A High-Performance Computing Approach to Approximate Bayesian Inference

Doctoral Dissertation submitted to the
Faculty of Informatics of the Università della Svizzera italiana
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

presented by
Lisa Gaedke-Merzhäuser

under the supervision of
Prof. Olaf Schenk

June 2024

Dissertation Committee

Prof. Ernst Wit	Università della Svizzera italiana, Switzerland
Prof. Michael Multerer	Università della Svizzera italiana, Switzerland
Prof. Håvard Rue	King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
Prof. Gerhard Wellein	Friedrich-Alexander-Universität, Erlangen-Nürnberg, Germany

Dissertation accepted on 17 June 2024

Research Advisor	PhD Program Director
Prof. Olaf Schenk	**Prof. Walter Binder, Prof. Stefan Wolf**

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Lisa Gaedke-Merzhäuser
Lugano, 17 June 2024

Tutti noi sentiamo che se una sala ha una buona metà del suo pavimento sgombro, essa dà un senso di sollievo: sembra che ci prometta la confortante possibilità di *muoverci*.

We all know the sense of comfort of which we are conscious when a good half of the floor space in a room is unencumbered; this seems to offer us the agreeable possibility of *moving* about freely.

Maria Montessori, 1917

Abstract

There is a growing demand for performing large-scale Bayesian inference tasks, arising from greater data availability and higher-dimensional model parameter spaces. The methodology of integrated nested Laplace approximations (INLA) provides a popular and reliable paradigm for performing inference applicable to a large subclass of additive Bayesian hierarchical models. The work presented in this thesis is dedicated to the integration and development of high-performance computational methods for the INLA framework. The main focus is twofold. The first objective is to improve the performance of the computational bottleneck operations, which consist of Cholesky factorizations, solving linear systems, and selected matrix inversions. We present two numerical solvers to handle these operations, a sparse CPU-based library and a novel blocked GPU-accelerated approach. Second, we establish parallelization strategies that target multi-core architectures (single node), making use of nested thread-level parallelism. For particularly large-scale applications, which arise in the context of spatio-temporal phenomena, we additionally put forward a performant distributed memory variant (multi node), capable of handling models with millions of latent parameters. We showcase the accuracy and performance of our proposed works on synthetic as well as real-world applications.

Acknowledgements

When I started this PhD four years ago, it seemed like I had all the time in the world to study and learn, to really become an expert. And then the days and nights, the week days and weekends, the summers and winters passed, and suddenly I was supposed to be done. And I am. I guess. I hope. Or maybe I don't, because it also means that an incredible era of my life is coming to an end. An amazing, unforgettable, formative and inspiring time. Something that surprised me, is how much research seems to be about learning to be patient. To chip away one little stone at a time, while neither losing sight of the big picture nor getting discouraged by its size. Undoubtedly a very valuable skill to improve or possess, independently of what life might bring. Just as important as the expert knowledge I have acquired over these years. The submission of this thesis marks the end of a chapter in my life, and I would like to take a moment to express my gratitude to all those who have lead, accompanied and supported me along the way.

I would like to thank my advisor, Prof. Olaf Schenk, for your oversight and guidance, for your scientific and personal advice. Thank you for the trust and freedom you have provided me with, and for opening doors I didn't even know existed. I would also like to thank Prof. Håvard Rue, who has been a great mentor to me. Thank you for the countless hours you have spent on video calls explaining statistical concepts to me, for never growing tired of my endless questions, for always having time for a chat, for all the experience and knowledge you have shared with me. Thank you for all the positive encouragement and confidence you have given me. I would also like to express my gratitude to Prof. Gerhard Wellein, Prof. Ernst Wit and Prof. Michael Multerer for taking the time and effort to be part of my committee. I would like to thank you for your insightful comments on this document and your constructive questions during my visits and presentations. Over the past years, I have also had the chance to meet, work and collaborate with many other inspiring people, here at USI, and during my brief or extended visits at KAUST, ICL, FAU, NYU, KIT and ETH. Thank you for your hospitality, for the constructive discussions, for your help, advice, and input. I

also want to thank those who have shared the office with me every day, for the countless coffee breaks and lunches, the little chats, for listening to all my work and non-work related problems. It has been a pleasure and continues to brighten my days.
I have also been incredibly fortunate to receive tremendous amounts of non-scientific support. I would like to express my gratitude to all those who have believed in the completion of this thesis long before I did, who knew, that patiently and persistently, I would get there. I could not have done it without you. Lugano and especially Gandria, you have become such an amazing home to me. It has been an honor to have met all of you different wonderful people. Thank you for all the days and nights that we have shared. For all the funny moments, the big discussions, the stupidities, the joy, and laughter. May there be more. I am also incredibly grateful for all the support that I have received from afar. From those who have unceasingly encouraged and cared for me, over the last years and especially during these past months and weeks. Thank you. From those who have listened to me for hours even though they probably didn't really know what I was talking about or why this was bothering me in the first place. I guess I might have involuntarily made you become more patient as well. I also want to thank the amazing women who have been by my side since the very first semester of university, who sat beside me on those uncomfortable foldable wooden benches that characterize German universities and who, like me, did not understand at all what was happening on the blackboard. Thank you for struggling through this with me, without you, I would have given up before (or otherwise at least immediately after) our very first exam session. I am so proud of every single one of us and so happy to have met you. I want to thank my parents and especially my mother, for all their love, support, patience (there it is again), and just being who you are. Watching and learning from how you navigate life has always been and continues to be an inspiration to me.

Finally, I would like to say that I am very grateful to have had the privilege to pursue this degree, sheltered away from most worries of this world, just being able to focus on making stuff run fast on GPUs. May hopefully something good come from it.

Contents

Chapter 1

Introduction

The overarching aim of this thesis is to build bridges between the realms of Bayesian statistics and high-performance computing. There is a growing demand for performing large-scale Bayesian inference tasks, arising from greater data availability and higher-dimensional model parameter spaces. Bayesian statistics offers a systematic approach to quantify uncertainty and combining prior knowledge with new data. From the statistical side, the main focus of this thesis is on the methodology of integrated nested Laplace approximations (INLA) [1], which offers a flexible framework for performing complex Bayesian inference tasks [2, 3, 4]. INLA is applicable to a wide subclass of additive Bayesian hierarchical models. Among its broad range of applications are climate and weather modeling [5], disease mapping [6, 7], medical image analysis [8, 9], traffic management [10, 11], environmental changes [12, 13, 14] and social disparity studies [15, 16]. Since its inception, combining computational efficiency with a high level of accuracy has been its main objective. A crucial aspect is its focus on sparse precision, i.e., inverse covariance, instead of covariance matrices. From a theoretical standpoint, they contain the same information, as one can directly be derived from the other. While covariance matrices store general dependency structures between random variables, the entries of a precision matrix encode conditional dependencies. This can bring advantages from a modeling viewpoint as it not only lays out which variables influence each other but also comprises information on how they are connected. Due to the type of information they contain, precision matrices are often naturally sparse or can be approximated as such, bearing the potential for large computational advantages.

From a computational or algorithmic viewpoint, the INLA methodology poses an interesting challenge with manifold opportunities. The first step to obtaining inference estimates entails solving a nonlinear optimization problem. Its objective function is computationally expensive to evaluate and its gradient not easily accessible. The computational bottleneck in the evaluation of the objective function consists of (high-dimensional) Cholesky decompositions of sparse matrices with recurring sparsity patterns. After determining the optimum, a fundamental step consists of estimating the posterior marginal variances. This induces an additional bottleneck as it requires the selected inversion of (high-dimensional) matrices with the same sparsity patterns. A recurring characteristic is the abundant repetition of the computational bottleneck operations with changing numerical values while the overall sparsity patterns remain the same. This allows for sophisticated preprocessing and the introduction of parallelism on various levels.

The following thesis explores these challenges and opportunities in detail and provides more scalable and performant implementations of the INLA methodology which allow for modeling more realistic and thus more complex phenomena at shorter run times. In the first chapter, a brief introduction to Bayesian statistics will be presented, followed by an overview of different inference methods. We also discuss their associated challenges, identifying computational (in)feasibility as a major concern. This chapter concludes with a general outline of the subsequent thesis.

1.1 Bayesian Inference

Bayesian inference originates from the work of Thomas Bayes and Pierre-Simon Laplace during the 18th century. According to Jaynes [17], Bayesian statistics offers a systematic approach to update beliefs and make predictions in the presence of uncertainty. At its core, it revolves around Bayes' theorem which establishes a relationship between conditional probabilities, allowing for the adaptation of prior beliefs in the presence of new evidence or data. In contrast with frequentist statistics, where model parameters are assumed to be fixed but unknown, the Bayesian approach treats all parameters as random variables with associated probability distributions, which provides a natural way to quantify uncertainty. More concretely, this means given a prior distribution $p(\boldsymbol{x})$ which encapsulates existing beliefs; a likelihood function $p(\boldsymbol{y}|\boldsymbol{x})$, describing the probability of observing data given a set of parameter values; and the marginal likelihood or evidence $p(\boldsymbol{y})$, representing the probability of observing the data over all possible parameter values; we can obtain the posterior distribution $p(\boldsymbol{x}|\boldsymbol{y})$, which denotes the updated beliefs after incorporating the new data, as given in the following equation

$$p(\boldsymbol{x}|\boldsymbol{y}) = \frac{p(\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x})}{p(\boldsymbol{y})}. \tag{1.1.1}$$

Adopting a Bayesian approach is especially valuable in scenarios where the observed data is limited, or prior information is available. From Chapter 2 onward we will be considering Bayesian hierarchical models, which comprise two types of parameters, latent variables which correspond to $\boldsymbol{x}$ as defined above and hyperparameters $\boldsymbol{\theta}$ which influence $p(\boldsymbol{x})$ and $p(\boldsymbol{y}|\boldsymbol{x})$. This will also be reflected in an extended version of Bayes' theorem that we will consider instead, where the general principle remains, however, exactly the same. From a theoretical perspective, obtaining $p(\boldsymbol{x}|\boldsymbol{y})$ from 1.1.1 is relatively straight-forward, as all terms on the right-hand side are well-defined. Deducing the posterior marginal distributions is, in practice, however, often a challenge. The mathematical equation describing the posterior distribution is often complicated, not existing in closed form, difficult to evaluate as well as high-dimensional, thus making direct inference impossible [18]. It is typically also only known up to a constant term which is independent of the unknown parameters. In particular, the marginal likelihood $p(\boldsymbol{y})$ is usually not available in closed form, but computed by integrating the likelihood over all possible parameter values. This integral is often analytically intractable, thus requiring numerical integration, which is, however, complicated for high-dimensional parameter spaces. Therefore, Bayesian statistics requires specialized solution methods in order to perform inference after all.

