Refinement and Coarsening of Online-Offline Data Mining Methods with Sparse Grids

Bachelor’s Thesis

by

Adrian Sieler

Supervisor: Prof. Dr. Hans-Joachim Bungartz
Advisor: Kilian Röhner, M.Sc.
Submission date: 15.03.2016
I hereby declare that this thesis is my own work and that no other sources have been used except those clearly indicated and referenced.

Garching bei München, 15.03.2016

Adrian Sieler
Zusammenfassung

Diese Bachelorarbeit beschäftigt sich mit einem adaptiven Dünngitter-Dichteschätzungsverfahren. Im Kontext eines zur Verfügung stehenden Datenstromes, z.B. im Fall von Online Data Mining, wird eine Offline/Online Teilung im Bezug auf das Schätzungsverfahren eingeführt. Die geschätzte Dichte wird über die Lösung eines linearen Gleichungssystems festgelegt, welches durch das verwendete Gitter und die zugrundeliegenden Daten bestimmt wird. Die Cholesky-Zerlegung wird als eine mögliche Faktorisierung zur Lösung des Gleichungssystems präsentiert. Um keine erneute Matrizenzerlegung anwenden zu müssen, sobald das Dünggitter verfeinert und/oder vergrößert wird, werden Algorithmen entwickelt, um Modifikationen direkt auf dem Cholesky-Faktor vornehmen zu können.

Abstract

This thesis deals with an adaptive sparse grid density estimation method. Given a data-stream, e.g. via Online Data Mining, an Offline/Online splitting corresponding to this density estimation approach is introduced. To obtain the density declaring coefficients a system of linear equations, determined by the sparse grid and underlying data, needs to be solved. To do so, the Cholesky decomposition is introduced as a possible factorization of the linear problem. In order to avoid a new factorization if the grid is refined and/or coarsened, algorithms are presented to perform modifications directly on the Cholesky factor.
## Contents

1 Introduction

2 Theory
   2.1 Density estimation ........................................... 3
   2.2 Sparse grids ............................................... 3
   2.3 Adaptivity .................................................... 8
   2.4 Grid based density estimation ............................. 9
   2.5 Offline/Online splitting .................................. 11
   2.6 Cholesky decomposition .................................... 12
   2.7 Cholesky modifications ...................................... 14
      2.7.1 Rank one update ...................................... 14
      2.7.2 Rank one downdate .................................... 15
      2.7.3 Permutations .......................................... 17
      2.7.4 Add a row/column ...................................... 18
      2.7.5 Remove first row/column .............................. 19

3 Algorithms and methods ........................................... 20
   3.1 Factorization ............................................... 20
   3.2 Refinement ................................................. 20
   3.3 Coarsening .................................................. 22
   3.4 Modify regularization parameter ......................... 26
      3.4.1 Increase $\lambda$ ...................................... 26
      3.4.2 Decrease $\lambda$ ....................................... 27
   3.5 Adaptive data-streaming .................................. 29

4 Implementation ..................................................... 30
   4.1 libtool ...................................................... 30
   4.2 clustc ....................................................... 31
   4.3 SG++ .......................................................... 31

5 Evaluation of algorithms ......................................... 32
   5.1 Offline step ................................................ 32
   5.2 Online step .................................................. 33
   5.3 Adaptivity .................................................... 34
      5.3.1 Coarsening ............................................. 34
      5.3.2 Refinement ............................................. 36
   5.4 Cross-validation ............................................ 37
Contents

6 Application 39
   6.1 Data-stream based classifier ............................... 39
   6.2 Classification of the DR10 Dataset .......................... 40

7 Conclusion 48

8 Appendix 49
   8.1 Examination of algorithms .................................... 49
   8.2 Examination of parameter c .................................... 52
   8.3 DR10 classification measurements .............................. 53
1 Introduction

Density estimation is one of the most popular unsupervised learning problems in statistics and data mining. Based on a data set \( S = \{ x_1, \ldots, x_M \} \subset \mathbb{R}^d \) of samples drawn from an unknown distribution \( p(X) \) of a random variable \( X \), we want to determine a representative estimation \( \hat{p} \) of the underlying probability density function \( p \).

A comprehensive theory exists regarding this topic [8]. Most of the current state of the art techniques like kernel density estimation lack feasibility in higher dimensions corresponding to a "big data" sample (Sec. 2.1).

But nowadays, experiments - e.g. collecting gene expressions of humans or evaluating sensor information - produce huge amounts of data, that probably need to be processed immediately.

In this thesis we are going to cope this problem via a sparse grid density estimation approach (Sec. 2.4) in combination with a data-stream framework, to enable computability for large data sets (Sec. 3.5).

Often when we are in a big data setting, it is not feasible to process all given data at once or in Online Data Mining, data is not accessible all the time. Therefore, we introduce an Offline/Online splitting according the sparse grid density estimation approach, to account for this problem (Sec. 2.5). The main idea is to compensate the costly Offline step by repeating the Online phase many times. Especially in data-streaming this approach makes sense. The Offline step can be interpreted as providing all required tools to directly solve for the density declaring coefficients in the Online step, if new data becomes available.

In our case the Offline step is to perform a decomposition of a matrix which only depends on the current grid and not on the given data. Whereas the Online step requires to solve a system of linear equations utilizing the calculated factorization.

However, we start with an a priori sparse grid (Sec. 2.2), that may not adequately represent the characteristics of the underlying problem. To handle this issue, we apply coarsening and refinement to the current grid (Sec. 2.3). This leads to changes in the grid structure, since certain grid points are removed and some new are integrated. Looking back to our Offline step, that means that our matrix changes and therefore requires a new decomposition. This would lead to infeasible runtimes especially in larger grid settings.

To account for this problem, we develop algorithms (Sec. 3) based on the Cholesky decomposition (Sec. 2.6) and related modifications (Sec. 2.7), to enable an adaptive factorization if grid changes occur. These methods should prevent a new Cholesky decomposition and therefore a full new Offline step (Sec. 3.1), if the grid is refined and/or coarsened (Sec. 3.2, 3.3). We are also addressing the challenge to change the regularization parameter \( \lambda \) directly in the current factorization (Sec. 3.4).
1 Introduction

Section 4 structures the implementation of the proposed methods into existing data mining frameworks. In Section 5 the algorithms are stressed due to different grid settings, to derive recommendations for use.

We will finish this thesis with extending our data-stream based density estimation into a data-stream based classifier (Sec. 6.2) and applying it to real world astronomical data set (Sec. 6.2).

At the Chair of Scientific Computing at the Technische Universität München, a toolbox called \texttt{SG++} was developed in order to provide a straightforward approach to work with sparse grids and different applications of them [1]. The underlying work extends the data mining toolboxes \texttt{libtool} and \texttt{clustc}, mainly depending on data structures and methods provided by \texttt{SG++}, for an adaptive and efficient density estimation method.
2 Theory

In this thesis we denote vectors with bold lower-case letters (e.g. \( \mathbf{a} \)) and matrices with bold upper-case letters (e.g. \( \mathbf{A} \)). The \( i \)-th component of a vector \( \mathbf{a} \) is denoted with \( a_i \), the element in the \( i \)-th row and the \( j \)-th column of a matrix \( \mathbf{A} \) is \( A_{ij} \).

2.1 Density estimation

Density estimation methods can basically be separated into parametric and non-parametric approaches. Whereas parametric models can be characterized by the fact that the number of parameters is fixed before training. These models often follow strong model assumptions. One prominent example is a Mixture of Gaussians. The corresponding parameters, denoted by means, (co)variances and mixing parameters, are estimated with the expectation-maximization (EM) algorithm \[8\]. Such models achieve good results if the assumptions are met. But if the underlying problem can’t be characterized through a combination of basic density functions, parametric models won’t perform good. Non-parametric models are the key to overcome this restriction. The most common and frequently used method is kernel density estimation \[8\]. Based on a given data sample \( \mathcal{S} = \{x_1, \ldots, x_m\} \) the underlying density is estimated via,

\[
\hat{p}(x) = \frac{1}{m} \sum_{i=1}^{m} K \left( \frac{x - x_i}{\sigma^2} \right),
\]

(2.1)

a linear combination of kernel functions \( K \) centered at the data points \( x_i \in \mathcal{S} \). Due to the high flexibility of this density estimation approach, the choice of the kernel functions and the bandwidth \( \sigma > 0 \) are a delicate matter and probably need to be determined through cross validation, see Figure 2.1. Since the number of evaluations to solve for \( \hat{p} \) depends on the amount of data points \( \mathcal{S} \), computational problems can occur if the sample size is increased. One way to reduce the amount of evaluations is to divide the data into a small number of bins and place a kernel at each bin. However, if the dimension of the underlying data increases, the number of bins increases exponentially. Because of the high sensibility due to the model parameters and the expensive evaluation of the estimator (Eq. 2.1) this approach is only feasible in up to four dimensions \[2\].

2.2 Sparse grids

To overcome the "curse of dimensionality" to some extend, a sparse grid based density estimation approach is introduced, see Sec. 2.5. The idea is to discretize the underlying
domain of the data with sparse grids, to prevent a full grid approach with $N$ grid points in every dimension. The resulting exponential growth of the $N^d$ grid points in the full grid setting would cause problems in terms of computational effort.

The following section provides a short overview of basic concepts of sparse grids and helps to understand the presented grid based density estimation method (Sec. 2.4). A more detailed description can be found in [1].

Before reaching the core of sparse grids, it is necessary to introduce some notations. When dealing with $d$-dimensional multi-indices $l$ and $k$, relational operators are used component wise,

$$1 \leq k \leftrightarrow l_j \leq k_j, \ 1 \leq j \leq d.$$  \hspace{1cm} (2.2)

Required norms are the $l_1$-norm $|l|_1$, maximum-norm $|l|_\infty$,

$$|l|_1 := \sum_{j=1}^{d} l_j \quad |l|_\infty := \max_{1 \leq j \leq d} |l_j|$$  \hspace{1cm} (2.3)

as well as the $L_p$-Norm for $d$-dimensional functions $f : \Omega \rightarrow \mathbb{R}$

$$\|f\|_{L_p}^p := \int_{\Omega} |f(x)|^p dx.$$  \hspace{1cm} (2.4)

The underlying density function needs to be of the form $f : \Omega \rightarrow \mathbb{R}$ with $\Omega := [0, 1]^d$ and is interpolated with sparse grids by using hierarchical basis functions. If the given domain of the data is not $\Omega$ it can be easily scaled into the required range. The approximation of $f$ is given by

$$f(x) \approx u(x) := \sum_i \alpha_i \varphi_i(x)$$  \hspace{1cm} (2.5)

with coefficients $\alpha_i$ and basis functions $\varphi_i$, see Figure 2.2.
2 Theory

(a) Sparse grid function approximation

Figure 2.2: (a) depicts a piecewise linear interpolation $u(x)$ of a function using weighted hierarchical basis functions (b) (Eq. 2.11), figure taken from [1].

Sparse grids depend on a hierarchical decomposition of the underlying approximation space. Assuming that the function $f$ is equal to zero at the domain boundaries, the standard hat function

$$\varphi(x) = \max(1 - |x|, 0)$$  \hspace{1cm} (2.6)

is transferred via dilatation and translation into the one-dimensional hat basis functions

$$\varphi_{l,i} := \varphi(2^lx - i)$$  \hspace{1cm} (2.7)

which depend on a level $l$ and an index $i$, $0 < i < 2^l$. The introduced basis functions have local support and are centered at grid points $x_{l,i} = 2^{-li}$ at which $f$ is interpolated. Depending on a given level $l$ the hierarchical index set is defined as

$$I_l := \{ i \in \mathbb{N} : 1 \leq i \leq 2^l - 1, i \text{ odd} \}.$$  \hspace{1cm} (2.8)

The concept of one-dimensional basis functions is extended to higher dimensions via a tensor product approach:

$$\varphi_{I_l}(x) := \prod_{j=1}^{d} \varphi_{l_j,i_j}(x_j)$$  \hspace{1cm} (2.9)

with $d$-dimensional multi-indices $I_l$ and $i$ indicating level and index for each dimension. For a given set of levels $I$, the one-dimensional hierarchical index set notation can be transferred straightforward into the multi-dimensional case:

$$I := \{ i : 1 \leq i_j \leq 2^{l_j} - 1, i_j \text{ odd}, 1 \leq j \leq d \}.$$  \hspace{1cm} (2.10)

Given the hierarchical index set and specific basis functions, the set of hierarchical subspaces $W_l$ is obtained, see Figure 2.3:

$$W_l := \text{span} \{ \varphi_{I_l}(x) : i \in I_l \}.$$  \hspace{1cm} (2.11)
Figure 2.3: Basis functions of the subspace \( W_1 \) (Eq. 2.11) for \( |l|_\infty \leq 3 \) in two dimensions, figure taken from [1].
2 Theory

The space of piecewise linear functions on a full grid, with mesh width \( h_n = 2^{-n} \) in each dimension and fineness level \( n \), is given by a direct sum of the corresponding \( W_1 \)

\[
V_n = \bigoplus_{|l|_\infty \leq n} W_1
\]  

(2.12)

leading to a full grid with \((2^n - 1)^d\) grid points. Leaving out those subspaces of the full grid space \( V_n \) with many basis functions of small support, the sparse grid space is obtained by:

\[
V_n^{(1)} := \bigoplus_{|l|_1 \leq n+d-1} W_1.
\]  

(2.13)

A two-dimensional example can be found in Figure 2.4. So we obtain sparse grid interpolants \( u(x) \in V_n^{(1)} \) of the form:

\[
u(x) = \sum_{|l|_1 \leq n+d-1, \in I_l} \alpha_l \varphi_{l_1}(x).
\]  

(2.14)

In the following, error decays of full and sparse grids interpolants in comparison with the amount of grid points, given certain smoothness conditions, will be discussed, for details see [1, 2]. With

\[
D^l f = \frac{\delta|l|_1}{\delta x_{l_1} \cdots \delta x_{l_d}}
\]  

(2.15)

we define the space of functions with bounded weak mixed derivations up to order two as

\[
H^2_{mix} = \{ f : \mathbb{R} \to \mathbb{R} : D^l f \in L_2(\Omega), |l|_\infty \leq 2, f|_{\delta \Omega} = 0 \}.
\]  

(2.16)

Given a function \( f \in H^2_{mix} \) and its full grid interpolant \( f \) in the space of piecewise \( d \)-linear functions with \( h_n = 2^{-n} \) as corresponding mesh width of the space \( V_n \), the asymptotic error

\[
\| f(x) - f(x) \|_{L_2} \in \mathcal{O}(h_n^2)
\]  

(2.17)

is obtained. But since the number of grid points of the full grid space \( V_n \) is in \( \mathcal{O}(2^{nd}) \) this approach suffers from the curse of dimensionality. Consider now a sparse grid interpolant \( f_S \in V_n^{(1)} \). The associated asymptotic error

\[
\| f(x) - f_S(x) \|_{L_2} \in \mathcal{O}(h_n^2 n^{d-1})
\]  

(2.18)

is only slightly downgraded compared to the error decay of the full grid interpolant. But the sparse grid approach pays off in terms of reduced computational effort, observing that the amount of grid points has been significantly reduced to \( \mathcal{O}(2^n n^{d-1}) \).
Figure 2.4: The two-dimensional subspaces $W_l$ up to level $l = 3$ (a) and the corresponding sparse grid for the sparse grid space $V_3^{(1)}$ (b) (Eq. 2.13). To obtain a full grid of level 3 (Eq. 2.12), the white subspaces have to be used as well [1].

