanalysis*. Springer Berlin Heidelberg, 2012.

[BBH13] Kerstin Bunte, Michael Biehl, and Barbara Hammer. A general framework for dimensionality reducing data visualization mapping. *Neural Computation*, 24:771 – 804, 2013.

[BG04] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. *Acta Numerica*, 13:1–123, 2004.

[BN03] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, June 2003.

[BN08a] Mikhail Belkin and Partha Niyogi. Convergence of laplacian eigenmaps. *preprint, short version NIPS 2008*, 2008.

[BN08b] Mikhail Belkin and Partha Niyogi. Towards a theoretical foundation for laplacian-based manifold methods. *Journal of Computer and System Sciences*, 74(8):1289–1308, 2008.

[BNS06] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularisation: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:2399–2434, 2006.

[BO27] Max Born and J. Robert Oppenheimer. Zur Quantentheorie der Molekeln. *Annalen der Physik*, 20:457–484, 1927.

[BO89] Ivo Babuška and John Edward Osborn. Finite element-Galerkin approximation of the eigenvalues and eigenvectors of selfadjoint problems. *Journal on Mathematics of Computation*, 52(186):275–297, April 1989.

[BO91] Ivo Babuška and John Edward Osborn. *Eigenvalue Problems*, pages 642 – 787. Elsevier Science Publishers B.V., North-Holland, 1991.

[Bof10] Daniele Boffi. Finite element approximation of eigenvalue problems. *Acta Numerica*, 19:1–120, 5 2010.

[BPV+04] Yoshua Bengio, Jean-Francois Paiement, Pascal Vincent, Olivier Delalleau, Nicolas LeRoux, and Marie Oiment. Out-of-sample extensions for lle, isomap, mds, eigenmaps and spectral clustering. *Advances in Neural Informations Processing Systems*, 16:177–184, 2004.

[Chu97] Fan R. K. Chung. *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92)*. American Mathematical Society, 1997.

[CL06a] Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and Computational Harmonic Analysis*, 21(1):5–30, 2006.

[CL06b] Ronald R Coifman and Stéphane Lafon. Geometric harmonics: A novel tool for multiscale out-of-sample extension of empirical functions. *Applied and Computational Harmonic Analysis*, 21(1):31–52, 2006.

[DH73] W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM J. Res. Dev.*, 17(5):420–425, September 1973.

[EHN00] Heinz Engl, Martin Hanke, and Andreas Neubauer. *Regularization of Inverse Problems*. Kluwer Academic Publishers, Dordrecht, NL, 2000.

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal*, 23(2):298–305, 1973.

[FPR+10] Mona Frommert, Dirk Pflüger, Thomas Riller, Martin Reinecke, Hans-Joachim Bungartz, and Torsten Ensslin. Efficient cosmological parameter sampling using sparse grids. *Monthly Notices of the Royal Astronomical Society*, 406:1177–1189, 2010.

[Gar98] Jochen Garcke. *Berechnung von Eigenwerten der stationären Schrödingergleichung mit der Kombinationstechnik*. Diplomarbeit, Institut für Angewandte Mathematik, Universität Bonn, 1998.

[Gar04] Jochen Garcke. *Maschinelles Lernen durch Funktionsrekonstruktion mit verallgemeinerten dünnen Gittern*. Doktorarbeit, Institut für Numerische Simulation, Universität Bonn, 2004.

[Gar06] Jochen Garcke. Regression with the optimised combination technique. In W. Cohen and A. Moore, editors, *Proceedings of the 23rd ICML '06*, pages 321–328, New York, NY, USA, 2006. ACM Press.

[Gar08] Jochen Garcke. An optimised sparse grid combination technique for eigenproblems. In *Proceedings of ICIAM 2007*, volume 7 of *PAMM*, pages 1022301–1022302, 2008.

[Gar13] Jochen Garcke. Sparse grids in a nutshell. In Jochen Garcke and Michael Griebel, editors, *Sparse grids and applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 57–80. Springer, 2013.

[Gar14] Jochen Garcke. An optimised sparse grid combination technique for eigenvalue problems. Presentation at SGA 2014, Stuttgart, 2014.

[GG00] Jochen Garcke and Michael Griebel. On the computation of the eigenproblems of hydrogen and helium in strong magnetic and electric fields with the sparse grid combination technique. *Journal of Computational Physics*, 165(2):694–716, 2000.

[GGT01] Jochen Garcke, Michael Griebel, and Michael Thess. Data mining with sparse grids. *Computing*, 67:225–253, 2001.

[GH09] Jochen Garcke and Markus Hegland. Fitting multidimensional data using gradient penalties and the sparse grid combination technique. *Computing*, 84:1–35, 2009.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[GK06] Evarist Giné and Vladimir Koltchinskii. Empirical graph laplacian approximation of laplace-beltrami operators: Large sample results. *IMS Lecture Notes-Monograph Series*, 51:238–259, 2006.

[GL12] Gene H Golub and Charles F Van Loan. *Matrix Computations*. The John Hopkins University Press, 2012.