1.2 Related Works

The most used class of Bayesian inference methods are Markov Chain Monte Carlo (MCMC) methods [18]. Their beginnings date back to the middle of the 20th century [19] but they did not gain wide popularity until the 1990s with the surge of compute power [20]. The basic underlying idea is to construct a Markov chain which has the desired posterior distribution as its equilibrium. A sequence or chain of correlated samples is produced, where each new sample, as the term Markov chain suggests, only depends on its predecessor. Once the chain has converged to its stationary distribution, an estimate of the posterior is obtained using Monte Carlo integration, which is a stochastic technique for approximating integrals using random sampling. From a theoretical viewpoint, MCMC methods become arbitrarily accurate when sampling for long enough [18]. Exploring the solution space efficiently is, however, non-trivial. Instead of computing the joint posterior distribution, it can be advantageous to focus on the marginal posterior distributions of each individual parameter, which are obtained by integrating over the remaining parameters. Among the most common transition kernels for generating a sequence of random samples are various variations of the Metropolis-Hastings [19, 21] algorithm which iteratively proposes a new sample based on the current one (Markov property). This sample is then, with some probability, either accepted or rejected for the next iteration. One particular variant is Hamiltonian Monte Carlo (HMC) [22, 23] which makes use of gradient information of the target distribution to propose more efficient parameter updates. It uses Hamiltonian dynamics by extending the state space using auxiliary “momentum” variables and volume-preserving numerical integration. Another possibility is particle filtering [24, 25, 26], also known as sequential Monte Carlo. It approximates the posterior densities of interest using a swarm of weighted particles, where each particle represents a sample from the posterior distribution. The particles are sequentially updated as new observations become available using importance sampling techniques to readjust the weights. A comprehensive overview of MCMC algorithms can e.g. be found in [27, 28]. Various MCMC-based software packages are available today. They have heavily contributed to the increased adoption of Bayesian statistics across many scientific disciplines, by enabling applied users to perform more complex inference tasks. Among the most popular packages are BUGS [29], JAGS [30], PyMC [31] and Stan [32].

A popular class of approximate Bayesian methods, especially prevalent in the field of machine learning, is variational inference, see e.g. [33, 34, 35] for an overview. The principal idea is to posit a family of simpler or variational distributions over the parameters of interest to approximate the true posterior. Multivariate normal distributions are a standard choice, where one often additionally assumes that some (or all) variables are independent of each other. Then an optimization problem is constructed over the set of unknown variational parameters whose optimum is the “closest” member in the family to the true distribution of interest. Closeness is defined in terms of the Kullback-Leibler divergence, whose minimization is equivalent to maximizing the Evidence Lower Bound. During the optimization, the parameters are iteratively updated until convergence. The quality of the approximation and the complexity of the involved computations depend on the choice of family of variational distributions. The aim is to choose a family that is flexible enough to adequately approximate the true posterior, while remaining simple enough to allow for efficiently solving the optimization problem. A known issue of variational inference is its tendency to underestimate the posterior variance [35].

MCMC and variational approaches are the most frequently used methodologies for performing inference. INLA, like the latter, is an approximate Bayesian inference method. INLA is applicable to the class of latent Gaussian models (LGMs), which will be formally introduced in Section 2.1.3, making it less general than the previous two. LGMs, nevertheless, comprise a large variety of commonly used statistical models. The INLA methodology and its underlying concepts will be discussed in detail in the following chapter. There additionally exist a number of popular empirical Bayesian approaches. For example empirical Bayes' methods, which lie at the intersection of frequentist and Bayesian statistics, as they estimate prior probabilities from the data, which contradicts the fully Bayesian perspective, but are often used as an approximation. Among them are Laplace approximations, which are also used within the INLA paradigm. Especially in spatial statistics, Gaussian process regression, also known as Kriging, is a popular interpolation method using Gaussian processes [36, 37]. As the name suggests, one assumes the parameter as well as the data to be normally distributed. This makes the problem tractable (or leads to an optimization problem when hyperparameters are present in the hierarchical modeling case). The results obtained by INLA[1] will coincide with those obtained by Gaussian process regression and be exact up to a constant, as it is a special case of INLA's larger framework. For an exhaustive overview of various Bayesian inference methods, we e.g. refer to [38, 39, 40].

1.3 Computational Challenges

While Bayes' theorem dates back to the 18th century, for a long time Bayesian statistics was restricted to relatively simple models. These models relied on cases where analytical solutions were available, conjugate priors were used or the number of parameters was very limited, due to the difficulty of performing calculations involving high-dimensional integrals or complex probability distributions. The rapid technological advancements over the last decades have allowed for performing previously unthinkable inference tasks. At the same time, greater data availability and more sophisticated models with higher-dimensional parameter spaces continue to create a growing demand for performing larger scale inference tasks. Thus, computational complexity remains the major challenge in Bayesian statistics, while the specific computational bottlenecks depend on the particular method employed. This is especially true for spatial and spatio-temporal models, where the dimension of the parameter space or the number of

[1]of course only if the exact same model is chosen which is typically not the case, as INLA usually works with an SPDE approach for spatial models which we will be discussed in detail in Section 2.2.4 while standard Kriging approaches often work with a Gaussian similarity kernel.

observations quickly grows with the space or space-time domain.

Especially, MCMC methods are known for their notoriously bad scaling [20]. The rate of convergence exhibited by MCMC methods often slows down as the space of possible solutions increases. One requires large numbers of consecutive samples, leading to long run times. Many remedies have been proposed to alleviate this problem, which nevertheless persists to exist. Some operations can be parallelized, but the sampling within each chain remains largely sequential.

Variational inference methods generally scale much better than MCMC approaches, but this typically comes with a loss in accuracy, as the improved scalability is a result of choosing simpler variational distributions. Classical variational inference methods often give rise to large dense systems of equations which scale cubically with the number of observations [41] leading to large associated computational cost. A plethora of ideas has been proposed to overcome this computational burden. We will restrict our brief overview to different approaches tailored to spatial and spatio-temporal data. A simple but very common technique is to only consider separable space-time covariance functions, which result in space-time covariance matrices that are assembled as Kronecker products of purely temporal and spatial matrices. This restriction limits the choice to simplistic models but allows to dramatically reduce the associated computational cost through this separability. Purely spatial covariance matrices can be efficiently approximated using hierarchical matrices [42, 43, 44]. A different perspective is taken by state-space approaches which avoid working with dense covariance matrices altogether, by considering the temporal dynamics [45]. They are, however, restricted in the class of available models as they do not accommodate for global variables. The stochastic partial differential equation (SPDE) approach [46, 47, 48], which will be presented in Section 2.2.4, derives the spatio-temporal precision matrix (inverse covariance matrix) from an SPDE which is discretized using the finite element method. The resulting precision matrices are naturally sparse (in contrast to their dense inverses) while still allowing for fixed effects. We consider a non-separable spatio-temporal model based on [49], which estimates the considered time span jointly, directly incorporating all available observations, while also easily allowing for time–independent fixed effects. This SPDE-based model formulation of the spatio-temporal random field gives rise to high-dimensional sparse precision matrices and thus aligns well with the overall INLA framework, whose computational challenges and opportunities will be discussed extensively throughout this thesis.

1.4 Outline

The next chapter introduces the relevant statistical concepts involved, as well as the fundamental ideas behind the INLA methodology and an algorithmic overview of the essential steps. This lays the ground for an in-depth analysis from the computational side. We examine how important concepts from both fields are interconnected, although usually formulated very differently. Further, we discuss how these connections can be leveraged to facilitate more efficient algorithmic designs and make better use of recent hardware developments. This includes a discussion of the arising computational bottleneck operations which will be presented in Chapter 3 along with novel implementations of how to handle them. In Chapter 4 we present opportunities for parallelism and put forward different multi-layer parallel schemes that operate on multi-core as well as distributed architectures. We showcase different applications which include medical imaging data, a drug efficiency study and air temperature prediction, utilizing our newly proposed or redesigned algorithms and analyze the results in Chapter 5. Our performance analysis includes strong scaling studies, a detailed examination of the computational bottleneck operations, as well as spatial and temporal scaling studies. This thesis concludes with a short summary and discussion in Chapter 6.

Chapter 2

Integrated Nested Laplace Approximations

This chapter begins with an introduction of essential statistical concepts that are relevant for the INLA paradigm. Foremost, we discuss the notion of conditional independence, and how it relates to sparse matrix structures, as well as, multivariate normal distributions. Subsequently, we delve into the Bayesian hierarchical model class of latent Gaussian models, which are applicable to INLA, and explore how they rely on conditional independence properties. With this in mind, the main ideas behind the INLA methodology are introduced before looking at it from an algorithmic perspective, providing a brief overview of the main steps. This includes a presentation of how spatial and spatio-temporal models are formulated in INLA using stochastic partial differential equations (SPDEs) and the finite element method. The last part of this chapter is dedicated to laying out the objectives of this thesis, fusing the statistical concepts with state-of-the-art approaches from high-performance computing. Finally, an overview of how this is realized in terms of implementation is given, for the long-standing R-INLA package using a shared memory approach as well as the distributed–shared memory implementation INLA$_{\text{DIST}}$ targeting spatio-temporal models. The theorems and definitions provided in the next sections follow [50].