### 2.3 Adaptivity

Since the sparse grid structure (Eq. 2.13) defines an a priori selection of grid points with optimality properties for certain functions (Eq. 2.16), no further knowledge about the function $f$ in question is used.

In many cases some more characteristics about $f$ are known. This knowledge can be utilized to influence the structure of the sparse grid adaptively, to carry out the special requirements of the corresponding problem.

To refine a selected grid point, all $2d$ children (grid points with corresponding supports of one half of the support associated with the chosen grid point) in the hierarchical structure are added to the current grid, if they haven’t been created yet (Figure 2.5).

Since most algorithms working on sparse grids depend on traversals of the hierarchical structure of basis functions, it is necessary to ensure the existence of all parent grid points as well (Figure 2.5), for details see [1].

In order to deal with high dimensional data, even the amount of grid points in an adaptive sparse grid structure should be controlled. One approach to get rid of superfluous grid points is to coarsen the current grid, for further details see [2].

A straightforward but robust and widely used adaptivity criterion is based on the hierarchical coefficients $\alpha$ (Eq. 2.14)(Eq. 2.29). Consider the coefficients with the largest absolute values for refinement and in contrast the ones with smallest absolute values for
2 Theory

![Figure 2.5](image)

Figure 2.5: The regular grid of level two in (a) is refined to obtain the sparse grid in (b). Another refinement step is performed and since most sparse grid algorithms require the existence of the hierarchical ancestors the gray points are created as well (c), see [1].

Because adaptivity is highly application-dependent, a good refinement criterion takes the context of the underlying problem into account. But there is no free lunch [8], for each refinement criterion it is possible to construct a problem where it fails.

2.4 Grid based density estimation

The density estimation method introduced in [5] is a promising approach to overcome the drawbacks of high sensitivity with respect to parameters and long runtimes for large data sets (Sec. 2.1).

Given the data set $S = \{x_1, \ldots, x_m\}$, the concept is to start with a highly overfitted estimation $f_e \in L_2$ and use spline smoothing to obtain a more generalized density approximation $\hat{f}$. So the optimal $\hat{f}$ given a function space $V$ is represented by

$$\hat{f} = \arg\min_{f \in V} \int_{\Omega} (f(x) - f_e(x))^2 dx + \lambda \|\Lambda f\|^2_{L_2}. \quad (2.19)$$

Whereby the left term ensures that $\hat{f}$ fits the initial guess $f_e$ properly and the right term $\|\Lambda f\|^2_{L_2}$, as a regularization or penalty term, is imposing a smoothness constraint. Assuming that $\Lambda$ is a linear functional. The regularization parameter $\lambda \in \mathbb{R}_+$ controls the trade-off between fidelity and smoothness.

Using the sparse grid space $V_l^{(1)}$ (Eq. 2.13) of level $l$ as function space, one can see the main advantage of this approach. The density function $\hat{f}$ is discretized with basis functions centered at grid points instead with kernels centered at data points. While binning for kernel density estimation would lead to a full grid approach, sparse grids are employed. Therefore, this method is scalable in the number of data points and overcomes the ”curse of dimensionality“ to some extend.
2 Theory

The variational equation associated with Eq. 2.19 becomes

\[
\int_{\Omega} s(x)(f(x) - f_\epsilon(x))dx + \lambda \int_{\Omega} \Lambda f(x) \cdot \Lambda s(x)dx = 0, \quad \forall s \in V,
\]  

(2.20)

for details see [5]. Given the function space \( V \), \( \lambda \) and the penalty operator \( \Lambda \), we are going to show how the optimal estimator \( f \), fulfilling Eq. 2.20, is obtained.

Eq. 2.20 can be rewritten as

\[
\int_{\Omega} s(x)f(x)dx + \lambda \int_{\Omega} \Lambda f(x) \cdot \Lambda s(x)dx = \int_{\Omega} s(x)f(x)\epsilon(x)dx .
\]

(2.21)

Declare \( f_\epsilon = \frac{1}{m} \sum_{i=1}^{m} \delta_{x_i} \) as a sum of Dirac delta functions centered at the data points \( x_i \) to make sure \( f_\epsilon \) is close to the given data. With this choice of \( f_\epsilon \), \( \int_{\Omega} s(x)f(x)\epsilon(x)dx \) turns into:

\[
\int_{\Omega} \frac{1}{m} \sum_{i=1}^{m} \delta_{x_i} \cdot s(x)dx .
\]

(2.22)

Since the sum is finite, integral and sum can be exchanged

\[
\frac{1}{m} \sum_{i=1}^{m} \int_{\Omega} \delta_{x_i} \cdot s(x)dx .
\]

(2.23)

The remaining integral can now be interpreted as Dirac integral and consequently the following is obtained:

\[
\frac{1}{m} \sum_{i=1}^{m} s(x_i) .
\]

(2.24)

Putting everything together Eq. 2.20 turns into:

\[
\int_{\Omega} s(x)f(x)dx + \lambda \int_{\Omega} \Lambda f(x) \cdot \Lambda s(x)dx = \frac{1}{m} \sum_{i=1}^{m} s(x_i), \quad \forall s \in V .
\]

(2.25)

Setting \( V \) as the sparse grid space \( V_1^{(1)} \) of level \( l \in \mathbb{N} \), the optimal \( f(x) \) can be posed as a linear combination of the hierarchical basis functions \( \Phi = \{\varphi_1, \ldots, \varphi_N\} \) of \( V_1^{(1)} \)

\[
f(x) = \sum_{i=1}^{N} \alpha_i \cdot \varphi_i(x) .
\]

(2.26)

Since \( \Phi \) spans \( V_1^{(1)} \) and \( f(x) \) is a linear combination of basis functions, the Galerkin approach [5] can be applied. Therefore \( m \) equations are obtained with \( s(x) \) set to the basis functions \( \varphi_1 \) to \( \varphi_N \). Substituting \( f(x) \) and \( s(x) \) back into (2.21) leads to:

\[
0 = \int_{\Omega} \sum_{i=1}^{N} \alpha_i \varphi_i \varphi_j dx + \lambda \int_{\Omega} \Lambda \left( \sum_{i=1}^{N} \alpha_i \varphi_i \right) \cdot \Lambda \varphi_j dx - \frac{1}{m} \sum_{i=1}^{m} \varphi_j(x_i), \quad 1 \leq j \leq N .
\]

(2.27)
2 Theory

Using that \( \Lambda \) is a linear functional and the sums are finite, the equation can be transformed into:

\[
0 = \sum_{i=1}^{N} \alpha_i \int_{\Omega} \varphi_i \varphi_j \, dx + \lambda \sum_{i=1}^{N} \alpha_i \int_{\Omega} \Lambda \varphi_i \cdot \Lambda \varphi_j \, dx - \frac{1}{m} \sum_{i=1}^{m} \varphi_j(x_i), \quad 1 \leq j \leq N.
\]  

(2.28)

Using \((v, w)_{L^2} = \int_{\Omega} v(x) \cdot w(x) \, dx\) as the \(L^2\)-inner-product on the Hilbert space \(\mathbb{R}\) with standard multiplication as the inner product, the final linear equation is obtained:

\[
(R + \lambda C) \alpha = b
\]

(2.29)

with \(R_{ij} = (\varphi_i, \varphi_j)_{L^2}\), \(C_{ij} = (\Lambda \varphi_i, \Lambda \varphi_j)_{L^2}\) and \(b_i = \frac{1}{m} \sum_{j=1}^{m} \varphi_i(x_j)\). Solving this equation with respect to \(\alpha\), a density function based on sparse grids is obtained.

In the following the regularization matrix \(C\) is assumed to be the identity matrix \(I\). Implying \(\sum_{i=1}^{m} \alpha_i^2\) as regularization term \(\|\Lambda f\|_{L^2}^2\), what is limiting the growth of the hierarchical coefficients.

2.5 Offline/Online splitting

The system of linear equations we need to solve (Eq. 2.29) to receive the density declaring coefficients, allows us to split the computational procedure of the sparse grid density estimation into an Online and Offline phase.

As one can see, the given data points influence only the right hand side of Eq. 2.29 while the system matrix \(R + \lambda C\) depends just on the current grid. So it is possible to precompute and and store the system matrix in decomposed form, denoted as Offline phase. When new data points arrive in the Online phase, the preprocessed matrix is loaded and the corresponding system of linear equations is solved. Such an Offline/Online splitting pays off if the costly Offline phase is compensated by repeating the Online phase many times, e.g. if a (real-time) application requires a density adaption immediately after new training data has been made available. Online learning, data-stream mining and cross-validation are practical examples that are described later, see Sec. 3.5.

To solve for \(\alpha\) in the Online phase, \((R + \lambda C)^{-1} b\) (Eq. 2.29) needs to be calculated. Therefore a matrix factorization of the corresponding system matrix is a good choice, since efficient solving methods are known, for details see [6].

The usage of Eigendecomposition and LU decomposition in the Offline phase has been proposed in [2]. Especially the Eigendecomposition has the main advantage that the decomposition doesn’t need to be performed from scratch if the regularization parameter \(\lambda\) is changed. This is not possible for the LU decomposition [2]. But in case the underlying grid is altered, both decomposition methods require a new factorization and therefore a Offline step. Since the proposed density estimation approach is not only grid based but also adaptive, i.e. grid modifications like refinement (Sec. 2.3) and coarsening (Sec. 2.3) can easily be adopted into Eq. 2.29 (Sec. 3), an adaptive factorization is desirable.
To reduce the computational effort of the recurring Offline phases and to tackle the problem of adaptive factorization, algorithms (Sec. 3) based on the Cholesky decomposition and corresponding Cholesky modifications are introduced.

### 2.6 Cholesky decomposition

**Theorem 2.1.** (Cholesky decomposition) For every symmetric positive-definite matrix $A$ exists a unique decomposition of the form

$$A = LL', \quad (2.30)$$

where $L$ is a lower triangular matrix with real and positive diagonal entries, and $L'$ denotes the transpose of $L$.

Before presenting the proof which contains the algorithm to derive the Cholesky decomposition, it is necessary to show that the system matrix $R + \lambda I$ (Eq. 2.29) is symmetric and positive definite to meet the assumptions of Theorem 2.1.

**Definition 2.2.** (Gramian matrix) Given a bilinear form on a n-dimensional K-vector space $V$ with basis $(v_1,\ldots,v_n)$, $\langle \cdot, \cdot \rangle : V \times V \to K$, $(v,w) \to \langle v,w \rangle$. The matrix

$$G := \begin{pmatrix} \langle v_1, v_1 \rangle & \cdots & \langle v_1, v_n \rangle \\ \vdots & \ddots & \vdots \\ \langle v_n, v_1 \rangle & \cdots & \langle v_n, v_n \rangle \end{pmatrix}, \quad (2.31)$$

is called the bilinear form $\langle \cdot, \cdot \rangle$ associated Gramian matrix.

**Corollary 2.3.** The bilinear form $\langle \cdot, \cdot \rangle : V \times V \to K$ is an inner product if and only if the associated Gramian matrix is symmetric and positive definite.

**Proof.** See [7].

Using the fact that $R_{ij} = (\varphi_i, \varphi_j)_{L_2}$ is defined on the $L_2$-inner-product on the Hilbert space $\mathbb{R}$ with standard multiplication, Corollary 2.3 can be applied to proof $R$ is symmetric and positive definite. Since $\lambda I$ is a diagonal matrix with positive entries it is obvious symmetric and positive definite as well. Thus, the corresponding system matrix $R + \lambda I$ is symmetric and positive definite.

The prerequisites to perform the Cholesky decomposition are fulfilled. In the following Theorem 2.1 is proven [6].

**Proof.** We construct the Cholesky decompositions $A_k = L_k L_k'$ of the submatrices of $A = A_m \in \mathbb{R}^{m \times m}$ stepwise,

$$A = \begin{pmatrix} A_k & | & B \\ \hline B & | & C \end{pmatrix}, \quad \text{with } A_k \in \mathbb{R}^{k \times k}. \quad (2.32)$$
Following partitions are used

$$A_k = \begin{pmatrix} A_{k-1} & a_k \\ a_k & \beta_k \end{pmatrix}, \quad L_k = \begin{pmatrix} L_{k-1} \\ l_k \\ \lambda_k \end{pmatrix}, \quad L'_k = \begin{pmatrix} L_{k-1} \\ l_k \end{pmatrix}. \quad (2.33)$$

In the k-th step the row \((l'_k, \lambda_k)\) of \(L\) is computed. At the same time we will inductively proof that \(L_k\) is a uniquely defined lower triangular matrix with positive diagonal entries. Multiplying

$$A_k = L_k L'_k$$

using the introduced partitions yields

$$A_k = (L_{k-1} - l'_k a_k), \quad L'_k = (L_{k-1} - l_k \lambda_k). \quad (2.34)$$

The first equation is the factorization of step \(k - 1\), the third is the adjunction of the second. Resolving the second and fourth equation into the desired parameters provides

$$l_k = L_{k-1}^{-1} a_k \quad \text{(forward substitution)} \quad \lambda_k = \sqrt{\beta_k - l'_k l_k}. \quad (2.35)$$

According to the induction hypothesis, \(L_{k-1}\) is a unique lower triangular matrix with positive diagonal entries and in particular invertible, therefore \(l_k\) (Eq. 2.35) is clearly unique as well (if \(k = 1\) do nothing). To obtain a (unique) positive square root to compute \(\lambda_k\) (Eq. 2.32) it remains to show

$$\beta_k - l'_k l_k > 0. \quad (2.37)$$

The submatrices \(A_k\) are positive definite as well

$$x' A_k x = \begin{pmatrix} x' \\ 0 \end{pmatrix} A \begin{pmatrix} x \\ 0 \end{pmatrix} > 0 \quad \text{with} \quad 0 \neq x \in \mathbb{R}^k. \quad (2.38)$$

Given \(x_k\) solving \(L'_{k-1} x_k = -l_k\), the following is obtained

$$0 < \begin{pmatrix} x_k \\ 1 \end{pmatrix}' A_k \begin{pmatrix} x_k \\ 1 \end{pmatrix} = (x_k' | 1) \begin{pmatrix} L_{k-1} L'_{k-1} \\ l_k l'_{k-1} \end{pmatrix} \begin{pmatrix} 1 \\ x_k \\ 1 \end{pmatrix}$$

$$= x_k' L_{k-1} L'_{k-1} x_k + x_k' l_k l'_{k-1} x_k + l'_k l'_{k-1} + l_k = \beta_k - l'_k l_k. \quad (2.39)$$

We conclude \(\lambda_k > 0\), consequently \(L_k\) is a unique positive lower triangular matrix with positive diagonal entries as well. The induction step is performed, concluding the proof. With a little more effort, the proof can be generalized for Hermitian positive-definite matrices.
2.7 Cholesky modifications

The following algorithms are all implemented by using orthogonal transformations called Givens rotations. Therefore, we start with a definition of these transformations [3].