[GLMH12] Andrej Gisbrecht, Wouter Lueks, Bassam Mokbel, and Barbara Hammer. Out-of-sample kernel extensions for nonparametric dimensionality reduction. In *ESANN 2012 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 531–535, 2012.

[GSZ92] Michael Griebel, Michael Schneider, and Christoph Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, North Holland, 1992.

[HA85] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[HAL06] Matthias Hein, Jean-Yves Audibert, and Ulrike Luxburg. From graphs to manifolds – weak and strong pointwise consistency of graph laplacians. In *Proceedings of the 19th Annual Conference on Learning Theory (COLT)*, pages 50 – 64, 2006.

[HAL07] Matthias Hein, Jean-Yves Audibert, and Ulrike Luxburg. Graph laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8:1325–1368, 2007.

[Heg02] Markus Hegland. Additive sparse grid fitting. In *Proceedings of the Fifth International Conference on Curves and Surfaces, Saint-Malo*, pages 209–218, 2002.

[HGC07] Markus Hegland, Jochen Garcke, and Vivien Challis. The combination technique and some generalisations. *Linear Algebra and its Applications*, 420(2–3):249–275, 2007.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[Kat76] Tosio Kato. *Perturbation Theory for Linear Operators*. Springer-Verlag Berlin Heidelberg New York, 1976.

[KH14] Christoph Kowitz and Markus Hegland. An opticom method for computing eigenpairs. In Jochen Garcke and Dirk Pflüger, editors, *Sparse grids and applications - Munich 2012*, volume 97 of *Lecture Notes in Computational Science and Engineering*, pages 239–253. Springer, 2014.

[Laf04] Stéphane Lafon. *Diffusion Maps and Geometric Harmonics*. PhD thesis, Yale University, 2004.

[LBB08] Ulrike Luxburg, Mikhail Belkin, and Olivier Bousquet. Consistency of spectral clustering. *The Annals of Statistics*, 36(2):555–586, 2008.

[Llo82] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:129–137, 1982.

[Lux07] Ulrike Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[MHL09] Markus Maier, Matthias Hein, and Ulrike Luxburg. Optimal construction of k-nearest neighbor graphs for identifying noisy clusters. *Theoretical Computer Science*, 410:1749–1764, 2009.

[Moh91] Bojan Mohar. The laplacian spectrum of graphs. In *Graph Theory, Combinatorics, and Applications*, pages 871–898. Wiley, 1991.

[MS01] Marina Meila and Jianbo Shi. A random walks view of spectral segmentation. In *8th International Workshop on Artificial Intelligence and Statistics (AISTATS)*, 2001.

[NJW01] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856. MIT Press, 2001.

[NLCK06] Boaz Nadler, Stéphane Lafon, Ronald Coifman, and Ioannis G Kevrekidis. Diffusion maps, spectral clustering and reaction coordinates of dynamical systems. *Applied and Computational Harmonic Analysis*, 21(1):113–127, 2006.

[Pfl10] Dirk Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Dissertation, Institut für Informatik, Technische Universität München, München, February 2010.

[Pol09] Eric Polizzi. Density-matrix-based algorithm for solving eigenvalue problems. *Physical Review B*, 79:115112, Mar 2009.

[PPB11] Benjamin Peherstorfer, Dirk Pflüger, and Hans-Joachim Bungartz. A sparse-grid-based out-of-sample extension for dimensionality reduction and clustering with laplacian eigenmaps. In Dianhui Wang and Mark Reynolds, editors, *AI 2011: Advances in Artificial Intelligence*, volume 7106 of *Lecture Notes in Computer Science*, pages 112–121, Berlin Heidelberg, December 2011. Murdoch University, Western Australia, Springer.

[PT14] Eric Polizzi and Ping Tak Peter Tang. Feast as a subspace iteration eigensolver accelarated by approximate spectral projection. *SIAM Journal on Matrix Analysis and Applications*, 35:354 – 390, 2014.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RBD10] Lorenzo Rosasco, Mikhail Belkin, and Ernesto De Vito. On learning with integral operators. *Journal of Machine Learning Research*, 11:905–934, 2010.

[Saa11] Yousef Saad. *Numerical Methods for Large Eigenvalue Problems*. Classics In Applied Mathematics. SIAM, 2011.

[SB10] Kaushik Sinha and Mikhail Belkin. Semi-supervised learning using sparse eigenfunction bases. In *Advances in Neural Information Processing Systems 22*. 2010.

[SM00] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[Smo63] Sergey A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Mathematics, Doklady*, 4:240–243, 1963.

[TSL00] Joshua B. Tenenbaum, Vin Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, 2000.

[Wei99] Yair Weiss. Segmentation using eigenvectors: a unifying view. In *Proceedings of the IEEE Internation Conference on Computer Vision*, pages 975–982, 1999.

[Zen91] Christoph Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel algorithms for partial differential equations, Proceedings of the Sixth GAMM-Seminar, Kiel, 1990*, volume 31 of *Notes in Numerical Fluid Mechanics*, pages 241–251. Vieweg Braunschweig, 1991.