2.1 Background

2.1.1 Conditional Independence

Let $\boldsymbol{x} = (x_1, x_2, \dots, x_n)^T$ be a vector of random variables following a multivariate normal distribution with mean $\boldsymbol{\mu}$ and symmetric positive definite covariance

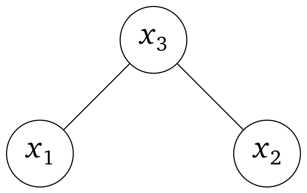


Figure 2.1: *Conditional dependence graph $\mathcal{G}(V,E)$ of $p(x_1,x_2|x_3) = p(x_1|x_3)p(x_2|x_3)$, where each node represents a random variable.*

matrix $\boldsymbol{\Sigma}$, denoted by $\boldsymbol{\Sigma} > 0$. Its probability density function is then defined as

$$p(\boldsymbol{x}) = (2\pi)^{-n/2}|\boldsymbol{\Sigma}|^{-1/2}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right), \quad \boldsymbol{x} \in \mathbb{R}^n. \tag{2.1.1}$$

The elements of $\boldsymbol{\Sigma}$ describe the (marginal) variances $\text{Var}(x_i) = \Sigma_{ii}$ and covariances $\text{Cov}(x_i,x_j) = \Sigma_{ij}$, respectively. The former quantifies the dispersion from the mean of a single variable x_i by integrating over the remaining parameters. From the covariances one can compute the correlation coefficients $\text{Corr}(x_i,x_j) = \Sigma_{ij}/\sqrt{\Sigma_{ii}\Sigma_{jj}}$ which measure the standardized linear dependence between x_i and x_j. Two random variables x_1, x_2 are called independent if and only if their probability densities satisfy $p(x_1,x_2) = p(x_1)p(x_2)$. In addition to the concept of independent random variables, we introduce the concept of conditional independence, which will be fundamental to this thesis.

Definition 2.1.1 ***(Conditional Independence)*** *Two random variables x_1, x_2 are called conditionally independent given x_3 if and only if for their conditional probability densities it holds that*

$$p(x_1,x_2|x_3) = p(x_1|x_3)p(x_2|x_3). \tag{2.1.2}$$

This can equivalently be denoted by $x_1 \perp x_2 \mid x_3$.

In a graphical model using the labelled graph $\mathcal{G}(V,E)$, with nodes V and edges E this relationship is expressed as shown in Figure 2.1, where the random variables x_1 and x_2 are connected through x_3. Thus, by fixing x_3, the other two become conditionally independent. This simple sounding concept is astonishingly powerful and allows for expressing relationships between variables. Let us for example assume that x_1 represents the water level in a lake, while x_2 expresses plant growth in a nearby area. We assume a positive dependence between larger plant growth and higher water levels and vice versa, implying that they are not independent. If x_3 now expresses the amount of rain over a period of time, it is reasonable to assume that x_3 influences, both, the water level in the lake and plant growth in the surrounding region. But once the amount of rain that has fallen is known, the lake water level and plant growth become conditionally independent, as they can be assumed to not directly influence each other. For multivariate normal distributions information on conditional independence can be found in the inverse covariance, the precision matrix as follows.

Theorem 2.1.1 *Let $\boldsymbol{x}$ be a normally distributed random vector with mean $\boldsymbol{\mu}$ and precision matrix $\mathbf{Q} > 0$. Then we have that for $i \neq j$,*

$$x_i \perp x_j \mid \boldsymbol{x}_{-ij} \quad \iff \quad Q_{ij} = 0. \tag{2.1.3}$$

The proof follows from separating the components involving x_i and x_j from the rest using Equation 2.1.1 and Definition 2.1.1 and can also be found in Section 2.2 in [50]. This implies that the nonzero pattern of the precision matrix $\mathbf{Q}$ corresponds to its conditional dependence graph.

2.1.2 Gaussian Markov Random Fields

With this in mind, we introduce the notion of Gaussian Markov Random fields (GMRFs).

Definition 2.1.2 *(GMRF) A vector of random variables $\boldsymbol{x} = (x_1, ..., x_n)^T \in \mathbb{R}^n$ is called a Gaussian Markov Random field with respect to the labeled graph $\mathcal{G} = (V, E)$ with mean $\boldsymbol{\mu}$ and precision matrix $\mathbf{Q} = \boldsymbol{\Sigma}^{-1} > 0$ if and only if its probability density has the form*

$$p(\boldsymbol{x}) = \frac{|\mathbf{Q}|^{1/2}}{(2\pi)^{n/2}} exp\,(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \mathbf{Q}(\boldsymbol{x}-\boldsymbol{\mu})) \tag{2.1.4}$$

and

$$Q_{ij} \neq 0 \quad \iff \quad \{i, j\} \in E \textit{ for all } i \neq j \tag{2.1.5}$$

Every normal distribution with a symmetric positive definite covariance matrix[1] is also a GMRF and vice versa. We are, however, mainly interested in cases where $\mathbf{Q}$ is sparse, as this is where the concept of GMRFs becomes advantageous. Sparsity in the precision matrix $\mathbf{Q}$ does not induce sparsity in the covariance matrix $\boldsymbol{\Sigma}$, as the inverse of a sparse matrix is generally dense. This also makes sense from a statistical perspective, since the modeled variables are usually not independent from each other. The term Markov in GMRF relates to the fact that all

[1] as $\boldsymbol{\Sigma} > 0 \iff \mathbf{Q} > 0$

variables (or nodes) are conditionally independent from each other if they are not neighbors, that is, if there is no edge E connecting them in the associated graph $\mathcal{G}(V,E)$.

In the next section, we will see how GMRFs are employed within the Bayesian hierarchical model formulation of latent Gaussian models.

2.1.3 Latent Gaussian Models

Latent Gaussian models (LGMs) comprise a subclass of hierarchical Bayesian additive models among which there are many of the frequently used statistical models, like regression, mixed or spatio-temporal models as well as many others, see e.g. [51, 2] for details. LGMs are characterized by three layers; the hyperparameters, the latent Gaussian field, and the observations with an associated likelihood. Starting from the latter, each observation y_i is assumed to belong to a distribution from the exponential family and is associated with the additive linear predictor η_i through a link function $g(\cdot)$. The linear predictor is defined as

$$\eta_i = \boldsymbol{\beta}^T \boldsymbol{Z}_i + \boldsymbol{u}_i(\boldsymbol{w}_i), \quad \text{for } i = 1, \dots, m. \tag{2.1.6}$$

It has fixed effects $\boldsymbol{\beta}$ with associated covariates $\boldsymbol{Z}_i$ and random effects $\boldsymbol{u}_i$ with associated covariates $\boldsymbol{w}_i$ such as temporal, spatial or spline effects, and m denotes the number of observations. We let $\boldsymbol{x} = (\boldsymbol{u}, \boldsymbol{\beta})$ which allows us to rewrite the linear predictor in matrix form such that

$$\boldsymbol{\eta} = \boldsymbol{A}\boldsymbol{x}, \tag{2.1.7}$$

where $\boldsymbol{A}$ is a sparse design or projection matrix that links the latent variables $\boldsymbol{x}$ to the linear predictor [52]. The observations are assumed to be conditionally independent given the parameters, such that

$$\boldsymbol{y} \mid \boldsymbol{\eta}, \boldsymbol{\theta} \sim \prod_{i=1}^{m} p(y_i|\eta_i, \boldsymbol{\theta}), \tag{2.1.8}$$

where $\boldsymbol{\theta}$ are the hyperparameters of the model. Typical likelihoods include Gaussian, Gamma, exponential, Weibull, Cox, binomial, Poisson and negative binomial distributions [3]. The latent parameters $\boldsymbol{x}$ form a Gaussian Markov random field (GMRF) with zero mean and sparse precision matrix $\boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{\theta})$, i.e., $\boldsymbol{x} \sim \mathcal{N}(0, \boldsymbol{Q}_{\boldsymbol{x}}^{-1}(\boldsymbol{\theta}))$, as described in the previous section. In addition, one adopts a prior $p(\boldsymbol{\theta})$ for the hyperparameters $\boldsymbol{\theta}$, which can influence both, the likelihood and the latent parameters. Thus, forming a three-stage model consisting of the hyperparameter distribution, the latent field and the likelihood. For a more exhaustive description of latent Gaussian models see e.g. [1, 3, 52], instead we conclude with a descriptive example. In Section 5.2.2, we apply INLA to model atmospheric air temperature. In this case study, the model encompasses the following components. The observations $\boldsymbol{y}$ are temperature measurements at different locations over time, where the likelihood is assumed to be normally distributed. The latent variables $\boldsymbol{x}$ consist of fixed effects $\boldsymbol{\beta}$ which encode effects such as elevation, distance to coastline or seasonal effects, and random effects $\boldsymbol{u}$ which are associated with different space-time locations, capturing the particularities the respective location. The hyperparameters $\boldsymbol{\theta}$ characterize effects like the noisiness of the data or the spatial range, which describes the distance over which spatial dependence is present. Using this model formulation, the aim is to update the prior beliefs $p(\boldsymbol{\theta})$ and $p(\boldsymbol{x}|\boldsymbol{\theta})$ using the data $\boldsymbol{y}$ to provide posterior inference estimates. How this is done will be discussed in the next section.