**Definition 2.4.** (Givens rotation) A plane rotation in the \((i,j)\)-plane is a matrix of the form

\[
U(i,j,\theta) := \begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \cdots & \vdots & \cdots & \vdots \\
i & 0 & \cdots & c & \cdots & s & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\
j & 0 & \cdots & -s & \cdots & c & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1
\end{pmatrix}
\]  

(2.40)

with \(c = \cos(\theta)\) and \(s = \sin(\theta)\). The matrix vector product \(U(i,j,\theta)\mathbf{x}\) is called Givens rotation. Geometrically \(U(i,j,\theta)\) is a rotation counterclockwise through \(\theta\) in the plane with the \(i\)-th unit vector as its \(x\)-axis and the \(j\)-th vector as its \(y\)-axis.

We observe that a plane rotation is inexpensive to apply to a vector. If \(\mathbf{x}'U(i,j,\theta) = \mathbf{v}'\), then

\[
v_i = cx_i - sx_j, \\
v_j = cx_j + sx_i, \\
v_k = x_k (k \neq i,j)
\]

(2.41)

A plane rotation may be multiplied into a vector at a cost of two additions and four multiplications.

Another attractive property of givens rotations, frequently applied in the presented algorithms, is that \(U(i,j,\theta)\) can be chosen to introduce a zero into the \(j\)-th component of a vector. If we set

\[
c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad s = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}
\]

(2.42)

in Eq. 2.41 then

\[
v_i = \sqrt{x_i^2 + x_j^2}, \\
v_j = 0
\]

(2.43)

In the following we assume \(A \in \mathbb{R}^{m \times m}\) and \(\mathbf{x} \in \mathbb{R}^m\).

2.7.1 Rank one update

If a positive semi-definite rank-one-matrix is added up to a symmetric positive definite matrix \(A\)

\[
\tilde{A} = A + \mathbf{x}\mathbf{x}'
\]

(2.44)
with Cholesky decomposition
\[ A = LL', \]  
we are interested in a method to update the existing Cholesky decomposition instead of computing it from scratch, for more details see [3]. To update the factorization we shall construct an orthogonal matrix \( U \) such that
\[ [\mathbf{L} \ x] U = [\tilde{\mathbf{L}} \ 0]. \]  
To see this approach will work and since \( U \) is orthogonal
\[ \tilde{\mathbf{L}} \mathbf{L}' = [\tilde{\mathbf{L}} \ 0] \cdot [\tilde{\mathbf{L}}'] = [\mathbf{L} \ x] \cdot [\mathbf{L}' \ x] = \mathbf{L} \mathbf{L}' + xx' = \tilde{\mathbf{A}}. \]  

The matrix \( U \) is constructed via the product of \( U_1 \cdots U_m \), with \( m \) the dimension of \( x \) and \( U_1 \) a rotation in the \((i,m+1)\)-plane (Eq. 2.40). In the following the approach is illustrated for \( m = 4 \). Consider the matrix
\[ [\mathbf{L} \ x] = \begin{bmatrix} l & 0 & 0 & 0 & \circ \\ l & l & 0 & 0 & x \\ l & l & l & 0 & x \\ l & l & l & l & x \end{bmatrix}. \]  
The rotation \( U_1 \) in the \((1,5)\)-plane is determined to introduce a zero (Eq. 2.42) in the circled element above. After this application the matrix has the form
\[ [\tilde{l} & 0 & 0 & 0 & \circ] \begin{bmatrix} \tilde{l} & 0 & 0 & 0 & \circ \\ \tilde{l} & l & 0 & 0 & x \\ \tilde{l} & l & l & 0 & x \\ \tilde{l} & l & l & l & x \end{bmatrix}. \]  
The next rotation \( U_2 \) in the \((2,5)\)-plane is introducing a zero into the \((2,5)\)-element, and so on:
\[ \begin{bmatrix} \tilde{l} & 0 & 0 & 0 & 0 \\ \tilde{l} & l & 0 & 0 & x \\ \tilde{l} & l & l & 0 & x \\ \tilde{l} & l & l & l & x \end{bmatrix} U_2 \begin{bmatrix} \tilde{l} & 0 & 0 & 0 & 0 \\ \tilde{l} & \tilde{l} & 0 & 0 & 0 \\ \tilde{l} & l & l & 0 & 0 \\ \tilde{l} & l & l & l & 0 \end{bmatrix} U_3 \begin{bmatrix} \tilde{l} & 0 & 0 & 0 & 0 \\ \tilde{l} & \tilde{l} & \tilde{l} & 0 & 0 \\ \tilde{l} & \tilde{l} & \tilde{l} & l & 0 \\ \tilde{l} & \tilde{l} & \tilde{l} & l & l \end{bmatrix} U_4 \begin{bmatrix} \tilde{l} & 0 & 0 & 0 & 0 \\ \tilde{l} & \tilde{l} & \tilde{l} & \tilde{l} & 0 \\ \tilde{l} & \tilde{l} & \tilde{l} & \tilde{l} & l \\ \tilde{l} & \tilde{l} & \tilde{l} & \tilde{l} & l \end{bmatrix}. \]  
Multiplying the rotations we finally obtain
\[ [\mathbf{L} \ x] U_1 \cdots U_m = [\tilde{\mathbf{L}} \ 0]. \]  

### 2.7.2 Rank one downdate

If a positive semi-definite rank-one-matrix is subtracted from a symmetric positive definite matrix \( A \)
\[ \tilde{\mathbf{A}} = \mathbf{A} - xx', \]  

\[ (2.52) \]
we are again interested in a method to update the corresponding Cholesky decomposition of $A$, for more details see [3].

To downdate the factorization we construct an orthogonal matrix $U$ similar to the update case, such that

$$[L \ 0] U = [	ilde{L} \ x].$$

(2.53)

To see the approach is going to work, note that

$$A = LL' = [L \ 0] U' \begin{bmatrix} L' \\ 0 \end{bmatrix} = [	ilde{L} \ x] \begin{bmatrix} \tilde{L}' \\ x \end{bmatrix} = \tilde{L}\tilde{L}' + xx'.$$

(2.54)

Hence

$$\tilde{A} = \tilde{L}\tilde{L}' = LL' - xx' = A - xx'.$$

(2.55)

To compute $U$ we need to solve the system

$$La = x.$$

(2.56)

Afterwards it must be established that $\|a\|_2 < 1$ since this is a necessary condition for downdating to be possible.

Assume $\|a\| \geq 1$, we have

$$A - xx' = LL' - xx' = L(I - L^{-1}xx'L^{-1})L' = L(I - aa')L'.$$

(2.57)

Based on the assumptions, the matrix $I - aa'$ has the non-positive eigenvalue $1 - \|a\|^2$ and with the help of Slyvester’s law of inertia [9] one can show that $L(I - aa')L'$ is not positive definite. So $\|a\| < 1$ must hold to assure a downdated Cholesky factorization exists.

Afterwards we set $\beta = \sqrt{1 - \|a\|^2}$ and need to determine rotations $U_1, U_2, \ldots U_m$, with $U_1$ a rotation in the $(m + 1, i)$-plane so that

$$\begin{bmatrix} a' \\ \beta \end{bmatrix} U_m \cdots U_2 U_1 = [0 \ 1].$$

(2.58)

In the following the order in which zeros are introduced (Eq. 2.42) is illustrated for $m = 3$:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \beta \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ a_2 \\ 0 \\ \beta \end{bmatrix} \rightarrow \begin{bmatrix} a_1 \\ 0 \\ 0 \\ \beta \end{bmatrix} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ \beta \end{bmatrix}.$$

(2.59)

The final value of $\beta$ must be 1, since

$$\|\begin{bmatrix} a' \\ \beta \end{bmatrix} U_m \cdots U_2 U_1\|_2 = \|\begin{bmatrix} a' \\ \beta \end{bmatrix}\|_2 = 1$$

(2.60)

resulting from the choice of $\beta$ and the orthogonal $U_i$’s. The structure of the rotations $U_1$ ensures that $\tilde{L}$ is in fact a lower triangular matrix. Specifically, if $l_j$ and $\tilde{l}_j$ denote the $j$-th rows of $L$ and $\tilde{L}$, we obtain, because the last $m - j$ elements of $l_j$ are zero,

$$[\tilde{l}_j \ \tilde{x}_j] = [l_j \ 0] U_m \cdots U_1 = [l_j \ 0] U_j \cdots U_1.$$

(2.61)

16
It remains to show that $x = \tilde{x}$, but this follows directly from Eq. 2.56, Eq. 2.58, Eq. 2.61 and $U = U_m \cdots U_1$ orthogonal

$$x' = [a' \beta] \begin{pmatrix} L' \\ 0 \end{pmatrix} = [a' \beta] UU' \begin{pmatrix} L' \\ 0 \end{pmatrix} = [0 1] \begin{pmatrix} \tilde{L} \\ \tilde{x} \end{pmatrix} = \tilde{x}' . \quad (2.62)$$

### 2.7.3 Permutations

Given a symmetric positive definite matrix $A$ with Cholesky decomposition $A = LL'$, we want to apply symmetric permutations of the rows and columns of $A$

$$\tilde{A} = E'AE , \quad (2.63)$$

with $E$ a permutation matrix (identity matrix with its columns permuted), for more details see [3].

To update the factorization we shall compute an orthogonal transformation $U$ such that

$$E'LU = \tilde{L} . \quad (2.64)$$

To see the approach works, notice that

$$E'AE = E'L \cdot L'E = E'LU \cdot U'L'E = \tilde{L} \tilde{L}' = \tilde{A} . \quad (2.65)$$

We will consider two kinds of permutations. This will be sufficient to perform any arbitrary permutation, since they can be computed via a sequence of the two kinds introduced. A right circular shift among columns $k$ through $l$ leads to a permutation of the rows and columns of the form

$$1, \ldots, k-1, l, k, k+1, \ldots, l-1, l+1, \ldots, m. \quad (2.66)$$

Analogous a left circular shift among columns $k$ through $l$ leads to

$$1, \ldots, k-1, k+1, \ldots, l-1, l, k, l+1, \ldots, m. \quad (2.67)$$

In the following both cases will be illustrated with $m = 6, k = 2, l = 5$. For the right circular shift, the matrix $E'L$ has the form

$$\begin{bmatrix} x & 0 & 0 & 0 & 0 & 0 \\ x & x & x^3 & x^2 & x^1 & 0 \\ x & x & 0^3 & 0 & 0 & 0 \\ x & x & x & 0^2 & 0 & 0 \\ x & x & x & x & 0^1 & 0 \\ x & x & x & x & x & x \end{bmatrix} . \quad (2.68)$$

To reduce $E'L$ to triangular form, the elements $x^i$ are successively set to 0 via rotations $U_i$ in the $(l - i, l - i + 1)$-plane. The rotation $U_i$ introduces a nonzero element on the
diagonal at $0^i$. Therefore, the final matrix $\tilde{L}$ is lower triangular. For the left circular shift, the matrix $E'L$ has the form

$$
\begin{bmatrix}
 x & 0 & 0 & 0 & 0 \\
 x & x^1 & 0 & 0 & 0 \\
 x & x & x^2 & 0 & 0 \\
 x & x & x & x^3 & 0 \\
 x & 0 & 0 & 0 & 0 \\
 x & x & x & x & x
\end{bmatrix}
$$

(2.69)

To reduce $E'L$ to triangular form, the elements $x^i$ successively set to 0 via rotations $U_i$ in the $(k+i-1, k+i)$-plane. The final matrix $\tilde{L}$ is lower triangular.

To obtain the final orthogonal transformation matrix $U$, we need to multiply the corresponding $U_i$’s

$$
U = U_1 U_2 \cdots U_p
$$

(2.70)

with $p = l - k$.

### 2.7.4 Add a row/column

Given a symmetric positive definite matrix $A$ with Cholesky decomposition $A = LL'$, $a \in \mathbb{R}^m$ and $\beta \in \mathbb{R}$. Suppose $A$ is extended as follows:

$$
\tilde{A} = \begin{pmatrix} A & a \\ a' & \beta \end{pmatrix}
$$

(2.71)

We are interested in an algorithm to expand the Cholesky factorization of $A$ without computing it from scratch, for further information see [12]. In the following it is shown that under certain conditions the corresponding Cholesky factor has the form

$$
\tilde{L} = \begin{pmatrix} L & 0 \\ c' & d \end{pmatrix}
$$

(2.72)

with $L \in \mathbb{R}^{m \times m}$ the Cholesky factor of $A$, $c \in \mathbb{R}^m$ and $d \in \mathbb{R}_+$.

By Theorem 2.1

$$
\tilde{A} = \begin{pmatrix} A & a \\ a' & \beta \end{pmatrix} = \begin{pmatrix} L & 0 \\ c' & d \end{pmatrix} \begin{pmatrix} L' & c \\ 0 & d \end{pmatrix} = \begin{pmatrix} LL' & Lc \\ c'L' & c'c + d^2 \end{pmatrix}
$$

(2.73)

should hold. $d$ and $c$ need to be chosen that we obtain

$$
a = Lc
$$

(2.74)

$$
\beta = c'c + d^2.
$$

(2.75)

Since $L$ is invertible

$$
c = L^{-1}a \quad \text{(forward substitution)}.
$$

(2.76)
Plugging Eq. 2.76 into Eq. 2.75 and resolving into $d$ leads to

$$d = \sqrt{\beta - \mathbf{a}' \mathbf{A}^{-1} \mathbf{a}}. \quad \text{(2.77)}$$

So we need to assure $\mathbf{a}' \mathbf{A}^{-1} \mathbf{a} < \beta$, otherwise $d$ is not well-defined. With the help of Sylvester’s law of inertia [9] one can show, that if $\mathbf{a}' \mathbf{A}^{-1} \mathbf{a} \geq \beta$, $\mathbf{A}$ would not be positive definite anymore and thus the Cholesky decomposition would not exist.

### 2.7.5 Remove first row/column

Given a symmetric positive definite matrix $\mathbf{A}$ with Cholesky decomposition $\mathbf{A} = \mathbf{L}\mathbf{L}'$, we want to receive a reduced Cholesky factorization of the matrix $\tilde{\mathbf{A}}$, whereas

$$\mathbf{A} = \begin{pmatrix} \beta & \mathbf{a}' \\ \mathbf{a} & \tilde{\mathbf{A}} \end{pmatrix}, \quad \text{(2.78)}$$

with $\beta \in \mathbb{R}$ and $\mathbf{a} \in \mathbb{R}^{m-1}$.