2.2 Methodology

2.2.1 Core Concepts

The INLA methodology leverages an analytic approximation scheme to provide estimates of the posterior marginal distributions for the hyperparameters $p(\theta_i|\boldsymbol{y})$ for all i and of the latent parameters $p(x_j|\boldsymbol{y})$ for all j. Other relevant statistics, such as credibility intervals and quantiles, can subsequently be derived. It generally holds that

$$p(\theta_i|\boldsymbol{y}) = \iint p(\boldsymbol{x},\boldsymbol{\theta}|\boldsymbol{y})\, d\boldsymbol{x}\, d\boldsymbol{\theta}_{-i} = \int p(\boldsymbol{\theta}|\boldsymbol{y})\, d\boldsymbol{\theta}_{-i}, \qquad \text{for all } i, \tag{2.2.1}$$

$$p(x_j|\boldsymbol{y}) = \iint p(\boldsymbol{x},\boldsymbol{\theta}|\boldsymbol{y})\, d\boldsymbol{x}_{-j}\, d\boldsymbol{\theta} = \int p(x_j|\boldsymbol{\theta},\boldsymbol{y})p(\boldsymbol{\theta}|\boldsymbol{y})\, d\boldsymbol{\theta}, \quad \text{for all } j. \tag{2.2.2}$$

where $\boldsymbol{\theta}_{-i}$ denotes all hyperparameters except for the i-th one, and respectively for $\boldsymbol{x}_{-j}$. The integral over $\boldsymbol{x}$ is often very high-dimensional, due to the potentially large number of latent parameters $\boldsymbol{x}$, while the integral over the underlying hyperparameters $\boldsymbol{\theta}$ is assumed to be of much lower dimension [2]. The equations above reformulate the problem such that only an integration over $\boldsymbol{\theta}$ is necessary, assuming that $p(\boldsymbol{\theta}|\boldsymbol{y})$ and $p(x_j|\boldsymbol{\theta},\boldsymbol{y})$ are known for all j. INLA approximates the former, i.e., the joint posterior of the hyperparameters, as

$$\begin{aligned} p(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{p(\boldsymbol{x},\boldsymbol{\theta}|\boldsymbol{y})}{p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})} &\propto \frac{p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})}{p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})} \\ &\approx \left.\frac{p(\boldsymbol{\theta})p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{\theta})}{p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})}\right|_{\boldsymbol{x}=\boldsymbol{x}^*(\boldsymbol{\theta})} := \tilde{p}(\boldsymbol{\theta}|\boldsymbol{y}), \end{aligned} \tag{2.2.3}$$

where $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ denotes a Laplace approximation to $p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ and will be described in detail below. The above equations can be derived from left to right as follows. The first identity is obtained through the chain rule, i.e., by rearranging the definition of conditional probability. The proportionality holds by Bayes' rule, admitting the normalizing constant $p(\boldsymbol{y})$, which is independent of $\boldsymbol{\theta}$. The distribution p_G represents a Gaussian approximation to $p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$, centered at the mode $\boldsymbol{x}^*(\boldsymbol{\theta})$ of the true conditional $p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ for a fixed vector $\boldsymbol{\theta}$. The precision matrix of p_G is defined as the curvature, i.e., the negative Hessian of p at $\boldsymbol{x}^*(\boldsymbol{\theta})$. More specifically, we assume that

$$\begin{aligned} &p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y}) = (2\pi)^{-n/2}|\boldsymbol{Q}_{x|y}(\boldsymbol{\theta})|^{1/2}\exp(-\frac{1}{2}(\boldsymbol{x}^*(\boldsymbol{\theta})-\boldsymbol{x})^T\boldsymbol{Q}_{x|y}(\boldsymbol{x}^*(\boldsymbol{\theta})-\boldsymbol{x})), \text{ with} \\ &\boldsymbol{Q}_{x|y}(\boldsymbol{\theta}) := \boldsymbol{Q}_x(\boldsymbol{\theta}) + \boldsymbol{A}^T\boldsymbol{D}\boldsymbol{A}, \quad \text{with } \boldsymbol{A} = [\tilde{\boldsymbol{A}},\boldsymbol{Z}]. \end{aligned} \tag{2.2.4}$$

Here $\boldsymbol{D}$ denotes a diagonal matrix which is derived from a second order Taylor expansion of the negative log-likelihood evaluated at the mode $\boldsymbol{x}^*(\boldsymbol{\theta})$ and $\boldsymbol{A}$ is a projection matrix, mapping the latent variables $\boldsymbol{x}$ to the linear predictor $\boldsymbol{\eta}$. Further details on this will be presented in Section 2.2.3. If the likelihood is normally distributed, $\boldsymbol{x}^*(\boldsymbol{\theta})$ can be determined directly by solving a linear system involving $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta})$, otherwise it is approximated iteratively [52].

The mode $\boldsymbol{\theta}^*$ of $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$ is not known a priori but of great interest to construct good approximations to Equation 2.2.1 as well as 2.2.2 and finding it poses an optimization problem. It can be found using a quasi-Newton method maximizing over $\boldsymbol{\theta}$, which, in each iteration, requires the evaluation of Equation 2.2.3 and Equation 2.2.4 using the current value of $\boldsymbol{\theta}$. Once $\boldsymbol{\theta}^*$ is determined, an exploration of $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$ around the mode is performed to set up an approximation scheme, using evaluation points $\{\boldsymbol{\theta}^k\}_{k=1}^K$, from which the marginal distributions $p(\theta_i|\boldsymbol{y})$ for all i, are computed, similar to a numerical integration scheme discretizing Equation 2.2.1. The posterior marginals of the latent parameters $p(x_j|\boldsymbol{y})$ for all j, are determined using Equation 2.2.2, where now the only missing components are the $p(x_j|\boldsymbol{\theta},\boldsymbol{y})$ for all j. There are a number of possibilities how to approximate them, each one presenting a different trade-off between accuracy and computational cost. The fastest but crudest strategy, known as empirical Bayes, uses only one evaluation point $\boldsymbol{\theta}^k$, namely at the mode $\boldsymbol{\theta}^*$. For more involved models this can, however, fail to capture skewness sufficiently or can generate bias. One strategy to improve the accuracy is through using additional evaluation points, where $p_G(\boldsymbol{x}|\boldsymbol{\theta}^k, \boldsymbol{y})$ is constructed for each of them. Additionally, it is possible to improve this approximation at every $\boldsymbol{\theta}^k$ using variational inference. Here, the mean of each $p_G(\boldsymbol{x}|\boldsymbol{\theta}^k, \boldsymbol{y})$ is updated by adding a correction term that is determined through solving a variational problem [53], see Section 2.2.3 for details.

The posterior marginal distributions $p(x_j|\boldsymbol{y})$ are computed using information from each evaluation point $\{\boldsymbol{\theta}^k\}_{k=1}^K$ and the respectively chosen approximations of $p(x_j|\boldsymbol{\theta}, \boldsymbol{y})$.

2.2.2 Algorithmic Overview

After introducing the main statistical concepts and ideas, we provide a more algorithmic overview of the INLA methodology that discusses the arising computational key components and indicates different opportunities for parallelism. We denote by $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$ the approximation to $p(\boldsymbol{\theta}|\boldsymbol{y})$ and respectively for other distributions. The main steps are listed below and subsequently described in further detail.

0. Construct approximation to evaluate $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$ for fixed $\boldsymbol{\theta}$.
1. Solve optimization problem given by Equation 2.2.3 for $\boldsymbol{\theta}$, to determine the mode $\boldsymbol{\theta}^*$ of $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$.
2. Compute the negative Hessian at the mode $\boldsymbol{\theta}^*$.
3. Locate evaluation points in the neighborhood of $\boldsymbol{\theta}^*$ using the Hessian. Use a numerical integration free algorithm to approximate $\tilde{p}(\theta_i|\boldsymbol{y})$.
4. Approximate densities $\tilde{p}(x_j|\boldsymbol{\theta}, \boldsymbol{y})$ for each evaluation point.
5. Combine the information from each evaluation point to obtain $\tilde{p}(x_j|\boldsymbol{y})$ similarly to Step 3.

To ease the notation, we will define the function $f(\boldsymbol{\theta})$ as

$$f(\boldsymbol{\theta}) := -\log \tilde{p}(\boldsymbol{\theta}|\boldsymbol{y}), \tag{2.2.5}$$

see Equation 2.2.3, where we consider the observations $\boldsymbol{y}$ as fixed in the current model.

Step 0: Evaluating $f(\boldsymbol{\theta})$, and thus $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$, efficiently for a fixed vector $\boldsymbol{\theta}$ is fundamental to the overall method and ubiquitous in the subsequent steps. The evaluation of f is split into its individual subcomponents, see Equation 2.2.3, that become additive through the introduced log-scale. While the likelihood and the prior of the hyperparameters are usually computationally cheap to evaluate, the prior $p(\boldsymbol{x}|\boldsymbol{\theta})$ as well as the conditional distribution $p_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$ of the latent parameters are much more costly to compute. They give rise to three kernel operations which depend on $\boldsymbol{\theta}$ and particularly stand out: The computation of the log determinant of the precision matrix of $\boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{\theta})$, as part of $p(\boldsymbol{x}|\boldsymbol{\theta})$, the log determinant of the precision matrix from the conditional distribution in the denominator, $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}(\boldsymbol{\theta})$ as part of $p_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$, as well as the computation of its conditional mean, which gives rise to the before-mentioned inner iteration to determine $\boldsymbol{x}^*$ and requires solving linear systems involving $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}(\boldsymbol{\theta})$. Thus, every evaluation of $f(\boldsymbol{\theta})$, i.e., Equation 2.2.5, comprises an inner iteration. As these precision matrices are symmetric positive definite, it is most efficient to perform a Cholesky decomposition, use the diagonal entries of the factors to compute the log determinant and then, if required, perform a forward–backward substitution, to solve the linear systems. In the spatio-temporal case, the dimension of the latent parameter vector $\boldsymbol{x}$, and thus the dimension of $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$, are directly related to the spatio-temporal discretization of the problem which grow quickly with increasing number of time steps or a finer resolution of the spatial domain.

Step 1: To find the minimum of f, or equivalently the maximum of Equation 2.2.3, a BFGS algorithm [54] is employed. As any quasi-Newton method, it requires gradient information in every iteration to determine the next search direction. We estimate the gradient ∇f using a finite difference approximation, as the analytical solution is not easily computable and would result in a loss of sparsity. Thus, every iteration l of the optimization scheme does not only require a single function evaluation of $f(\boldsymbol{\theta}^l)$ for the current iterate $\boldsymbol{\theta}^l$ but also necessitates the evaluation of many $\boldsymbol{\theta}^l \pm \boldsymbol{\epsilon}_i$ that arise from the finite difference scheme to approximate the i-th directional derivative. The exact number of required function evaluations depends on the dimension of $\boldsymbol{\theta}$, $d(\boldsymbol{\theta})$, as well as the chosen finite difference scheme. In [55] a "smart" gradient approach was proposed, improving the numeric stability of the directional derivatives.

Step 2: The negative Hessian at the mode $\boldsymbol{\theta}^*$ is approximated using a second order finite difference scheme. This requires further evaluations of Equation 2.2.1 which can be computed in parallel if resources allow.

Step 3: Approximation of the marginal posteriors of the hyperparameters $\tilde{p}(\theta_i|\boldsymbol{y})$. The space around the mode $\boldsymbol{\theta}^*$ is explored according to the chosen integration strategy. The inverse of the negative Hessian at the mode corresponds to a Gaussian approximation of the covariance matrix of $\boldsymbol{\theta}$. To correct for deviations from this Gaussian approximation, information from additional evaluation points can be employed, in a numerical integration free algorithm following [51].

Step 4: Approximation of the conditional distributions $p(x_j|\boldsymbol{\theta}, \boldsymbol{y})$. When the empirical Bayes integration strategy is used, the marginal means μ_j are directly deduced from the Gaussian approximation $p_G(\boldsymbol{x}|\boldsymbol{\theta}^*, \boldsymbol{y})$ at the mode. The marginal variances are the diagonal entries of the inverse of the precision matrix $\boldsymbol{Q}_{\boldsymbol{x}|y}(\boldsymbol{\theta}^*)$ of p_G, i.e., $\Sigma_{jj} = (\boldsymbol{Q}^{-1}_{\boldsymbol{x}|y}(\boldsymbol{\theta}^*))_{jj}$ for all j. The marginal variances of the observations $\text{Var}(\boldsymbol{y})$ can be deduced from $\boldsymbol{A}\boldsymbol{Q}^{-1}_{\boldsymbol{x}|y}(\boldsymbol{\theta}^*)\boldsymbol{A}^T$, see Appendix C for details. When an integration strategy is used that utilizes multiple evaluation points $\{\boldsymbol{\theta}^k\}_{k=1}^K$, selected inversions are performed for each $\boldsymbol{\theta}^k$, and then used to construct $p(x_j|\boldsymbol{\theta}, \boldsymbol{y})$.

Step 5: The marginal posterior distributions $p(x_j|\boldsymbol{y})$ are computed using the previously determined subcomponents $p(\boldsymbol{\theta}^k|\boldsymbol{y})$ and $p(x_j|\boldsymbol{\theta}^k, \boldsymbol{y})$ for all k.

2.2.3 Laplace Approximation and Variational Bayes Correction

INLA approximates the true conditional distribution $p(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$, arising in Equation 2.2.3, which is generally not normally distributed, using a normal distribution $p_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$. It is centered at the true mode and its precision matrix is chosen to match the curvature, i.e., negative Hessian, at the mode, as described in Section 2.2.1 and in particular Equation 2.2.4. Thus, p_G is a Laplace approximation [56, 53] which is constructed using

$$p(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) = \exp\left(-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}_x(\boldsymbol{\theta})\boldsymbol{x}\right) \cdot \prod_{i=1}^{m} p(y_i|(\boldsymbol{A}\boldsymbol{x})_i, \boldsymbol{\theta}), \tag{2.2.6}$$

where the prior of the latent parameters is by assumption Gaussian, while the likelihood is generally not. We approximate it using a second order Taylor expansion, resulting in a multivariate normal distribution which, in log-scale, equates to a quadratic functional. In log-scale, the product in the above equation becomes a sum of one-dimensional quadratics that can be rewritten in matrix format as

$$\begin{aligned} \log\left(p_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})\right) &= -\frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}_x(\boldsymbol{\theta})\boldsymbol{x} + \sum_{i=1}^{m}\left(b_i(\boldsymbol{A}\boldsymbol{x})_i - \frac{1}{2}d_i(\boldsymbol{A}\boldsymbol{x})_i^2\right) + c \\ &= -\frac{1}{2}\boldsymbol{x}^T(\boldsymbol{Q}_x(\boldsymbol{\theta}) + \boldsymbol{A}^T\boldsymbol{D}\boldsymbol{A})\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{A}\boldsymbol{x} + c, \end{aligned} \tag{2.2.7}$$

where $\boldsymbol{b} = (b_1, \ldots, b_m)^T$ and $\boldsymbol{D}$ is a diagonal matrix with $D_{ii} = d_i$ for all $i = 1, \ldots, m$ and c is a constant independent of $\boldsymbol{x}$. The entries of b_i, d_i are coefficients derived from the Taylor series expansion. Equation 2.2.7 is equivalent to Equation 2.2.4 but unnormalized, in log-scale and canonical representation. The location of the mode cannot be computed directly but is found iteratively through an inner optimization scheme[2]. We employ Newton's method to determine the mode $\boldsymbol{x}^*(\boldsymbol{\theta})$. In every iteration this requires solving a linear system involving $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta}) = \boldsymbol{Q}_x(\boldsymbol{\theta}) + \boldsymbol{A}^T\boldsymbol{D}\boldsymbol{A}$ and $\boldsymbol{b}$, where $\boldsymbol{D}$ and $\boldsymbol{b}$ depend on the current iterate of $\boldsymbol{x}$, for details see e.g. [57].

Variational Correction

For many years, using a Laplace approximation to define p_G has been the standard approach in the R-INLA package to fit the conditional distributions during the optimization process, at and around the mode of $\boldsymbol{\theta}^*$ as described in Step 4, see e.g., [1, 51, 57]. And while a Laplace approximation appears to be an intuitive choice for approximating the true conditional $p(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$, there is no reason why this is the "best" possible Gaussian approximation. Especially since it is not clear how "better" or "best" are exactly defined in this context. An alternative approach was suggested in [53] which is based on a fundamental concept from Zellner [58], where the author establishes Bayes' theorem as an optimal information processing rule. They show that utilizing information from the prior $p(\boldsymbol{x})$ and the conditional likelihood $p(\boldsymbol{y}|\boldsymbol{A}\boldsymbol{x})$, it is possible to deduce the marginal likelihood of the data $p(\boldsymbol{y})$ and the conditional posterior $p(\boldsymbol{x}|\boldsymbol{y})$ of the unknown latent parameters[3]. To use the available input information optimally, they then find the latter to be the minimizer of

$$q^*(\boldsymbol{x}) = \arg\min_{q \in P(\boldsymbol{x})} \left(-\mathbb{E}_{q(\boldsymbol{x})}[\log p(\boldsymbol{y}|\boldsymbol{A}\boldsymbol{x})] + \text{KLD}\ (q(\boldsymbol{x})||p(\boldsymbol{x})) \right) \qquad (2.2.8)$$

where $P(\boldsymbol{x})$ is the set of all probability distributions on $\boldsymbol{x}$, $\mathbb{E}_{q(\boldsymbol{x})}$ denotes the expectation and KLD the Kullback-Leibler divergence. The true posterior will be recovered in Equation 2.2.8 if it is contained in the set $P(\boldsymbol{x})$. Similar to other variational frameworks, the idea is, however, to choose $P(\boldsymbol{x})$ to be a family of simpler distributions. We restrict $P(\boldsymbol{x})$ to the set of Gaussian distributions with fixed precision matrix $\boldsymbol{Q}_{x|y}$. Therefore, one only performs a correction of the

[2] where finding the optimum of Equation 2.2.5 is the "outer" optimization problem

[3] the dependency on $\boldsymbol{\theta}$ is explicitly omitted here as well as for the remainder of this section, as $\boldsymbol{\theta}$ remains constant throughout the subsequent derivations

mean [53], simplifying Equation 2.2.8 to

$$\boldsymbol{\delta}^* = \arg\min_{\boldsymbol{\delta}} (\underbrace{\mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{x}^*+\boldsymbol{\delta},\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1})}[-\log p(\boldsymbol{y}|\boldsymbol{A}\boldsymbol{x})]}_{g_1(\boldsymbol{\delta})} + \underbrace{\text{KLD}\,(\phi(\boldsymbol{x};\boldsymbol{x}^*+\boldsymbol{\delta},\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}))||p(\boldsymbol{x}))}_{g_2(\boldsymbol{\delta})}) \tag{2.2.9}$$

where $\phi(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma})$ denotes a normal probability density function with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. The optimization over $\boldsymbol{\delta}$ in Equation 2.2.9 can be done iteratively and has a computational cost that scales with the number of latent parameters n. For ease of notation we define

$$g(\boldsymbol{\delta}) := g_1(\boldsymbol{\delta}) + g_2(\boldsymbol{\delta}). \tag{2.2.10}$$