Partitioning of $\mathbf{L}$ yields

$$\mathbf{A} = \mathbf{L}\mathbf{L}' = \begin{pmatrix} \beta & \mathbf{a}' \\ \mathbf{a} & \tilde{\mathbf{A}} \end{pmatrix} = \begin{pmatrix} d & 0 \\ c & \tilde{\mathbf{L}} \end{pmatrix} \begin{pmatrix} d & \mathbf{c}' \\ 0 & \tilde{\mathbf{L}}' \end{pmatrix} = \begin{pmatrix} d^2 & \mathbf{d}c' \\ cd & \mathbf{c}c' + \tilde{\mathbf{L}}\tilde{\mathbf{L}}' \end{pmatrix}, \quad \text{(2.79)}$$

with $\tilde{\mathbf{L}} \in \mathbb{R}^{(m-1)\times(m-1)}$, $\mathbf{c} \in \mathbb{R}^{m-1}$ and $d \in \mathbb{R}$. So we obtain

$$\beta = d^2 \quad \text{(2.80)}$$
$$\mathbf{a} = cd \quad \text{(2.81)}$$
$$\tilde{\mathbf{A}} = \mathbf{c}c' + \tilde{\mathbf{L}}\tilde{\mathbf{L}}'. \quad \text{(2.82)}$$

Observing, that we achieved a Cholesky factorization of

$$\tilde{\mathbf{A}} - \mathbf{c}c' = \tilde{\mathbf{L}}\tilde{\mathbf{L}}'. \quad \text{(2.83)}$$

To receive the desired decomposition of $\tilde{\mathbf{A}}$, we need to perform a rank one update based on the update vector $\mathbf{c}$ (Sec. 2.7.1):

$$\tilde{\mathbf{A}} = (\tilde{\mathbf{A}} - \mathbf{c}c') + \mathbf{c}c' = \tilde{\mathbf{A}} = \tilde{\mathbf{L}}\tilde{\mathbf{L}}'. \quad \text{(2.84)}$$
3 Algorithms and methods

In the following section we apply the Cholesky decomposition (Sec. 2.6) and proposed Cholesky modifications (Sec. 2.7) to the sparse grid based density estimation approach (Sec. 2.4), in order to achieve an adaptive factorization of the system matrix $R + \lambda I$ (Eq. 2.29) regarding an Offline/Online splitting (Sec. 2.5). We assume the system matrix and $b$ (Eq. 2.29) are available, for details see [2]. Also we denote the grid size by $N$, so we obtain $R \in \mathbb{R}^{N \times N}$, $I \in \mathbb{R}^{N \times N}$ and $\alpha, b \in \mathbb{R}^N$.

3.1 Factorization

Since we have shown that the matrix $R + \lambda I$ is symmetric and positive definite, the Cholesky decomposition of order $O(N^3)$ can be applied (Sec. 2.6)

$$R + \lambda I = LL'.\quad (3.1)$$

The corresponding lower triangular matrix $L$ is stored in the Offline phase. To construct a density in the Online phase, the system of linear equations $(R + \lambda I)\alpha = b$ to obtain the coefficients $\alpha$ needs to be solved. In order to do this $(R + \lambda I)^{-1}b$ has to be calculated. Utilizing the stored $L$, this can be efficiently achieved via forward and back substitution

$$\alpha = (R + \lambda I)^{-1}b = (LL')^{-1}b = L^{-1}L^{-1}b\quad (3.2)$$
$$c = L^{-1}b \quad \text{(forward substitution)}\quad (3.3)$$
$$\alpha = L^{-1}c \quad \text{(back substitution)}\quad .\quad (3.4)$$

Since forward and back substitution are in $O(N^2)$ the Online phase is in $O(N^2)$ as well, details can be found in [6].

3.2 Refinement

In the following an algorithm is presented to reshape the Cholesky factor (Eq. 3.1) if the grid size is increased, i.e. if refinement took place, without computing it from scratch. This enables us to avoid a full new Offline step.

We start with the simplest case that one additional grid point is integrated. So the resulting grid size is denoted by $N + 1$. The expanded system matrix looks as follows

$$\tilde{R} + \tilde{\lambda}I = \begin{pmatrix} R & a \\ \alpha & \beta \end{pmatrix} + \begin{pmatrix} \lambda I & 0 \\ 0 & \lambda \end{pmatrix} = \begin{pmatrix} R + \lambda I & a \\ \alpha & \beta + \lambda \end{pmatrix} \in \mathbb{R}^{(N+1) \times (N+1)}\quad (3.5)$$
3 Algorithms and methods

with

\[ a_i = (\varphi_i, \varphi_{N+1})_{L_2}, \ i \in \{1, \ldots, N\} \]  
and \( \varphi_{N+1} \) (Eq. 2.7) as the hierarchical basis function corresponding to the new grid point. Since the expanded system matrix (Eq. 3.5) is symmetric and positive definite (Sec. 2.6), we can apply Sec. 2.7.4.

**Algorithm 1** Add one grid point

1: \( \mathbf{L} \leftarrow \text{Cholesky factor (Eq. 3.1)} \)
2: \( \mathbf{a} \leftarrow \text{update vector (Eq. 3.6)} \)
3: \( \beta \leftarrow \text{new diagonal element (Eq. 3.6)} \)
4: \( N \leftarrow \text{grid size} \)
5: \( \lambda \leftarrow \text{regularization parameter} \)

6: \text{function}\ ADDONEGRIDPOINT(\( \mathbf{L}, \mathbf{a}, \beta, N, \lambda \))
7: \( c \leftarrow \mathbf{L}^{-1} \mathbf{a} \)
8: \( d \leftarrow \sqrt{\beta + \lambda - c^c} \)
9: \text{for } i \leftarrow 1 \text{ to } N \text{ do}
10: \quad \mathbf{L}(N+1;i) \leftarrow c_i
11: \quad \mathbf{L}(i,N+1) \leftarrow 0
12: \text{end for}
13: \mathbf{L}(N+1,N+1) \leftarrow d
14: \text{end function}

This approach can easily be extended to add \( n \) grid points:

**Algorithm 2** Add \( n \) grid points

1: \( \mathbf{L} \leftarrow \text{Cholesky factor (3.1)} \)
2: \( N \leftarrow \text{grid size} \)
3: \( n \leftarrow \text{number of new points} \)
4: \( \lambda \leftarrow \text{regularization parameter} \)

5: \text{function}\ REFINEMENT(\( \mathbf{L}, N, n \))
6: \text{for } i \leftarrow 1 \text{ to } n \text{ do}
7: \quad \text{for } j \leftarrow 1 \text{ to } N + i - 1 \text{ do}
8: \quad \quad \quad a_j \leftarrow (\varphi_j, \varphi_{N+i})_{L_2} \quad \triangleright \varphi_{N+i} \text{ hierarchical basis functions of added points}
9: \quad \text{end for}
10: \quad \beta \leftarrow (\varphi_{N+i+1}, \varphi_{N+i})_{L_2}
11: \quad \text{ADDONEGRIDPOINT(}\( \mathbf{L}, \mathbf{a}, \beta, N + i - 1, \lambda \))
12: \text{end for}
13: \text{end function}

Afterwards the new Cholesky factor is located in \( \mathbf{L} \in \mathbb{R}^{(N+n) \times (N+n)} \). The order of Alg. 1 is in \( \mathcal{O}(N^2) \) and needs to be called \( n \)-times, if \( n \) grid points are added to the grid (Alg. 2). For corresponding runtimes and comments see Sec. 5.3.2.
3 Algorithms and methods

3.3 Coarsening

In case the grid size is decreased, e.g. if superfluous grid points are removed, we are interested in an algorithm to downsize the Cholesky factor (Eq. 3.1) without applying a new Cholesky decomposition, to prevent a full new Offline step.

Again we start with the simplest case of removing one grid point. The new grid size is denoted by $N - 1$. We assume the index $i$ of the removed grid point is in $\{1, \ldots, N\}$. The resulting matrix can be posed as follows,

$$\tilde{R} + \lambda \tilde{I} = \left( \begin{array}{cc} R_{(1; i-1; 1:i-1)} & R_{(1:i-1; i+1:N)} \\ R_{(i+1:N; 1:i-1)} & R_{(i+1:N; i+1:N)} \end{array} \right) + \lambda \tilde{I} \in \mathbb{R}^{(N-1)\times(N-1)} \quad (3.7)$$

corresponding to delete the $i$-th row and column of the old system matrix. Since the modified system matrix is symmetric and positive definite (Sec. 2.6), the existence of a Cholesky decomposition is ensured.

In case

$$i \in \{c, \ldots, N\}, \quad (3.8)$$

we start with permuting the corresponding row and column of the removed grid point outwards, such that

$$A_{\text{perm}2^i} := \left( \begin{array}{cc} R_{(1;i-1; 1:i-1)} & R_{(1:i-1; i+1:N)} \\ R_{(i+1:N; 1:i-1)} & R_{(i+1:N; i+1:N)} \end{array} \right) + \lambda I. \quad (3.9)$$

By applying Sec. 2.7.3 with the appropriate left circular shift (Eq. 3.9) denoted by $k = i$ and $l = N$, we receive a Cholesky factorization of the permuted system matrix

$$A_{\text{perm}2^i} = L_{\text{perm}2^i} R_{\text{perm}2^i}. \quad (3.10)$$

Observing

$$A_{\text{perm}2^i} = L_{\text{perm}2^i} R_{\text{perm}2^i} = \begin{pmatrix} \tilde{L} & 0 \\ c' & d \end{pmatrix} \begin{pmatrix} \tilde{L}' & c \\ 0 & d \end{pmatrix} = \begin{pmatrix} \tilde{R} + \lambda \tilde{I} & a \\ a' & \beta \end{pmatrix}$$

$$= \begin{pmatrix} \tilde{L} \tilde{L}' & \tilde{L}c \\ c' \tilde{L}' & c'c + d^2 \end{pmatrix} \quad (3.11)$$

with

$$a = \begin{pmatrix} R_{(1:i-1; i)} \\ R_{(i+1:N; i)} \end{pmatrix}, \quad \beta = R_{(i; i)} + \lambda \quad (3.12)$$

and $(c' \ d)$ denoting the $N$-th row of $L_{\text{perm}2^i}$, we obtain a Cholesky decomposition of the downsized system matrix (Eq. 3.7) just by dropping the last row and column of $L_{\text{perm}2^i}$, since

$$\tilde{R} + \lambda \tilde{I} = \tilde{L} \tilde{L}'. \quad (3.13)$$
If
\[ i \in \{1, \ldots, c\}, \] (3.14)
we permute the corresponding row and column of the removed grid point towards the first row/column, such that
\[
A_{\text{perm}1} := \begin{pmatrix}
R_{(i;i)} & R_{(i;i-1;1:i-1)} & R_{(i;i-1;1:i;N)} \\
R_{(1:i-1;i)} & R_{(1;i-1;1:i-1)} & R_{(1;i-1;1:i;N)} \\
R_{(i+1:N;i)} & R_{(i+1:N;1;i-1)} & R_{(i+1:N;1:i;N)} \\
R_{(1:i-1;1)} & R_{(1:i-1;1;i-1)} & R_{(1:i-1;1;i;N)} \\
R_{(i+1:N;i)} & R_{(i+1:N;1;i-1)} & R_{(i+1:N;1;i;N)}
\end{pmatrix} + \lambda I. \] (3.15)

By applying Sec. 2.7.3 with the appropriate right circular shift (Eq. 3.15) denoted by \( k = 1 \) and \( l = i \), we receive a Cholesky factorization of the permuted system matrix
\[
A_{\text{perm}1} = L_{\text{perm}1} L_{\text{perm}1}'. \] (3.16)

To obtain the Cholesky decomposition of the downsized system matrix (Eq. 3.7), we employ Sec. 2.7.5 with
\[
A_{\text{perm}1} = L_{\text{perm}1} L_{\text{perm}1}' = \begin{pmatrix}
d & 0 \\
c & \tilde{L}
\end{pmatrix} \begin{pmatrix}
d & c' \\
0 & \tilde{L}'
\end{pmatrix} = \begin{pmatrix}
\beta & a' \\
a & \tilde{R} + \lambda \tilde{I}
\end{pmatrix}. \] (3.17)

To determine \( c \) in Eq. 3.8 and Eq. 3.14, we need to take a closer look at the amount of floating point operations in Alg. 3 and Alg. 4. \( c \) can be seen as the break-even point according that the cost of a right circular shift and the corresponding rank one update is less (case Eq. 3.14) than the cost of the possible left circular shift followed by removing the last row/column (Eq. 3.10, 3.13).

Alg. 4 requires about
\[ 3 \cdot N^2 \] floating-point operations. (3.18)

Whereas the operation counts regarding the givens rotations (Definition 2.4) in Alg. 3 depend on \( k \) and \( l \), but they do not depend on the applied kind of shift, for further details see [3]:
\[ 6 \cdot \left( N - \frac{1}{2} (l + k) \right) (l - k) \] floating-point operations are required. (3.19)

As one can see, if \( l \) is near \( N \) and \( k \) is near 1 the operation count of Alg. 3 approaches the one of Alg. 4. But, we need to swap the rows before applying the transformations to \( L \). Those swaps take around \( \frac{1}{3} \) of the time to perform a Cholesky permutation (Eq. 3.19) for larger grid settings (Figure 8.1). Therefore, Alg. 4 is faster for \( l \) chosen around \( N \) and \( k \) around \( \frac{1}{10}N \). In case of Eq. 3.14 we need to perform right circular shifts with \( k \) around 1 and \( l \) smaller \( c \) and memory allocations to get the resized Cholesky factor \( L \) (Alg. 5),
\[ c \in \{ \frac{1}{20}N, \ldots, \frac{1}{10}N \} \] would be an appropriate a priori choice (Sec. 5.3). (3.20)

This approach can be extended to remove \( n \) grid points, if both cases (Eq. 3.8, 3.14) are combined:
Algorithm 3 Permutation of Cholesky factor