It is assumed that the mode of the Laplace approximation $\boldsymbol{x}^*$, was previously determined using a Newton iteration as described above. We first consider $g_2(\boldsymbol{\delta})$, where the Kullback-Leibler divergence is computed between two multivariate normal distributions. As the variance remains constant and $p(\boldsymbol{x})$ has zero mean, this reduces to

$$g_2(\boldsymbol{\delta}) = \text{KLD}\,(\phi(\boldsymbol{x}|\boldsymbol{x}^*+\boldsymbol{\delta},\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}))\,||\,p(\boldsymbol{x})) = \frac{1}{2}(\boldsymbol{x}^*+\boldsymbol{\delta})^T\boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{x}^*+\boldsymbol{\delta}), \tag{2.2.11}$$

where we note that $\boldsymbol{Q}_{\boldsymbol{x}}$ is the prior precision matrix of the latent parameters $\boldsymbol{x}$. For the expected negative log-likelihood $g_1(\boldsymbol{\delta})$, we have that

$$g_1(\boldsymbol{\delta}) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{x}^*+\boldsymbol{\delta},\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1})}[-\log p(\boldsymbol{y}|\boldsymbol{A}\boldsymbol{x})] = \mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\boldsymbol{A}(\boldsymbol{x}^*+\boldsymbol{\delta}),\boldsymbol{A}\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}\boldsymbol{A}^T)}[-\log p(\boldsymbol{y}|\,\boldsymbol{\eta})]. \tag{2.2.12}$$

with $\boldsymbol{\eta} = \boldsymbol{A}\boldsymbol{x}$, see Equation 2.1.7. This reformulation is advantageous since by model assumption the data are conditionally independent given the linear predictor (see Equation 2.1.8). Thus one can rewrite the above as

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\boldsymbol{A}(\boldsymbol{x}^*+\boldsymbol{\delta}),\boldsymbol{A}\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}\boldsymbol{A}^T)}[-\log p(\boldsymbol{y}|\,\boldsymbol{\eta})] &= -\mathbb{E}_{\boldsymbol{\eta}\sim\mathcal{N}(\boldsymbol{A}(\boldsymbol{x}^*+\boldsymbol{\delta}),\boldsymbol{A}\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}\boldsymbol{A}^T)}\left[\sum_{i=1}^{m}\log p(y_i|\eta_i)\right] \\
&= \sum_{i=1}^{m}\mathbb{E}_{\eta_i}[-\log p(y_i|\eta_i)] \\
&= -\sum_{i=1}^{m}\int \log p(y_i|\eta_i)\,\phi\left(\eta_i;(\boldsymbol{A}(\boldsymbol{x}^*+\boldsymbol{\delta}))_i,(\boldsymbol{A}\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}\boldsymbol{A}^T)_{ii}\right)d\eta_i.
\end{aligned} \tag{2.2.13}$$

The final equality, thus, consists of a sum of univariate expectations, which can be evaluated efficiently as presented in the next section.

Gauss-Hermite Quadrature

A short excursion to Gauss-Hermite quadrature is presented to approximate univariate integrals of the form 2.2.13. We additionally discuss how to derive the first and second order derivatives of $g(\boldsymbol{\delta})$, which are used in a Newton iteration to determine the optimal value $\boldsymbol{\delta}^*$. Gauss-Hermite quadrature is used to approximate integrals involving Gaussian functions of the following type

$$\int_{-\infty}^{\infty} \exp(-v^2) f(v)\, dv \approx \sum_{i=1}^{K} w_k f(v_k). \tag{2.2.14}$$

where w_k are quadrature weights and v_k quadrature points of the Hermite polynomials for $k = 1, \ldots, K$. Given a function $f(v)$ with $v \sim \mathcal{N}(\mu, \sigma^2)$, the expectation of f is

$$E_v[f(v)] = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(v-\mu)}{2\sigma^2}\right) f(v)\, dv. \tag{2.2.15}$$

This does not exactly match the form Equation 2.2.14 until after applying a change of variables which results in

$$E_v[f(v)] = \int_{-\infty}^{\infty} \frac{1}{\sqrt{\pi}} \exp\left(-z^2\right) f(\sqrt{2}\sigma z + \mu)\; dz \text{ with } v = \sqrt{2}\sigma z + \mu. \tag{2.2.16}$$

Considering Equation 2.2.16 as a function of the mean μ we define[4]

$$I(\mu) := \int_{-\infty}^{+\infty} \frac{1}{\sqrt{\pi}} \exp\left(-z^2\right) f(\sqrt{2}\sigma z + \mu)\, dz. \tag{2.2.17}$$

Additionally, we derive the first and second order derivatives of $I(\mu)$ with respect to μ, as they are needed for the Newton iteration. Their full derivation is provided in Appendix A and equate to

$$\frac{d}{d\mu} I(\mu) = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{\pi}} \exp(-z^2) \frac{z}{\sigma} f(\sqrt{2}\sigma z + \mu)\, dz \tag{2.2.18}$$

$$\frac{d^2}{d\mu d\mu} I(\mu) = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{\pi}} \exp(-z^2) \frac{z^2 - 1}{\sigma^2} f(\sqrt{2}\sigma z + \mu)\, dz. \tag{2.2.19}$$

[4] as this is what Equation 2.2.9 has to be optimized for

With these results in place we return to evaluating Equation 2.2.13 by combining it with Equation 2.2.17, where subsequently we obtain after the change of variables

$$g_1(\boldsymbol{\delta}) = \int -\sum_{i=1}^{m} \log p(y_i|(\boldsymbol{A}\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}\boldsymbol{A}^T)_{ii}^{1/2} z + (\boldsymbol{A}(\boldsymbol{x}^* + \boldsymbol{\delta}))_i)\phi(z)dz. \tag{2.2.20}$$

While $g_2(\boldsymbol{\delta})$ is a quadratic functional and can thus be easily differentiated, for $g_1(\boldsymbol{\delta})$ we make use of Equation 2.2.18 and 2.2.19. Due to the conditional independence property, the partial derivatives with respect to the linear predictor can be computed separately, resulting in a diagonal matrix for the Hessian. The gradient ∇g_1 and Hessian $\nabla^2 g_1$, employing the above change of variables and using the Hermitian quadrature nodes $z_1, \dots, z_K$ and quadrature weights $w_1, \dots, w_K$, are

$$\sigma_i = (\boldsymbol{A}\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}^{-1}\boldsymbol{A}^T)_{ii}^{1/2}, \tag{2.2.21}$$

$$v_k = \sqrt{2}\sigma_i z_k + (\boldsymbol{A}(\boldsymbol{x}^* + \boldsymbol{\delta}))_i, \tag{2.2.22}$$

$$\frac{\partial}{\partial \mu_i} g_1(\boldsymbol{\mu}) = -\frac{1}{\sqrt{\pi}} \sum_{k=1}^{K} w_k \frac{z_k}{\sigma_i} \exp(-z_k^2) \log p(y_i|v_k), \tag{2.2.23}$$

$$\frac{\partial^2}{\partial \mu_i \partial \mu_i} g_1(\boldsymbol{\mu}) = -\frac{1}{\sqrt{\pi}} \sum_{k=1}^{K} w_k \frac{z_k^2 - 1}{\sigma_i^2} \exp(-z_k^2) \log p(y_i|v_k). \tag{2.2.24}$$

The sum over k is over the different quadrature points, while each i relates to a different observation y_i with $i = 1, \dots, m$. We can observe that in order to compute ∇g_1 and $\nabla^2 g_1$ we do not have to differentiate the log-likelihood, but instead it is sufficient to evaluate it for each quadrature point v_k. The first and second order derivatives of g_2 with respect to $\boldsymbol{\delta}$ can easily be computed from Equation 2.2.11. Putting all components together one finally obtains

$$\nabla g = \boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{x}^* + \boldsymbol{\delta}) + \boldsymbol{A}^T \nabla g_1 \tag{2.2.25}$$

$$\nabla^2 g = \boldsymbol{Q}_{\boldsymbol{x}} + \boldsymbol{A}^T \nabla^2 g_1 \boldsymbol{A}, \tag{2.2.26}$$

which provides a way to approximate the necessary first and second order derivatives for the Newton iteration, and we can thus determine the minimum $\boldsymbol{\delta}^*$ of Equation 2.2.9, resulting in an updated mean

$$\boldsymbol{x}_{\mathrm{VB}} = \boldsymbol{x}^* + \boldsymbol{\delta}^*. \tag{2.2.27}$$

Low-Rank Variational Correction

This can, however, be computationally expensive as we have to iteratively solve linear systems of size $n \times n$, as $\boldsymbol{\delta}^* \in \mathbb{R}^n$, where n is the number of latent variables and thus often high-dimensional. Therefore, a low-rank correction is used, where the linear systems to be solved are much smaller, while still propagating an update of the mean to all latent variables. Instead of optimizing over $\boldsymbol{\delta}^* \in \mathbb{R}^n$ directly, we consider $\boldsymbol{\delta}_{\text{LR}}$

$$\boldsymbol{x}_{\text{VB}} = \boldsymbol{x}^* + \boldsymbol{\delta}_{\text{LR}} \quad \text{with } \boldsymbol{\delta}_{\text{LR}} = \boldsymbol{Q}_I^{-1}\boldsymbol{\lambda} \tag{2.2.28}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^p$, $\boldsymbol{Q}_I^{-1} \in \mathbb{R}^{n\times p}$ with $p < n$. Here, I denotes an index set, containing a subset $I \subset \{1, 2, \ldots, n\}$ of the columns of $\boldsymbol{Q}_{x|y}^{-1}$. This allows for optimizing Equation 2.2.9 for $\boldsymbol{\lambda}$ in dimension p, as

$$\begin{aligned}
\boldsymbol{\lambda}^* &= \arg\min_{\lambda} \, (\mathbb{E}_{\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{x}^*+\boldsymbol{Q}_I^{-1}\boldsymbol{\lambda}, \boldsymbol{Q}_{x|y}^{-1})}[-\log p(\boldsymbol{y}|\boldsymbol{A}\boldsymbol{x})] \\
&\qquad\qquad + \text{KLD}(\phi(\boldsymbol{x}; \boldsymbol{x}^* + \boldsymbol{Q}_I^{-1}\boldsymbol{\lambda}, \boldsymbol{Q}_{x|y}^{-1})||p(\boldsymbol{x}))) \\
&= \arg\min_{\lambda} \, (\mathbb{E}_{\eta\sim\mathcal{N}(A(x^*+Q_I^{-1}\boldsymbol{\lambda},), AQ_{x|y}^{-1}A)}[-\log p(\boldsymbol{y}|\boldsymbol{\eta})] \\
&\qquad\qquad + \text{KLD}(\phi(\boldsymbol{x}; \boldsymbol{x}^* + \boldsymbol{Q}_I^{-1}\boldsymbol{\lambda}, \boldsymbol{Q}_{x|y}^{-1})||p(\boldsymbol{x})))
\end{aligned} \tag{2.2.29}$$

This lowers the computational cost to $O(np^2)$ instead of $O(n^3)$. Nevertheless, all latent variables $\boldsymbol{x}$ are updated jointly due to the multiplication with $\boldsymbol{Q}_I^{-1}$. In particular, we select a set of indices $I \subset \{1, 2, \ldots, n\}$ which are associated with the most influential model parameters. Those typically include the fixed effects and random effects which are associated with many observations. Numerical experiments have shown that it is sufficient to choose relatively small subsets I with $p \ll n$ to nevertheless provide corrections of high accuracy [53]. Therefore, the low-rank variational Bayes correction offers a computationally efficient approach to improve the Gaussian approximation to the conditional joint posterior $p(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$ and can be used for the approximation of the posterior marginal distributions of the latent parameters.