1: \( \mathbf{L} \leftarrow \) Cholesky factor (Eq. 3.1)
2: \( k \leftarrow \) column/row to permute \( \triangleright \) corresponding to removed grid point (Eq. 3.8)
3: \( l \leftarrow \) destination index
4: \( \text{job} \leftarrow (1) \) corresponds to case Eq. 3.14, (2) corresponds to case Eq. 3.8
5:
6: \textbf{function PERMUTEONEGRIDPOINT}(\( \mathbf{L}, k, l, \text{job} \))
7: \quad \textbf{if} \ \text{job} \ \text{equals} \ 2 \ \textbf{then}
8: \quad \quad \text{Perform left circular shift (Eq. 3.10)}
9: \quad \quad \textbf{for} \ i \quad \textbf{from} \ k \ \text{to} \ l - 1 \ \textbf{do}
10: \quad \quad \quad \text{SWAPROWS}(\mathbf{L}, i, i + 1) \ \triangleright \text{Permute} \ k\text{-th column/row of} \ \mathbf{L} \ \text{outwards}
11: \quad \quad \textbf{end for}
12: \quad \quad \text{Transform} \ \mathbf{L} \ \text{back to lower triangular}
13: \quad \quad \textbf{for} \ j \quad \textbf{from} \ 1 \ \text{to} \ l - k \ \textbf{do}
14: \quad \quad \quad \text{GIVENSROTATION}(\mathbf{L}, k, j) \ \triangleright \text{For details see Sec. 2.7.3.}
15: \quad \quad \textbf{end for}
16: \quad \textbf{else if} \ \text{job} \ \text{equals} \ 1 \ \textbf{then}
17: \quad \quad \text{Perform right circular shift (Eq. 3.16)}
18: \quad \quad \textbf{for} \ i \quad \textbf{from} \ l \ \text{to} \ k - 1 \ \textbf{do}
19: \quad \quad \quad \text{SWAPROWS}(\mathbf{L}, i, i - 1)
20: \quad \quad \textbf{end for}
21: \quad \quad \text{Transform} \ \mathbf{L} \ \text{back to lower triangular}
22: \quad \quad \textbf{for} \ j \quad \textbf{from} \ 1 \ \text{to} \ l - k \ \textbf{do}
23: \quad \quad \quad \text{GIVENSROTATION}(\mathbf{L}, k, j) \ \triangleright \text{For details see Sec. 2.7.3.}
24: \quad \quad \textbf{end for}
25: \quad \textbf{end if}
26: \textbf{end function}

Algorithm 4 Rank one update

1: \( \mathbf{L} \leftarrow \) Cholesky factor (Eq. 3.1)
2: \( \mathbf{x} \leftarrow \) vector representing rank one matrix \( \mathbf{xx}' \)
3: \( N \leftarrow \) grid size
4: 5: \textbf{function RANKONEUPDATE}(\( \mathbf{L}, \mathbf{x}, N \))
6: \quad \textbf{for} \ i \quad \textbf{from} \ 1 \ \text{to} \ N \ \textbf{do}
7: \quad \quad \text{GIVENSROTATION}(\mathbf{L}, \mathbf{x}, i) \ \triangleright \text{Determine and apply rotation} \ \mathbf{U}_i \ \text{(Sec. 2.7.1)}
8: \quad \textbf{end for}
9: \textbf{end function}
Algorithm 5 Remove n grid points

1: $\mathbf{L} \leftarrow$ Cholesky factor (3.1)
2: $N \leftarrow$ grid size
3: $V \leftarrow$ list containing the indices of $n$ removed points
4:
5: function Coarsening($\mathbf{L}$, $N$, $V$)
6: $Job1 \leftarrow 0$  \hspace{1em} $\triangleright$ Counts how often case Eq. 3.14 is tackled
7: $Job2 \leftarrow 0$  \hspace{1em} $\triangleright$ Counts how often case Eq. 3.8 is tackled
8: for $i$ in $V$ do
9: index $\leftarrow i - Job2$
10: if $i > c$ then \hspace{1em} $\triangleright$ Eq. 3.20
11: PermuteOneGridPoint($\mathbf{L}$, $i$, $N$, 2)
12: $Job2 \leftarrow Job2 + 1$
13: else
14: PermuteOneGridPoint($\mathbf{L}$, 1, $i$, 1)
15: $Job1 \leftarrow Job1 + 1$
16: end if
17: end for
18: Delete last $Job2$ many rows/columns permuted corresponding to case Eq. 3.8
19: ResizeCholeskyFactor($\mathbf{L}$, $N - Job2$)
20: Calculate $Job1$ many rank one updates (Sec. 2.7.5) (Eq. 3.17) to obtain
21: the final Cholesky factor $\tilde{\mathbf{L}}$
22: $\tilde{\mathbf{L}} \leftarrow$ Submatrix $\mathbf{L}(Job1+1,:)\cdot(Job1+1,:)$ \hspace{1em} $\triangleright$ See Eq. 3.17
23: for $i \leftarrow 1$ to $Job1$ do
24: updateVector $\leftarrow \mathbf{L}(Job1+1,:),i$ \hspace{1em} $\triangleright$ See Alg. 4
25: RankOneUpdate($\tilde{\mathbf{L}}$, updateVector, $N - Job1 - Job2$)
26: end for
27: end function

Afterwards the new Cholesky factor is located in $\tilde{\mathbf{L}} \in \mathbb{R}^{(N-n)\times(N-n)}$. The order of Alg. 5 is in $O(N^2)$ since $n$ permutations, which are in $O(N^2)$ (Alg. 3), and rank one updates, based on $c$ of order $O((N-n)^2)$ (Alg. 4), need to be applied if $n$ grid points are removed. For corresponding runtimes see Sec. 5.3.1.
3 Algorithms and methods

3.4 Modify regularization parameter

Until now we need to compute a new Cholesky decomposition of \( R + \lambda I \) every time the regularization parameter \( \lambda \in \mathbb{R}_+ \) is changed. This leads to high degree of computational effort if cross-validation is applied, to determine the optimal lambda regarding an accuracy measure.

We develop an algorithm to modify the existing Cholesky factor \( L \) (Sec. 3.1), that applies \( N \)-rank one updates (Sec. 2.7.1) if \( \lambda \) is increased and corresponding \( N \)-rank one downdates (Sec. 2.7.2) if \( \lambda \) is decreased. In the following \( \lambda \) denotes the old regularization parameter and \( \tilde{\lambda} \) the new one.

3.4.1 Increase \( \lambda \)

At first the increasing \( \lambda \) case is addressed: \( \lambda < \tilde{\lambda} \)

\[
R + \tilde{\lambda} I = R + \lambda I + (\tilde{\lambda} - \lambda)I = LL^T + (\tilde{\lambda} - \lambda)I.
\] (3.21)

Whereas \((\tilde{\lambda} - \lambda)I\) can be expressed as a sum of rank one matrices

\[
(\tilde{\lambda} - \lambda)I = \sum_{i=1}^{N} x_i x_i^T
\] (3.22)

with

\[
x_{ij} = \begin{cases} \sqrt{\tilde{\lambda} - \lambda}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}, \quad 1 \leq j \leq N.
\] (3.23)

Since \( \tilde{\lambda} \in \mathbb{R}_+ \), the modified system matrix (Eq. 3.21) is symmetric and positive definite (Sec. 2.6). Therefore we can apply \( N \)-rank one updates (Sec. 2.7.1) of the form

\[
A_1 = R + \lambda I + x_1 x_1^T,
A_i = A_{i-1} + x_i x_i^T, \quad 1 \leq i < N,
A_N = R + \tilde{\lambda} I
\] (3.24)

to receive an updated Cholesky factor. In the following the implementation is sketched:
### 3 Algorithms and methods

**Algorithm 6** Increase regularization parameter $\lambda$

1: $L \leftarrow$ Cholesky factor (Eq. 3.1)
2: $N \leftarrow$ grid size
3: $\lambda \leftarrow$ current regularization parameter
4: $\tilde{\lambda} \leftarrow$ new regularization parameter
5: 
6: **function** INCREASELAMBDA($L$, $N$, $\lambda$, $\tilde{\lambda}$)
7:     **for** $i \leftarrow 1$ to $N$ **do**
8:         **for** $j \leftarrow 1$ to $N$ **do**
9:             **if** $i$ equals $j$ **then**
10:                 $x_j \leftarrow \sqrt{\lambda - \tilde{\lambda}}$
11:             **else**
12:                 $x_j \leftarrow 0$
13:             **end if**
14:             RANKONEUPDATE($L$, $x$, $N$) \hspace{1cm} ▷ Alg. 4
15:     **end for**
16: **end function**

The order of Alg. 4 is $O(N^2)$ and since $N$ such updates need to be applied to increase $\lambda$, Alg. 6 is in $O(N^3)$, details can be found in [3]. Because Alg. 6 is of the same order as the Offline step (Sec. 3.1), we worked out that Alg. 6 is slower than computing a new Cholesky decomposition (Sec. 5.4). But due to possible parallelization we proposed this method (Sec. 5.4).

#### 3.4.2 Decrease $\lambda$

Now the decreasing $\lambda$ case is tackled: $\lambda > \tilde{\lambda}$

$$R + \tilde{\lambda}I = R + \lambda I + (\tilde{\lambda} - \lambda)I = R + \lambda I - (\lambda - \tilde{\lambda})I = LL' - (\lambda - \tilde{\lambda})I.$$  \hspace{1cm} (3.25)

Whereas $(\lambda - \tilde{\lambda})I$ can again be expressed as a sum of rank one matrices

$$(\lambda - \tilde{\lambda})I = \sum_{i=1}^{N} x_i x_i'$$  \hspace{1cm} (3.26)

with

$$x_{ij} = \begin{cases} \sqrt{\lambda - \tilde{\lambda}}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases}, \quad 1 \leq j \leq N.$$  \hspace{1cm} (3.27)

Since $\tilde{\lambda} \in \mathbb{R}_+$, the modified system matrix (Sec. 3.25) is symmetric and positive definite (Sec. 2.6). Therefore we can apply $N$-rank one downdates (Sec. 2.7.2) of the form

$$A_1 = R + \lambda I - x_1x_1',$$

$$A_i = A_{i-1} - x_ix_i',$$

$$A_N = R + \tilde{\lambda}I$$  \hspace{1cm} (3.28)
3 Algorithms and methods

to receive an updated Cholesky factor. Alg. 7 and Alg. 8 sketch the implementation:

Algorithm 7 Rank one downdate
1: \( L \leftarrow \) Cholesky factor (3.1) 
2: \( x \leftarrow \) vector representing rank one matrix \( xx' \) 
3: \( N \leftarrow \) grid size 
4: 
5: \textbf{function} RANKONEDOWNDATE\( (L, x, N) \) 
6: \( a \leftarrow L^{-1}x \) 
7: \textbf{for} \( i \leftarrow N \) to 1 \textbf{do} 
8: \hspace{1em} GIVENSROTATION\( (L, a, i) \) \( \triangleright \) Determine and apply rotation \( U_i \) (Sec. 2.7.2) 
9: \textbf{end for} 
10: \textbf{end function} 

Algorithm 8 Decrease regularization parameter \( \lambda \)
1: \( L \leftarrow \) Cholesky factor (3.1) 
2: \( N \leftarrow \) grid size 
3: \( \lambda \leftarrow \) current regularization parameter 
4: \( \tilde{\lambda} \leftarrow \) new regularization parameter 
5: 
6: \textbf{function} DECREASELAMBDA\( (L, N, \lambda, \tilde{\lambda}) \) 
7: \textbf{for} \( i \leftarrow 1 \) to \( N \) \textbf{do} 
8: \hspace{1em} \textbf{for} \( j \leftarrow 1 \) to \( N \) \textbf{do} 
9: \hspace{2em} \textbf{if} \( i \) equals \( j \) \textbf{then} 
10: \hspace{3em} \( x_j \leftarrow \sqrt{\lambda - \tilde{\lambda}} \) 
11: \hspace{2em} \textbf{else} 
12: \hspace{3em} \( x_j \leftarrow 0 \) 
13: \hspace{2em} \textbf{end if} 
14: \hspace{1em} RANKONEDOWNDATE\( (L, x, N) \) 
15: \textbf{end for} 
16: \textbf{end for} 
17: \textbf{end function} 

Alg. 7 is in \( O(N^2) \) and since \( N \) such downdates need to be applied to decrease \( \lambda \), Alg. 8 is in \( O(N^3) \), details can be found in [3]. Because Alg. 8 is of the same order as the Offline step (Sec. 3.1) we worked out that Alg. 8 is slower than computing a new Cholesky decomposition (Sec. 5.4). But due to possible parallelization we proposed this method (Sec. 5.4).
3.5 Adaptive data-streaming

Often when we are in a big data setting, it is not feasible to process all given data at once or in Online Data Mining, data is not accessible in total all the time. In these cases Online/Offline splitting pays off. In the first Offline step an appropriate sparse grid is chosen and the corresponding system matrix will be decomposed (Sec. 3.1). The underlying data $S = \{x_1, \ldots, x_M\} \subset \mathbb{R}^d$ is partitioned into subsets of the form $S_1, S_2, \ldots S_g$ of size $m_1 \ldots m_g$ with $\sum_{i=1}^g m_i = m$ (3.29) to tackle the problems introduced before.

The first Online phase is determined to solve the system

$$(R_1 + \lambda I_1)\alpha_1 = L_1L_1'\alpha_1 = b_1$$

with

$$b_{1i} = \frac{1}{m_i} \sum_{j=1}^{m_1} \varphi_i(x_j), \quad 1 \leq i \leq N \text{ and } x_j \in S_1.$$  (3.31)

Based on the received $\alpha_1$ or e.g. local error estimation, refinement and coarsening (Sec. 2.3) of the underlying sparse grid will be performed. But before the next data batch $S_2$ can be processed, the Cholesky factor $L_1$ of the current system matrix needs to be adapted corresponding to the added and removed grid points. This can be achieved via the introduced algorithms in Sec. 3.2 and Sec. 3.3. Due to runtime savings, Alg. 5 is applied before Alg. 2 is used.

However, the right-hand-side $b_1$ must also be modified to obtain $b_2$. Assume $n_c$ points have been coarsened and $n_r$ points have been refined.

At first the size of $b_1$ is adjusted, given the indices $\mathcal{K}$ of the removed points this leads to

$$\tilde{b}_{2i} = b_{1i}, \quad \forall i \in \{1, \ldots, N\} \setminus \mathcal{K}$$

and set

$$\tilde{b}_{2i} = 0, \quad \forall i \in \{N - n_c + 1, \ldots, N - n_c + n_r\}.$$  (3.33)

to account for the refined points.

Now we need to embed the the data batch $S_2$ into $\tilde{b}_2$

$$b_{2i} = \frac{1}{m_1 + m_2} \left( \sum_{j=1}^{m_2} \varphi_i(x_j) + m_1 \cdot \tilde{b}_{2i} \right), \quad \forall i \in \{1, \ldots, N\} \setminus \mathcal{K} \text{ and } x_j \in S_2$$

$$b_{2i} = \frac{1}{m_2} \sum_{j=1}^{m_2} \varphi_i(x_j), \quad \forall i \in \{N - n_c + 1, \ldots, N - n_c + n_r\}.$$  (3.34)

Afterwards we are able to perform a second Online step (Sec. 3.1) to obtain $\alpha_2$

$$(R_2 + \lambda I_2)\alpha_2 = L_2L_2'\alpha_2 = b_2.$$  (3.35)

This approach can be continued until all data is processed or a stop command is given, for further details see [11].
4 Implementation

Based on SG++ [1] the introduced algorithms have been implemented into libtool and clustc. libtool and clustc are toolboxes to tackle data mining and machine learning tasks mainly depending on data structures and methods provided by SG++ [1, 13, 14, 15, 16].