2.2.4 The Stochastic Partial Differential Equation Approach

Many statistical applications involve data which are linked to spatial or spatio-temporal locations. The fundamental as well as intuitive concept of geostatistics is Tobler's first law of geography, which states "everything is related to everything else, but near things are more related than distant things" [59]. To represent the connections and particularities of a spatial or spatio-temporal region, additional random effects are included in the model. Let $u(s)$ be a Gaussian random field over a domain D with $s \in D$ where $u(s)$ represents a random spatial effect at location s. By Tobler's law, this implies that two locations s_1 and s_2 which are close together, have a high probability of showing similar spatial effects $u(s_1)$ and $u(s_2)$ as opposed to two locations that are far apart from each other. Mathematically, this relationship of similarity is expressed through a covariance function. One option is the Matérn covariance function $C_M(u(s_1), u(s_2))$, which has been shown to realistically capture spatial correlation and is the most popular choice in geostatistics [49]. In his work [60], Whittle first noted that the stochastic weak solution $u(s)$ to

$$\gamma_e(\gamma_s^2 - \Delta)^{\frac{\alpha}{2}}\, u(s) = \mathcal{W}(s), \quad s \in \mathbb{R}^d \tag{2.2.30}$$

has a Matérn covariance function, assuming $\gamma_s, \alpha > 0$ are constants, Δ is the Laplace operator and $\mathcal{W}$ represents Gaussian white noise, that is, an independent random process with mean zero and an underlying standard normal distribution. Lindgren et al. show in [47] that Equation 2.2.30 can be solved for $u(s)$ to find the spatial random effects included in the model. The differential operator $L = (\gamma_s^2 - \Delta)^{\frac{\alpha}{2}}$ directly relates to the precision matrix of the spatial random effects. The finite element method, see e.g. [61], is employed to discretize Equation 2.2.30. This so-called SPDE approach has shown to be very attractive in spatial modeling, both from a theoretical and a practical point of view. It inherits consistent convergence properties and favorable computational aspects, such as sparsity in the precision matrix for $\alpha = 2n$, $n \in \mathbb{N}$, from the finite element method. It implies for example that more involved spatial domains such as irregular shaped subsets of $\mathbb{R}^d$ as well as different manifolds can easily be accommodated for as Equation 2.2.30 remains well-defined [47]. Additionally, the finite element method involves a spatial mesh discretization that can be constructed separately from the locations of the observations, granting additionally flexibility.

Rephrasing the previous paragraphs from a different perspective, this means that to formulate a model which also includes spatial phenomena that are particular to every location, additional spatial random effects are introduced. This raises the question of how to define a suitable prior for them. Instead of attempting to directly define a prior precision matrix $\boldsymbol{Q}_u$[5] (or its covariance matrix), which encapsulates the spatial correlation, one forms a precision matrix from the discretized differential operator of Equation 2.2.30. Hence, the precision matrix $\boldsymbol{Q}_u$ encodes information about the relationship between the spatial

[5]When considering the complete model, the prior precision matrix $\boldsymbol{Q}_u$ is a submatrix of $\boldsymbol{Q}_x$ which relates to the random variables $\boldsymbol{u}$.

random effects, where two spatial random effects $u(s_1)$ and $u(s_2)$ are assumed to be more similar the closer s_1 and s_2 are. This intuitively also aligns with the diffusive behavior of the Laplace operator.

For spatio-temporal models, we have a random field $u(s,t)$ containing the additional time component t, hence its covariance function $C(u(s_1,t_1),u(s_2,t_2))$ also has a dependency on time t. There are numerous approaches for a temporal extension of the Matérn covariance function, see e.g. [62, 63]. In all cases, the additional time dimension adds complexity to the model and therefore, separable covariance functions are a popular choice, i.e., $C(u(s_1,t_1),u(s_2,t_2)) = C_s(s_1,s_2)C_t(t_1,t_2)$, where C_s represents the spatial and C_t the temporal correlation which factor into a product. This simplifies the precision matrix of the corresponding model. However, separable models have a number of disadvantages, like smoothing property restrictions [63]. Additionally, if one considers defining a model through the direct definition of its dynamics, a separable model does not give rise to physically realistic dynamics [49].

A particular focus of this thesis is on non-separable space-time models, and in particular we rely on the non-separable spatio-temporal model extension as suggested by Lindgren et al. [49]. The authors present a diffusion-based family of models as an extension of the Matérn fields that additionally contains a first order derivative over time and smoothness parameters $(\alpha_t,\alpha_s,\alpha_e)$ which determine the order of the differential operators and thus the smoothness of the solution. The spatio-temporal random effects of the model are then represented as the solution to the following SPDE

$$\left(\gamma_t \frac{\partial}{\partial t} + (\gamma_s^2 - \Delta)\right) u(s,t) = \mathcal{E}_{Q,\gamma_e}(s,t), \tag{2.2.31}$$

where we assume $\alpha_t = 1$, $\alpha_s = 2$ and $\alpha_e = 1$. For the general form see Appendix B as well as [49]. The SPDE has the non-negative scale parameters $(\gamma_s,\gamma_t,\gamma_e)$, the time derivative $\frac{\partial}{\partial t}$, the Laplace operator in space Δ and $\mathcal{E}_{Q,\gamma_e}(s,t)$ which describes Gaussian noise that is uncorrelated in time but has an exponential correlation in space. The stochastic weak solution of this SPDE gives rise to a Gaussian field with diffusive behavior. Lindgren et al. [49] argue that the resulting covariance function is the most natural, as diffusion processes are fundamental to modeling spatio-temporal phenomena. When restricting Equation 2.2.31 to only its spatial component, one obtains a Matérn covariance function as in Equation 2.2.30. The differential operator of Equation 2.2.31 directly relates to the precision operator of the random effects of the spatio-temporal model components, and the parameters $(\gamma_s,\gamma_t,\gamma_e)$ are contained in the model's hyperparameters $\boldsymbol{\theta}$. The discretization of Equation 2.2.31 uses piece-wise linear basis functions in space and time. Thus, it gives rise to a continuous space-time solution while the underlying computations rely on sparse linear algebra operations. The induced sparsity pattern of the precision matrix has a block tridiagonal structure, where each diagonal block describes the discretization of a spatial domain at a different time step. The time steps are coupled through the off-diagonal blocks, similar to an autoregressive model of order one. A higher order coupling in time translates to an increased off-diagonal block bandwidth, i.e., second order gives rise to a block pentagonal structure. A detailed overview of the discretized formulation, the exact definition of all matrices and the arising sparsity patterns including an illustrative example can be found in Appendix B, as well as further notes on the interpretation of the equation.

Separable space-time models give rise to precision matrices which can be expressed as Kronecker products between the spatial and the temporal component, i.e., $\boldsymbol{Q}_u = \boldsymbol{Q}_s \otimes \boldsymbol{Q}_t$. This is the key component to the reduced computational complexity, as now these two parts can be handled separately throughout the more involved operations. For non-separable models this is not the case. Thus, all arising computational key operations require treating $\boldsymbol{Q}_u$ jointly, which is one of the reasons necessitating advanced linear algebra solution strategies. One faces the same challenges with other types of large-scale models, that are not reducible into subcomponents without the loss of information and require performant solution methods.

2.3 Objectives and Contributions: Performing Inference at Scale

In this section, I want to elucidate the greater aim of this thesis; combining in-depth knowledge of statistical concepts with expertise from the field of high-performance computing to allow for larger-scale, more realistic Bayesian modeling at shorter run times. The previous part of this chapter delved into the particular Bayesian inference method of interest in this thesis, the methodology of integrated nested Laplace approximations. We also introduced its applicable model class, discussed the underlying assumptions and started analyzing the required linear algebra operations. With this in mind, we further explore the computational aspects of the INLA methodology. First by laying out what challenges arise, and then how these can be addressed by combining domain specific knowledge with state-of-art high-performance computing approaches. This objective is pursued along two different main routes. The first one is to identify, isolate, and subsequently improve the performance of the computational bottleneck operations. This is achieved through a variety of strategies, one being the development of algorithms tailored to the specific problem structures. Further, through optimized implementations of the existing algorithms incorporating advances from the field of linear algebra. And finally, through the use of new hardware architectures by putting forward solution strategies that leverage the strengths of GPU accelerators. The second main route is the introduction of massive parallelism. We put forward multi-layer parallel schemes that leverage the power of modern multi-core as well as multi-node architectures. This introduction of parallelism allows us to use the often plentiful availability of resources on modern computers. To guarantee a high degree of efficiency this requires, however, that operations are independent, separable or only necessitate limited communication which has to be taken under consideration during the methodological development and algorithmic design. The majority of this work has been published in [64] and [65]. The former presents the advancements realized within the R-INLA package, while the latter summarizes the work related to large-scale spatio-temporal modeling with $\text{INLA}_{\text{DIST}}$.