Italic functions/classes are already implemented and bold ones represent newly implemented methods/classes or modified methods.

libtool contains the core of the algorithms of Sec. 3, whereas clustc implements applications of the proposed algorithms like adaptive density estimation (Sec. 3.5) or classification (Sec. 6.1). Minor changes took place in SG++ primary to boost the performance via elaborate memory allocation. The algorithms of Sec. 3 were implemented with the help of [1, 4].

4.1 libtool

- **DBMatOffline**:
  Based on the presented Offline/Online splitting (Sec. 2.5) the class DBMatOffline represents the Offline part. The function decomposeMatrix has been extended by the possibility to factorize the system matrix due to the Cholesky decomposition (Eq. 3.1). To avoid a new decomposition call if the underlying grid is changed, the function updateCholeskyDecomposition was implemented to take account for refinement and coarsening (Sec. 2.3) (Alg. 5, Alg. 2). Passing a list of indices of removed grid points and the amount of added points, choleskyPermutation (Alg. 3) is called to enable coarsening and corresponding choleskyAddPoint (Alg. 1) is used to allow refinement.

- **DBMatOnlineDE**:
  This class represents the Online phase in the density estimation process (Eq. 3.2). The function computeDensityFunction takes now additional parameters to identify removed and added grid points in order to modify the right hand side of Eq. 2.29 (Sec. 3.5). Additionally procedure calls of class DBMatDMSChol to solve for the density declaring coefficients \( \alpha \) (Eq. 3.3, Eq. 3.4) and to perform cross-validation (Sec. 3.4), based on the Cholesky decomposition explicitly given via a DBMatOffline object, have been added.

- **DBMatDMSChol**:
  This class enables to solve a system of linear equations based on a given Cholesky
factor. The function solve covers forward (Eq. 3.3) and back (Eq. 3.4) substitution to do so. Additionally solve takes two more parameters, one representing the current regularization parameter $\lambda$ and the other one representing the new one $\tilde{\lambda}$ (sec. 3.4). In case $\lambda < \tilde{\lambda}$ (sec. 3.4.1) Alg. 6 is implemented into solve using the function choleskyUpdate (Alg. 4). Otherwise if $\lambda < \tilde{\lambda}$ (Sec. 3.4.2) Alg. 8 is covered utilizing choleskyDowndate.

- **DensityStreamSG:**
  This class represents a density estimation achieved via adaptive data-streaming (Sec. 3.5). The function updateGrid was implemented to enable coarsening and refinement in every streaming step, based on the functions proposed in DBMatOffline.

- **ClassStream:**
  Represents a data-stream classifier based on DensityStreamSG objects. The train function was extended via additional parameters and function calls of covered DBMatOffline and DBMatOnlineDE objects, to enable coarsening and refinement in every streaming step (Sec. 3.5).

4.2 clustc

- **destream:**
  This application to compute a density function via data-streaming is based on a DensityStreamSG object. Using the Cholesky decomposition (Sec. 3.1), a template has been provided to tackle refinement and coarsening tasks in order to enable a fast and adaptive density estimation (Sec. 3.5).

- **classifstream:**
  This application builds a classifier via data-streaming and is based on a ClassStream object. Similar to destream a template has been provided to cope refinement and coarsening tasks if the Cholesky decomposition is applied. Within each streaming step the grids of the different classes get independently updated, to build a fast and adaptive classifier.

4.3 SG++

- **DataMatrix:**
  To prevent the usage of recurring transpose calls in updateCholeskyDecomposition and choleskyAddPoint, the functions resizeQuadratic, appendCol and resizeToSubMatrix have been implemented.
5 Evaluation of algorithms

In the following section we are going to stress the introduced algorithms (Sec. 3) for different sparse grid settings, e.g. dimension and level (Sec. 2.2.), and several volumes of grid refinement/coarsening, to derive recommendations for use. All algorithms are tested with respect to the adaptive density estimation approach presented in Sec. 3.5. We will also sketch the determination process of the parameter $c$ (Eq. 3.20, Sec. 3.3).

All tests have been performed on an Oracle VM Linux Mint operating system with the following hardware:

- 3 processors of Intel(R) Core(TM) i7-4700HQ CPU @ 2.40 GHz
- 4877 MB memory

As compiler, GCC version 4.8.4 was used.

5.1 Offline step

We will start with a runtime comparison of the different matrix factorization methods in the first Offline step, after the system matrix $R + \lambda I$ (Eq. 2.29) is initialized. As proposed in [2] we apply the Eigendecomposition, LU decomposition and, as introduced in Sec. 2.6, the Cholesky decomposition based on an a priori sparse grid choice.

All decomposition methods are in $O(N^3)$. The different settings can be found in Table 8.1. Up to a grid size of around 600 grid points the corresponding runtimes are less than 1 second and the differences are negligible. But as one can see in Figure 5.1, once the grid size exceeds 1000 grid points, the Eigendecomposition runtime is clearly growing more rapidly than the according runtimes of the LU and Cholesky decomposition. So if the underlying problem requires adaptivity due to grid refinement/coarsening, the Eigendecomposition is inapplicable. Based on the runtimes the Cholesky decomposition should be the method of choice. But, even the Cholesky decomposition would cause problems in adaptive and higher dimensional settings if the factorization needs to performed from scratch, since the corresponding decomposition times are not negligible. To account for this fact, the proposed Cholesky modifications (Sec. 3, 5.3) have been introduced.
5 Evaluation of algorithms

Figure 5.1: Decomposition runtimes of $R + \lambda I$ (Eq. 2.29) in the Offline step (Sec. 2.5) for different grid sizes and decomposition methods [2]. The x-axis is labeled by 'Grid size - (Dimension, Level)' due to the underlying sparse grid (Sec. 2.2).

5.2 Online step

Given a factorization of the system matrix (Eq. 2.29) and a data set $S$, one need to solve

$$\alpha = (R + \lambda C)^{-1}b,$$

(5.1)
denoted as Online step, to obtain the density declaring coefficients $\alpha$. Since different decomposition methods have been implemented for the Offline step (Sec. 5.1), different methods exist to solve Eq. 5.1 efficiently [2, 6]. The approach in case the Cholesky decomposition is chosen is presented in Sec. 3.1.

The Online step is in $O(N^2)$ for all decomposition methods [2, 6]. Solving for $\alpha$ in Eq. 5.1 takes nearly the same time for LU and Eigendecomposition based factorizations of Eq. 2.29, whereas the Cholesky based method of Sec. 3.1 needs some more time (Fig. 5.2). The lower triangular matrix of the LU factorization does have ones as diagonal entries, this is not given for the Cholesky factor. Therefore, more operations need to be applied to obtain $\alpha$. A more detailed discussion in terms of flops and memory accesses can be found in [6]. But however, even in larger grid settings ($\sim$ 5000 grid points) all runtimes are comparatively low (< 1 second). Therefore the runtime of the Online step doesn’t restrict the usage of an specific decomposition method.
5 Evaluation of algorithms

Figure 5.2: Time to solve Eq. 5.1 for different grid sizes and decomposition methods [2] given an artificial data set (Table 8.2). The x-axis is labeled by ‘Grid size - (Dimension, Level)’ due to the underlying sparse grid (Sec. 2.2).

5.3 Adaptivity

In Sec. 5.1 and Sec. 5.2 we elevated that the runtime of the adaptive data stream density estimation approach (Sec. 3.5), based on an Offline/Online splitting (Sec. 2.5), is mainly caused by the recurring Offline steps and the applied decomposition method. Therefore we developed different algorithms (Sec. 3), to employ modifications to the current Cholesky factor in case the grid is altered (Sec 3.2, 3.3) due to refinement and coarsening [1, 2].

5.3.1 Coarsening

Before we start with the evaluation of Alg. 5, we sketch the determination process of the parameter $c$ (Eq. 3.20, Sec. 3.3).

One can see in Figure 5.3 that a $c$ between $\frac{1}{20}N$ and $\frac{1}{10}N$ may be a good choice. Because if $c$ is chosen to be $\frac{1}{20}N$ the orange and sky-blue line run nearly constantly equal. Therefore, a rank-one update plus the corresponding right circular shift of row/column $\frac{N}{20}$ to row/column 1 (Eq. 3.15) takes approximately the same time as the left circular shift of row/column $\frac{N}{20}$ to row/column $N$ (Eq. 3.9). Since the applied rank-one updates in Alg. 5 are of order $O((N - n)^2)$ and therefore depend on the amount of removed points $n$, $c = \frac{1}{10}N$ is possible as well. If a lot points are removed, the corresponding red, dark-red and sky-blue line would be shifted downwards.

In summary, $c$ needs to be determined according to the amount of removed points and grid size, but an a priori selection between $\frac{1}{20}N$ and $\frac{1}{10}N$ would be an appropriate choice.

34
5 Evaluation of algorithms

Figure 5.3: Runtimes of Cholesky permutations (Alg. 3) in order to determine the threshold $c$ (Eq. 3.20) in the coarsening algorithm (Alg. 5)(Table 8.3, 8.4). The x-axis is labeled by 'Grid size - (Dimension, Level)' due to the underlying sparse grid (Sec. 2.2).

Now we continue with evaluating the coarsening method (Sec. 3.3, Alg. 5):

Figure 5.4: Remove first 50 (red), middle 50 (blue) and the grid points corresponding to indices $N - 500$ to $N - 400$ for different sparse grid settings (Alg. 5)(Table 8.5). The x-axis is labeled by 'Grid size - (Dimension, Level)' due to the underlying sparse grid (Sec. 2.2).

Since removing grid points corresponding to lower indices of the system matrix (Eq.
Evaluation of algorithms

3.7) is more costly than for the ones located in the last rows/columns (Figure 5.4), we can not recommend hard thresholds for the amount of grid points to remove according a certain grid size.

But if the underlying problem requires coarsening in the higher level grid points, our method would beat a new Cholesky decomposition by far in terms of runtime. However, if we are removing coarse grid points long runtimes can occur corresponding to delete these grid points.

All in all, Alg. 5 provides a fast approach to delete arbitrary minor amounts of grid points and if some knowledge about the coarsening domain is present, even for large quantities.

To boost the performance of Alg. 5, one could parallelize the calculation of applying a Givens rotation (Eq. 2.40) to specific columns of \( L \) (Eq. 2.49, 2.68, 2.69) (Alg. 3, Alg. 4). Since, this procedure is implemented via a for loop with independent loop runs.

5.3.2 Refinement

In the following the refinement method (Sec. 3.2, Alg. 2) is analyzed:

Adding a new grid point to the current Cholesky factorization \( L \) of the system matrix (Eq. 3.1) can be interpreted as the \( N+1 \) step of the Cholesky decomposition (Eq. 2.35, 2.36) of the extended system matrix

\[
\tilde{R} + \tilde{A}I \in \mathbb{R}^{(N+1) \times (N+1)}.
\]  

(5.2)

Based on this fact, any kind of refinement applied via Alg. 2 is faster than decompose the resulting system matrix from scratch. Since the required calculations in Alg. 2 are a subset of the ones needed to determine the Cholesky decomposition of the new system matrix.

Figure 5.5 shows the comparatively low runtime of adding a new grid point in comparison with a rank one update (Alg. 4) for the same grid size.

Therefore Alg. 2 enables clear runtime improvements in the refinement setting.
5 Evaluation of algorithms

Figure 5.5: The red line depicts adding a new grid point for a certain grid size (Alg. 2), whereas the blue line corresponds to an arbitrary rank one update (Alg. 4) (Table 8.6). The x-axis is labeled by 'Grid size - (Dimension, Level)' due to the underlying sparse grid (Sec. 2.2).

5.4 Cross-validation

If cross-validation is applied to determine the best regularization parameter regarding an accuracy measure, one has to test different $\lambda$. This can be achieved via Alg. 6 and Alg. 8. But both algorithms and the Cholesky decomposition itself are of the same order $O(N^3)$. Figure 5.6 clearly depicts that Alg. 6 and Alg. 8 are significantly slower than a new Cholesky decomposition with changed $\lambda$.

However, we proposed both methods, because aspects of Alg. 6 can be parallelized due to the applied rank-one modifications, to boost the performance and possibly beat the runtime of the corresponding Cholesky factorization. In case of Alg. 6:

1.  
\[
\begin{bmatrix}
 l & 0 & 0 & \sqrt{\lambda - \lambda} \\
 l & l & 0 & 0 \\
 l & l & l & 0 \\
\end{bmatrix}
U_1 \rightarrow 
\begin{bmatrix}
 \tilde{l} & 0 & 0 & 0 \\
 \tilde{l} & l & 0 & x \\
 \tilde{l} & \tilde{l} & l & x \\
\end{bmatrix}
U_2 \rightarrow 
\begin{bmatrix}
 \tilde{l} & 0 & 0 & 0 \\
 \tilde{l} & \tilde{l} & 0 & 0 \\
 \tilde{l} & \tilde{l} & l & x \\
\end{bmatrix}
U_3 \rightarrow \ldots \quad (5.3)
\]

2.  
\[
\begin{bmatrix}
 l & 0 & 0 & 0 \\
 l & l & 0 & \sqrt{\lambda - \lambda} \\
 l & l & l & 0 \\
\end{bmatrix}
U_2 \rightarrow 
\begin{bmatrix}
 l & 0 & 0 & 0 \\
 l & \tilde{l} & 0 & 0 \\
 l & \tilde{l} & l & x \\
\end{bmatrix}
U_3 \rightarrow 
\begin{bmatrix}
 l & 0 & 0 & 0 \\
 l & \tilde{l} & 0 & 0 \\
 l & \tilde{l} & \tilde{l} & 0 \\
\end{bmatrix} \quad (5.4)
\]
The equations above depict the rank-one updates (Eq. 3.22) applied in Sec. 3.4.1 in case $\lambda$ is increased. Since the transformations in the $i$-th step do not change the columns 1 to $i-1$, one can start with the $i$-th step immediately after the $(i-1)$-step applied changes to the $i$-th column of the Cholesky factor. Unfortunately Alg. 8 doesn’t enable such shortcuts.

But both methods strongly depend on applying Givens rotations (Eq. 2.40) to columns of $L$ (Eq. 2.49, 2.68, 2.69)(Alg. 4, Alg. 7). Since this procedure is implemented via a for loop with independent loop runs, one could use parallelization as well.

Figure 5.6: Comparison of one cross-validation step in case the regularization parameter $\lambda$ is increased (Alg. 6) (blue) or decreased (Alg. 8)(green) (Sec. 3.4) with computing a new Cholesky decomposition (red) if $\lambda$ is changed (Table 8.7). The x-axis is labeled by ’Grid size - (Dimension, Level)’ due to the underlying sparse grid (Sec. 2.2).
6 Application

One of the most important tasks in machine learning and data mining, is to build a robust and fast classifier for new incoming data. Given the training data

\[ S = \{(x_i, y_i)\}_{i=1}^M \subset \mathbb{R}^d \times \{1, \ldots, k\}, \]  

Bayesian classification is one approach to address this problem \[8\]. The class \( y \in \{1, \ldots, k\} \) of a data point \( x \in \mathbb{R}^d \) is determined by evaluating

\[ f(Y = y \mid x) = \frac{f(x \mid Y = y) \cdot p(Y = k)}{f(x)}, \quad \forall y \in \{1, \ldots, k\}, \]  

with prior \( p \). The class with the highest probability is assigned to the point \( x \). The class-conditional density functions need to be estimated from the data samples, before one can solve Eq. 6.2.

In the following chapter we will use Bayesian classification based on the introduced sparse grid density estimation approach (Sec. 2.4)\[10\] to build an adaptive classifier for data streaming. This method is applied to a real world data set later on.

6.1 Data-stream based classifier

Analogue to Sec. 3.5 the training data Eq. 6.1 is partitioned into subsets of the form

\[ S_1, S_2, \ldots S_g \text{ of size } m_1 \ldots m_g \text{ with } \sum \limits_{i=1}^{g} m_i = m \text{ and } S_i = \{(x_i, y_i)\}_{i=1}^{m_i}. \]  

In the first Offline step the same a priori sparse grid is chosen for all classes 1 to \( k \), in order to perform the costly Cholesky decomposition of the system matrix (Eq. 2.29) just once (Sec. 3.1).

The first data batch \( S_1 \) is processed and sorted according the classes, so we obtain

\[ S_{1j}, \quad \forall j \in \{1, \ldots, k\}, \quad \text{with } S_{1j} = \{(x_i, j)\}. \]  

Based on these class-specific data batches, the \( b_{1j} \) (Eq. 3.31) are computed. \( k \) linear equations of the form Eq. 3.2 need to be solved, to receive \( k \) class-conditional density functions (Eq. 6.2) based on

\[ \alpha_{1j}, \quad \forall j \in \{1, \ldots, k\}. \]
6 Application

to complete the first Online step. Due to these coefficients and an appropriate prior, Eq. 6.2 can be solved to classify new data, measure the accuracy of the classifier or perhaps to perform cross-validation based on a test set.

Until now we used an a priori sparse grid to build the density functions. To take specific characteristics of the classes into account, refinement and coarsening can be applied to every class separately (Sec. 2.3). The arising changes in the system matrix and therefore in the Cholesky factor are handled via the introduced algorithms in Sec. 3.2 and Sec. 3.3.

Based on the probably different Cholesky factors

\[ L_1, \ldots, L_k \]

and the updated \( b_{2j} \) (Sec. 3.5), the second Online step according to the data batch \( S_2 \) can be processed, for further details see [11].

6.2 Classification of the DR10 Dataset

To see the introduced data-stream classification method in action and to challenge the algorithms of Sec. 3, we are going to apply this approach to a real-world astronomical data set.

The DR10 data set from the Sloan Digital Sky Survey (SDSS) repository [17] contains four photometric properties of astronomical objects we will learn from, to classify them either as stars or quasars [17]. In particular, we stream-wise regard 640,000 data points to build a classifier and then apply the model to another 277,001 objects.

Following settings are used:

- The training data is portioned into 16 subsets with respective batch size of 40000.
- For refinement and coarsening the naive coefficient based approach (Sec. 2.3) is chosen.
- An a priori sparse grid of dimension 4 and level 7 with corresponding grid size of 7937 is selected.
- In every streaming step 150 points are regarded to remove and 5 points are considered to refine. Refinement of one grid point implies addition of all hierarchical children and related hierarchical parents (Figure 2.5).
- No prior is used.
- The regularization parameter \( \lambda \) (Sec. 3.4) is set to \( 1e^{-5} \).
- 195,131 objects of the test data-set are stars and 81,870 are quasars.
6 Application

- The threshold $c$ (Sec. 3.3, 5.3.1) is set to $N_{10}$.

Due to the most visibility in the changes of the grid, we are going to depict the corresponding sparse grids of class 1 (stars) in dimensions one and two. Whereas we will consider dimensions one and four for class 2 (quasars). Therefore the classifications are pictured in these two different settings as well.

![Figure 6.1: The test set distributions of the two classes in dim. one and dim. two (a) and in dim. one and dim. four (4) are depicted, whereas stars are red and quasars are blue.](image)

(a) Class distribution for dim. one and two  
(b) Class distribution for dim. one and four

Figure 6.1 illustrates the location of the different classes in two perspectives and is used to compare the classification results in the following. Figure 6.2 visualizes the underlying a priori sparse grid in two dimensions.

In the first Offline step the system matrix (Eq. 2.29) based on the a priori sparse grid (Dim. 4, Lvl. 7) is build and decomposed. Corresponding to a grid size of 7937, the Cholesky decomposition needs around 100 seconds to perform. The first data batch $S_1$ of size 40000 is loaded and processed into $b_{1,1}$ and $b_{1,2}$. In the following first Online step the desired $\alpha_{1,1}$ and $\alpha_{1,2}$ are computed (Sec. 6.1). Classification based on the $\alpha$'s yields a matching rate of 84.6%:

<table>
<thead>
<tr>
<th>Actual stars</th>
<th>Actual quasars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted stars</td>
<td>16724 14800</td>
</tr>
<tr>
<td>Predicted quasars</td>
<td>27857 67070</td>
</tr>
</tbody>
</table>

![Table 6.1: Classification result of first streaming step. A definition of the values can be found in Table 8.11.](image)

Table 6.1: Classification result of first streaming step. A definition of the values can be found in Table 8.11.
6 Application

Figure 6.2: A priori sparse grid (Dim. 4, Level 7) projected into two dimensions.

Figure 6.3: (a) pictures the classification based on the $\alpha$’s computed in the first Online step in dim. one and two, whereas (b) visualizes the result in dim. one and four. Stars are red and quasars are blue.
6 Application

Due to the $\alpha$‘s, refinement and coarsening based on the naive coefficient based approach is performed. The sparse grid of class 1 (stars) is reduced by 150 points, that takes around 38.8 seconds time to execute and is increased by 40 grid points, which needs 1.6 seconds to perform. The new grid size of class 1 is denoted by 7827.

The grid of class 2 (quasars) is reduced by 150 points as well (51.65 sec.) and increased by 39 grid points (1.56 sec.). The new grid size is 7826.

Obviously, both grids got smaller. The adaptivity criteria is adjusting the data-intensive areas of the underlying problem in the first streaming steps, therefore more grid points are removed than spent.

In streaming step 7 the grid size of class 1 (stars) is extended from 7944 to 8026, whereas 150 grid points are removed (47.15 sec.) and 232 are added (8.87 sec.). The according grid size of class 2 (quasars) is changed from 8317 to 8590, with 150 grid points removed (49.72 sec.) and 423 points added (17.69 sec.) (Figure 6.4). One can see that in both grids (Figure 6.4) the grid points near the border get coarsened. Whereas a lot refinement takes place in the area between 0.25 and 0.5 in dimension one and tow/four, because both classes have their main amount of data points in this domain. And due to the naive coefficient based refinement, this area gets refined the most.

Utilizing these grids the new data batch $S_8$ is processed into $b_{8,1}$ and $b_{8,2}$. In the following Online step $\alpha_{8,1}$ and $\alpha_{8,2}$ are obtained. Meanwhile 320000 data points are used for training.

Classification based on these $\alpha$‘s yields a matching rate of 88.73 % (Figure 6.5):

<table>
<thead>
<tr>
<th>Predicted stars</th>
<th>Actual stars</th>
<th>Actual quasars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>179851</td>
<td>15939</td>
</tr>
<tr>
<td>Predicted quasars</td>
<td>15280</td>
<td>65931</td>
</tr>
<tr>
<td></td>
<td>92.17%</td>
<td>80.53%</td>
</tr>
</tbody>
</table>

Table 6.2: Classification result of eighth streaming step. A definition of the values can be found in Table 8.11.

The last refinement and coarsening before the final evaluation of the classifier, is performed in streaming step 15. Due to these modifications the grid of class 1 (stars) reached a size of 10592, 150 grid points are removed (96.13 sec.) and 345 are added (22 sec.). The grid of class 2 (quasars) is slightly smaller with 10525 grid points, whereas 150 points are removed (110.4 sec.) and 414 grid points are added (25.9 sec.). The trends observed in the grids resulting from step 7 (Figure 6.4) continue. The borders of the domain get coarser, whereas in the area between 0.25 and 0.5 in dimension one and tow/four a large amount of grid points is spent (Figure 6.6).

Again, the new data batch $S_{16}$ is processed into $b_{16,1}$ and $b_{16,2}$ and the following Online step yields $\alpha_{16,1}$ and $\alpha_{16,2}$. All 640.000 training observations are now contribute to the classifier.
Figure 6.4: (a) visualizes the grid of class 1 (stars) used in the eight streaming step in dim. one and two, whereby (b) represents the corresponding modified sparse grid of class 2 (quasars) in dim. one and four. Red points label the removed grid points and blue ones indicate the added grid points in streaming step seven.

Figure 6.5: (a) pictures the classification based on the $\alpha$’s computed in the eight Online step in dim. one and two, whereas (b) visualizes the result in dim. one and four. Stars are red and quasars are blue.
Classification based on the final $\alpha$’s yields a matching rate of 88.77% (Figure 6.7):

<table>
<thead>
<tr>
<th></th>
<th>Actual stars</th>
<th>Actual quasars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted stars</td>
<td>179833</td>
<td>15813</td>
</tr>
<tr>
<td>Predicted quasars</td>
<td>15298</td>
<td>66057</td>
</tr>
<tr>
<td></td>
<td>92.16%</td>
<td>81.31%</td>
</tr>
</tbody>
</table>

Table 6.3: Classification result of last streaming step. A definition of the values can be found in Table 8.11.

Figure 6.6: (a) visualizes the grid of class 1 (stars) used in the last streaming step in dim. one and two, whereby (b) represents the corresponding modified sparse grid of class 2 (quasars) in dim. one and four. Red points label the removed grid points and blue ones indicate the added grid points in streaming step fifteen.
6 Application

Figure 6.7: (a) pictures the classification based on the $\alpha$'s computed in the last Online step in dim. one and two, whereas (b) visualizes the result in dim. one and four. Stars are red and quasars are blue.

In summary, the matching rate in streaming step 16 is just slightly better than the one in step 8. The reason for this is the fact that a refinement and coarsening method was chosen, which performs modifications in domains with the most data present. Therefore, classification critical regions may be omitted. Perhaps better results can be achieved if refinement is realized at the borders of the two classes.

But Figure 6.8 and 6.9 present the clear runtime improvements, if the algorithms regarding an adaptive Cholesky factorization (Sec. 3) are applied. Especially for larger grid sizes the differences of runtimes between the adjustment and a new Cholesky decomposition become significant.
Figure 6.8: Runtime analysis of class 1 (stars). The blue line visualizes the time needed to perform refinement in every streaming step, whereas the red one corresponds to the coarsening time of 150 grid points per step, the orange line depicts the sum of both. The green line is the time needed to perform a Cholesky decomposition from scratch. For according grid sizes and added points in every streaming step see Table 8.9.

Figure 6.9: Runtime analysis of class 2 (quasars). A description of the lines can be found in Figure 6.8. For according grid sizes and added points in every streaming step see Table 8.10.
7 Conclusion

Due to the Cholesky decomposition (Sec. 2.6) and modifications (Sec. 2.7) we build algorithms (Sec. 3) to tackle the problem of adaptivity in a sparse grid density estimation approach (Sec. 2.4). In a framework of an Offline/Online splitting (Sec. 2.5), we are now able to modify the underlying Cholesky factorization of the system matrix (Eq. 2.29) if the grid is refined or coarsened. Thus, the cost of an Offline step may be drastically reduced, since a new factorization does not need to be applied. With introducing the Cholesky factorization and related modifications, we enlarged the list of possible decomposition methods in the Offline step and provide a full theory and implementation to perform any kind of grid changes (Sec. 3).

We embedded this methods into a data-stream based density estimation (Sec. 3.5) and consequently into a data stream-based classifier as well (Sec. 6.1). Directly after a new data batch is processed and the density declaring coefficients are obtained in the corresponding Online step, we can include the new knowledge into the underlying sparse grid(s). After the learning is done via problem adjusted coarsening and refinement, the changes are induced into the Cholesky factor of the system matrix through our introduced algorithms.

Further work remains in the field of determining the optimal threshold $c$ (Sec. 3.3) of the coarsening Alg. 5. In Section 5.3.1 we showed that the optimal $c$ is based on the amount of removed points as well as on the underlying grid size.

Utilizing rank one updates of a Cholesky factor (Sec. 2.7.1), we developed methods to increase and decrease the regularization parameter $\lambda$ (Sec. 3.4). But for now this methods shouldn’t be applied for grid settings larger 1000 grid points, because computing a new Cholesky factor with changed $\lambda$ would be faster (Sec. 5.4). Nevertheless we presented these methods, since the kind of the applied rank one updates allows parallelization (Sec. 5.4). So, further research is needed in the referred topic of cross-validation to determine the optimal $\lambda$ even in larger grid settings.

The methods to adjust the Cholesky factor for coarsening (Alg. 5) and to change the regularization parameter $\lambda$ (Alg. 6,8) heavily depend on applying various given rotations (Definition 2.4). Since this procedure is implemented via a for loop with independent loop runs, one could try to integrate a parallelized version into the current implementation.

Until now all methods are implemented in the data mining toolbox libtool (Sec. 5). But, perspectivez they should be integrated into SG++ [1] as well.
Appendix

All tests have been performed on an Oracle VM Linux Mint operating system with following hardware:

- 3 processors of Intel(R) Core(TM) i7-4700HQ CPU @ 2.40 GHz
- 4877 MB memory

As compiler, GCC version 4.8.4 was used.

8.1 Examination of algorithms

Underlying measurements of the graphics in Sec. 5.:

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Level</th>
<th>Grid size</th>
<th>LU decomp.</th>
<th>Eigendecomp.</th>
<th>Cholesky decomp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>351</td>
<td>0.01</td>
<td>0.11</td>
<td>0.01</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>769</td>
<td>0.13</td>
<td>6.51</td>
<td>0.09</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1023</td>
<td>0.32</td>
<td>23.5</td>
<td>0.21</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1471</td>
<td>1.21</td>
<td>87.6</td>
<td>0.72</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1793</td>
<td>2.29</td>
<td>185.3</td>
<td>1.29</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2561</td>
<td>7.07</td>
<td>563</td>
<td>3.80</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2815</td>
<td>9.90</td>
<td>-</td>
<td>4.90</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>4097</td>
<td>30.4</td>
<td>-</td>
<td>15.0</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5530</td>
<td>72.8</td>
<td>-</td>
<td>34.4</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7937</td>
<td>-</td>
<td>-</td>
<td>99.7</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>9217</td>
<td>-</td>
<td>-</td>
<td>157</td>
</tr>
</tbody>
</table>

Table 8.1: The corresponding runtimes in seconds of the LU decomposition (Col. 4), Eigendecomposition (Col. 5) and the Cholesky decomposition (Col. 6) for certain grid settings.
## Appendix

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Level</th>
<th>Grid size</th>
<th>Eigendecomp.</th>
<th>Cholesky decom.</th>
<th>LU decom.</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>4</td>
<td>351</td>
<td>0.0005</td>
<td>0.0009</td>
<td>0.0006</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>769</td>
<td>0.004</td>
<td>0.0036</td>
<td>0.0025</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>1023</td>
<td>0.005</td>
<td>0.005</td>
<td>0.004</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1471</td>
<td>0.007</td>
<td>0.012</td>
<td>0.006</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1793</td>
<td>0.008</td>
<td>0.020</td>
<td>0.007</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>2561</td>
<td>0.014</td>
<td>0.051</td>
<td>0.012</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2815</td>
<td>-</td>
<td>0.060</td>
<td>0.013</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>4097</td>
<td>-</td>
<td>0.153</td>
<td>0.030</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5530</td>
<td>-</td>
<td>0.271</td>
<td>0.068</td>
</tr>
</tbody>
</table>

Table 8.2: The corresponding runtimes in seconds for solving the Online step (Eq. 3.2) based on LU (Col. 6), Eigendecomposition (Col. 4) and the Cholesky decomposition (Col. 5) for certain grid settings.

<table>
<thead>
<tr>
<th>Grid size</th>
<th>Rank one update</th>
<th>Perm. 1 to N</th>
<th>Perm. N/10 to N</th>
<th>Perm. N/20 to N</th>
</tr>
</thead>
<tbody>
<tr>
<td>2561</td>
<td>0.061</td>
<td>0.117</td>
<td>0.084</td>
<td>0.086</td>
</tr>
<tr>
<td>4097</td>
<td>0.168</td>
<td>0.221</td>
<td>0.188</td>
<td>0.192</td>
</tr>
<tr>
<td>5503</td>
<td>0.284</td>
<td>0.390</td>
<td>0.317</td>
<td>0.344</td>
</tr>
<tr>
<td>7937</td>
<td>0.609</td>
<td>0.814</td>
<td>0.656</td>
<td>0.703</td>
</tr>
<tr>
<td>9217</td>
<td>0.875</td>
<td>1.121</td>
<td>0.918</td>
<td>0.969</td>
</tr>
</tbody>
</table>

Table 8.3: The corresponding runtimes in seconds of an arbitrary rank one update (Col. 2)(Alg. 4), of a permutation of column/row one to column/row N (Col. 3)(Alg. 3), a permutation of N/10 to N (Col. 4) and a permutation of N/20 to N (Col. 5) for certain grid sizes.

<table>
<thead>
<tr>
<th>Dim. level</th>
<th>Grid size</th>
<th>Remove first 50</th>
<th>Remove middle 50</th>
<th>Remove N-500 to N-400</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>1023</td>
<td>0.130</td>
<td>0.095</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1793</td>
<td>0.733</td>
<td>0.232</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2561</td>
<td>1.91</td>
<td>0.626</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>4097</td>
<td>6.62</td>
<td>2.28</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5503</td>
<td>13.4</td>
<td>4.12</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7937</td>
<td>28.9</td>
<td>9.28</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>9217</td>
<td>41.4</td>
<td>13.4</td>
</tr>
</tbody>
</table>

Table 8.5: The corresponding runtimes in seconds of removing the first 50 (Col. 4), middle 50 (Col. 5) and the grid points corresponding to indices N – 500 to N – 400 (Col. 6) for certain grid settings.
Table 8.4: The corresponding runtimes in seconds of a permutation of column/row $N/10$ to column/row 1 (Col. 2), a permutation of $N/20$ to $N1$ (Col. 3), a permutation of $N/10$ to 1 + an arbitrary rank one update (Col. 3) and a permutation of $N/20$ to 1 + an arbitrary rank one update for certain grid sizes.

<table>
<thead>
<tr>
<th>Dim.</th>
<th>level</th>
<th>Grid size</th>
<th>Cholesky decomp.</th>
<th>Increase $\lambda$</th>
<th>Decrease $\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>1793</td>
<td>1.37</td>
<td>8.67</td>
<td>12.411</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2561</td>
<td>3.88</td>
<td>33.1</td>
<td>42.8</td>
</tr>
</tbody>
</table>

Table 8.7: The corresponding runtimes in seconds of increasing the regularization parameter $\lambda$ based on Alg. 6 (Col. 5), decreasing $\lambda$ based on Alg. 8 (Col. 6) and the Cholesky decomposition in case $\lambda$ is changed (Col. 4) for certain grid settings.

<table>
<thead>
<tr>
<th>Dim.</th>
<th>level</th>
<th>Grid size</th>
<th>Add one grid point</th>
<th>Rank one update</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>1023</td>
<td>0.002</td>
<td>0.004</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>1793</td>
<td>0.004</td>
<td>0.022</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2561</td>
<td>0.007</td>
<td>0.061</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>4097</td>
<td>0.020</td>
<td>0.168</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5503</td>
<td>0.045</td>
<td>0.284</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>7937</td>
<td>0.085</td>
<td>0.619</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>9217</td>
<td>0.108</td>
<td>0.875</td>
</tr>
</tbody>
</table>

Table 8.6: The corresponding runtimes in seconds of adding one grid point (Col. 4) (Alg. 2) and an arbitrary rank one update (Col. 5) (Alg. 4) for certain grid settings.
8.2 Examination of parameter $c$

Ratios of swaps and Givens rotations in Alg. 5 in order to determine $c$ (Sec. 3.3):

![Graph showing runtime ratios of swaps and Givens rotations](image)

Figure 8.1: Runtime ratios of swaps and Givens rotations in Alg. 5. The blue line visualizes the relation between the applied swaps and the Givens rotation in case of a permutation of grid point $\frac{N}{8}$ to $N$, the green one the same relation in case of grid point $\frac{N}{10}$ to $N$ and the orange one in case 1 to $N$. For larger grid settings the swap time approaches $\frac{1}{5}$ of the time needed to perform the Givens rotations (Eq. 3.19). The red line depicts the ratio of the runtimes of an arbitrary rank one update and the Givens rotations in the $\frac{N}{10}$ to $N$ permutation case. The underlying observations can be found in Table 8.8.

<table>
<thead>
<tr>
<th>G. size</th>
<th>Rank one update</th>
<th>Swaps N/8 to N</th>
<th>Swaps N/10 to N</th>
<th>Swaps 1 to N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023</td>
<td>1.72</td>
<td>0.39</td>
<td>0.48</td>
<td>0.40</td>
</tr>
<tr>
<td>1793</td>
<td>0.79</td>
<td>0.47</td>
<td>0.32</td>
<td>0.48</td>
</tr>
<tr>
<td>2561</td>
<td>0.97</td>
<td>0.30</td>
<td>0.32</td>
<td>0.34</td>
</tr>
<tr>
<td>4097</td>
<td>1.10</td>
<td>0.22</td>
<td>0.23</td>
<td>0.21</td>
</tr>
<tr>
<td>5503</td>
<td>1.11</td>
<td>0.28</td>
<td>0.24</td>
<td>0.22</td>
</tr>
<tr>
<td>7937</td>
<td>1.16</td>
<td>0.25</td>
<td>0.25</td>
<td>0.23</td>
</tr>
<tr>
<td>9217</td>
<td>1.15</td>
<td>0.24</td>
<td>0.21</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 8.8: Runtime ratios of swaps and Givens rotations in Alg. 5 for certain grid sizes (Col. 1). The listed ratios can be found in the headers of Col. 2 to 5.
8.3 DR10 classification measurements

Runtime and adaptivity measurements obtained in Section 6.2:

<table>
<thead>
<tr>
<th>Step</th>
<th>Grid size</th>
<th>Coarsening</th>
<th>Add. points</th>
<th>Refinement</th>
<th>Cholesky</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7827</td>
<td>51.65</td>
<td>40</td>
<td>1.60</td>
<td>98.88</td>
<td>53.25</td>
</tr>
<tr>
<td>2</td>
<td>7758</td>
<td>37.03</td>
<td>81</td>
<td>3.04</td>
<td>-</td>
<td>40.08</td>
</tr>
<tr>
<td>3</td>
<td>7759</td>
<td>35.09</td>
<td>151</td>
<td>5.48</td>
<td>-</td>
<td>40.57</td>
</tr>
<tr>
<td>4</td>
<td>7841</td>
<td>32.64</td>
<td>232</td>
<td>8.32</td>
<td>99.64</td>
<td>40.96</td>
</tr>
<tr>
<td>5</td>
<td>7866</td>
<td>36.75</td>
<td>175</td>
<td>6.52</td>
<td>-</td>
<td>43.27</td>
</tr>
<tr>
<td>6</td>
<td>7944</td>
<td>38.14</td>
<td>228</td>
<td>8.44</td>
<td>-</td>
<td>46.58</td>
</tr>
<tr>
<td>7</td>
<td>8026</td>
<td>47.15</td>
<td>232</td>
<td>8.87</td>
<td>-</td>
<td>56.02</td>
</tr>
<tr>
<td>8</td>
<td>8126</td>
<td>43.39</td>
<td>250</td>
<td>9.89</td>
<td>109.48</td>
<td>53.28</td>
</tr>
<tr>
<td>9</td>
<td>8300</td>
<td>44.93</td>
<td>324</td>
<td>12.82</td>
<td>-</td>
<td>57.75</td>
</tr>
<tr>
<td>10</td>
<td>8912</td>
<td>45.77</td>
<td>762</td>
<td>32.88</td>
<td>-</td>
<td>78.65</td>
</tr>
<tr>
<td>11</td>
<td>9215</td>
<td>58.18</td>
<td>453</td>
<td>21.82</td>
<td>158.00</td>
<td>80.00</td>
</tr>
<tr>
<td>12</td>
<td>9680</td>
<td>70.09</td>
<td>615</td>
<td>32.00</td>
<td>-</td>
<td>102.09</td>
</tr>
<tr>
<td>13</td>
<td>9953</td>
<td>86.27</td>
<td>423</td>
<td>23.70</td>
<td>-</td>
<td>109.97</td>
</tr>
<tr>
<td>14</td>
<td>10397</td>
<td>91.60</td>
<td>594</td>
<td>35.29</td>
<td>224.20</td>
<td>126.89</td>
</tr>
<tr>
<td>15</td>
<td>10592</td>
<td>96.13</td>
<td>345</td>
<td>22.00</td>
<td>-</td>
<td>118.1</td>
</tr>
</tbody>
</table>

Table 8.9: Data stream measurements for class 1 (stars) in Sec. 6.2. Col. 3 contains the time needed for removing 150 grid points, Col. 4 the added points, Col. 5 time to add this points, Col. 6 the corresponding Cholesky decomposition and Col. 7 the sum of Col. 3 and Col. 5 for certain streaming steps (Col. 1) and grid sizes (Col. 2). All runtimes are measured in seconds.
Table 8.10: Data stream measurements for class 1 (quasars) in Sec. 6.2. Col. 3 contains the time needed for removing 150 grid points, Col. 4 the added points, Col. 5 time to add this points, Col. 6 the corresponding Cholesky decomposition and Col. 7 the sum of Col. 3 and Col. 5 for certain streaming steps (Col. 1) and grid sizes (Col. 2). All runtimes are measured in seconds.

<table>
<thead>
<tr>
<th>Step</th>
<th>Grid size</th>
<th>Coarsening</th>
<th>Add. points</th>
<th>Refinement</th>
<th>Cholesky</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7826</td>
<td>51.65</td>
<td>39</td>
<td>1.62</td>
<td>98.88</td>
<td>53.27</td>
</tr>
<tr>
<td>2</td>
<td>7770</td>
<td>40.38</td>
<td>94</td>
<td>3.48</td>
<td>-</td>
<td>43.86</td>
</tr>
<tr>
<td>3</td>
<td>7744</td>
<td>41.41</td>
<td>124</td>
<td>4.79</td>
<td>-</td>
<td>46.20</td>
</tr>
<tr>
<td>4</td>
<td>7875</td>
<td>36.38</td>
<td>281</td>
<td>10.24</td>
<td>99.64</td>
<td>46.62</td>
</tr>
<tr>
<td>5</td>
<td>8013</td>
<td>33.87</td>
<td>288</td>
<td>10.87</td>
<td>-</td>
<td>44.74</td>
</tr>
<tr>
<td>6</td>
<td>8317</td>
<td>46.54</td>
<td>454</td>
<td>18.62</td>
<td>-</td>
<td>65.16</td>
</tr>
<tr>
<td>7</td>
<td>8590</td>
<td>49.72</td>
<td>423</td>
<td>17.69</td>
<td>-</td>
<td>67.41</td>
</tr>
<tr>
<td>8</td>
<td>8893</td>
<td>54.71</td>
<td>453</td>
<td>20.22</td>
<td>140.60</td>
<td>74.93</td>
</tr>
<tr>
<td>9</td>
<td>9127</td>
<td>61.93</td>
<td>384</td>
<td>18.68</td>
<td>-</td>
<td>80.61</td>
</tr>
<tr>
<td>10</td>
<td>9352</td>
<td>70.79</td>
<td>375</td>
<td>18.59</td>
<td>-</td>
<td>89.38</td>
</tr>
<tr>
<td>11</td>
<td>9586</td>
<td>79.12</td>
<td>384</td>
<td>20.27</td>
<td>-</td>
<td>99.39</td>
</tr>
<tr>
<td>12</td>
<td>9772</td>
<td>88.55</td>
<td>336</td>
<td>18.38</td>
<td>-</td>
<td>106.93</td>
</tr>
<tr>
<td>13</td>
<td>10036</td>
<td>87.72</td>
<td>414</td>
<td>24.40</td>
<td>212.50</td>
<td>112.12</td>
</tr>
<tr>
<td>14</td>
<td>10261</td>
<td>90.10</td>
<td>375</td>
<td>22.64</td>
<td>-</td>
<td>112.74</td>
</tr>
<tr>
<td>15</td>
<td>10525</td>
<td>110.36</td>
<td>414</td>
<td>25.91</td>
<td>237.50</td>
<td>136.27</td>
</tr>
</tbody>
</table>

Table 8.11: Classification evaluation of a DR10-classifier

<table>
<thead>
<tr>
<th>Predicted stars</th>
<th>Actual stars</th>
<th>Actual quasars</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>Predicted quasars</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>g</td>
<td>h</td>
</tr>
</tbody>
</table>

- a := How many predicted stars are actually stars
- b := How many predicted stars are quasars
- c := How many predicted quasars are stars
- d := How many predicted quasars are actually quasars
- e := Stars predictive value := \( \frac{a}{a+b} \cdot 100\% \)
- f := Quasars predictive value := \( \frac{d}{c+d} \cdot 100\% \)
- g := True stars rate := \( \frac{a}{a+c} \cdot 100\% \)
- h := True quasars rate := \( \frac{d}{b+d} \cdot 100\% \)
Bibliography