2.3.1 Performance Considerations

The computational complexity of performing inference with INLA is tightly linked to the dimension of the latent parameter space and the number of hyperparameters of the associated model but independent of the number of observations[6]. We consider Equation 2.2.3 which is to be evaluated for a fixed parameter configuration of $\boldsymbol{\theta}$. The prior of the hyperparameters $p(\boldsymbol{\theta})$ and the conditional likelihood $p(\boldsymbol{y}|\boldsymbol{\theta},\boldsymbol{x})$ are typically computationally cheap to evaluate, whereas the evaluation of $p(\boldsymbol{x}|\boldsymbol{\theta})$ and $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ pose the challenge of computing the log-determinants and solving linear systems. Thus, referring to large-scale applications implies that the model has a large number of latent parameters or hyperparameters or both. The required computational bottleneck operations within INLA involve the symmetric positive-definite precision matrices $\boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{\theta})$ and $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta})$, whose dimensions are induced by $d(\boldsymbol{x}(\boldsymbol{\theta}))$, and comprise

(1) *Cholesky factorization*

(2) *Forward-backward substitution*

[6]in terms of computational core operations, not runtime spent on handling the potentially large associated data structures, etc.

(3) *Selected matrix inversion*

The dimension of the hyperparameter space, $d(\boldsymbol{\theta})$, on the other hand determines the number of required evaluations of $f(\boldsymbol{\theta})$ during each BFGS-iteration as the number of partial derivatives increases with $d(\boldsymbol{\theta})$ and therefore the number of finite difference evaluations. Moreover, a larger number of hyperparameters requires more evaluation points around the mode $\boldsymbol{\theta}^*$, also inducing more selected matrix inversions. Thus, the importance of optimizing the computational bottleneck operations (1)–(3) is linked to the dimension of the latent parameter space, whereas the necessity or opportunity for parallelism (outside of the linear solvers) is more closely tied to the dimension of the hyperparameters $\boldsymbol{\theta}$.

Matrix Sparsity Patterns and Sparse Solvers

For larger models naive implementations such dense Cholesky factorization or full matrix inversions quickly become prohibitive in terms of computational cost. Thus, it is necessary to exploit the sparsity of the underlying GMRF. We can, however, not only utilize general sparse linear algebra operations but make use of the fact that many of the arising sparsity patterns are recurrent. Foremost, this includes the precision matrices $\boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{\theta})$ and $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}(\boldsymbol{\theta})$. They depend on the hyperparameters $\boldsymbol{\theta}$ and latent parameters $\boldsymbol{x}$, which iteratively change throughout the algorithm. In a typical use case, hundreds if not a couple of thousand different parameter configurations arise, inducing new numerical values of both matrices, while their respective sparsity patterns remain constant. The majority of necessary matrix factorizations arises within the optimization problem, in the evaluation of $f(\boldsymbol{\theta})$, as well as its gradient. This allows for various forms of optimization. Sparse matrices are typically stored in so-called compressed sparse row (CSR) or compressed sparse column (CSC) format, consisting only of the non-zero values and two arrays from which the appropriate indices can be deduced. Thus, changes in $\boldsymbol{Q}_{\boldsymbol{x}}(\boldsymbol{\theta})$ and $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}(\boldsymbol{\theta})$ only require updating the numerical values, while the index arrays remain untouched. For small matrices the time improvement might be negligible, but for large-scale problems this becomes relevant. Recurring sparsity patterns in the precision matrices also imply that the sparsity pattern of the associated Cholesky factors remain the same, and are therefore known a priori (or at least after the very first factorization). When employing an entirely sparse approach it is therefore advantageous to first perform a separate symbolic factorization to pre-compute the non-zero patterns where the matrices are reordered to reduce fill-in and allow for more parallelism while computing the numerical values of the Cholesky factor. The symbolic factorization only needs to be computed once. A detailed discussion of how this is achieved and the advantages it entails are presented in Section 3.1.1. To compute the marginal variances of the latent parameters as well as the observations, see Step 4, we require entries from the covariance matrix, i.e., the inverse of the precision matrix $\boldsymbol{Q}_{x|y}^{-1}(\boldsymbol{\theta})$. The inverse of a sparse matrix is generally dense, and therefore inversions of large sparse matrices are computationally expensive, if not infeasible, operations. In the case at hand, however, we only require particular elements of the full inverse. More precisely, those related to the marginal variances of the latent variables and the observations, see also Appendix C. There are specialized strategies to perform a partial or selected inversion that do not require the computation of all elements [66, 67]. An efficient and versatile selected inversion routine for sparse matrices which is based on Cholesky decomposition is known as Takahashi inversion [68] which we make use of and present in Section 3.1.2, thus being able to dramatically reduce memory requirements as well as computation time. Many large-scale models include data with a spatial or spatio-temporal component, as the associated random effects quickly become high-dimensional. The class of diffusion-based spatio-temporal models discussed in Section 2.2.4 is particularly high-dimensional due to its non-separable precision matrix structure. It gives, however, rise to precision matrices with a regular block n-diagonal arrowhead sparsity pattern, where each diagonal block corresponds to a discretization of the spatial domain at a different time step. Each of the blocks themselves are again sparse, as they arise from a spatial finite element discretization. Our objective is to leverage this additional knowledge of the sparsity structure at hand and derive tailored solution strategies that perform the computational bottleneck operations targeting exactly this sparsity pattern. We limit ourselves to this particular spatio-temporal SPDE model, however, other large-scale applications also exhibit block n-diagonal arrowhead sparsity patterns. Thus, this work can hopefully also serve as a prototype for more general large-scale applications in the future. The details are presented in Section 3.2, where we develop block dense algorithms for the required Cholesky decompositions as well as selected matrix inversion which exhibit linear scaling in time.

Parallelism

In addition to improving the performance of the individual computational bottleneck operations, it is crucial to execute them simultaneously whenever possible, as the scalability of an algorithm is closely tied to its ability to parallelize efficiently. All modern computer architectures heavily rely on parallelism to support increasingly many as well as increasingly complex operations. This is also a key component for amplifying the scalability of INLA. The algorithmic overview in Section 2.2.2 reveals that many computationally expensive operations within the INLA methodology can be executed concurrently. Most notably the function evaluations of $f(\boldsymbol{\theta})$ to approximate the gradient in Step 1, the function evaluations for the Hessian in Step 2 as well as the evaluation of the integration points and the partial matrix inversions in Step 3 and 4, respectively. The number of parallel function evaluations depends on the dimension of the hyperparameters $\boldsymbol{\theta}$ as well as the chosen approximation scheme around the mode $\boldsymbol{\theta}^*$. Thus, we first compute the different necessary parameter configurations of $\boldsymbol{\theta}$ and then perform the associated function evaluations in parallel. In terms of implementation, this is supported among two routes, a shared memory implementation within R-INLA and a shared-distributed memory implementation within $\text{INLA}_{\text{DIST}}$, greatly decreasing overall run times as presented in Chapter 5.

Implementation and Compute Architectures

GPU accelerators are becoming increasingly important across all scientific computing domains. They have seen a rapid rise in performance and general adoption over the last decades. This can also be seen by the fact that 9 out of the top 10 supercomputers[7] have GPUs as integral compute units who deliver the majority of the arithmetic throughput [69]. Most of the existing Bayesian libraries do not include GPU support yet, with PyMC [31] and Stan [32] being an exception for some operations. GPU support in itself does not, however, guarantee accelerated computations. GPUs especially excel at large-scale parallel independent computing tasks involving regular memory access patterns like dense matrix-matrix multiplication. On the other hand, they may exhibit inferior performance to CPUs when it comes to more involved tasks that are sequential or have limited parallelism, simple or small-scale computations, and memory-intensive operations. To leverage the respective advantages of the different processing units, it is necessary during algorithm design to keep their particular properties in mind. For the work done in this thesis, we have thus concluded it to be beneficial employing entirely sparse approaches on CPU and a block dense approach on GPU for the computational key operations. Both will be discussed in detail in Chapter 3 with performance comparisons in Chapter 5.

Large-scale clusters using GPU-accelerators are typically organized in partitions with GPGPU (General-Purpose Graphics Purposing Units) nodes where a many-core central processing unit is attached to multiple GPUs, that are often additionally interconnected. The number of CPU cores, memory domains, as

[7]The only exception being Japan's supercomputer Fugaku.

well as attached GPUs varies across clusters. A typical setup would be around 64 CPU multi-core processor, 4 or 8 NUMA domains and 4 or 8 GPUs. Architecture specific knowledge is therefore required for an optimal arrangement. As the specifications vary across clusters, it is non-trivial to automatically identify setups that connect the different components most efficiently.

The efforts in this thesis regarding software libraries have been two-fold, firstly by adding scalability to the R-INLA package and secondly, in the form of the standalone $\text{INLA}_{\text{DIST}}$ implementation targeting multi-node architectures, optionally utilizing GPU-accelerators for large-scale spatial and spatio-temporal models. More detailed overviews of the work conducted on both libraries are presented in the following two sections.

2.3.2 R-INLA – A shared Memory Implementation

A user-friendly implementation of INLA is available in the form of an R-package under the same name, referred to as R-INLA [70]. Since its inception, there have been many papers exploring advancements of theoretical concepts of the INLA methodology, also leading to a constant evolution of its implementation. Their collection forms an impressive repertoire for fast, versatile and reliable approximate Bayesian inference, see e.g. [47, 51], and include a wide variety of applications, see e.g. [10, 71, 72, 5, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85]. Algorithmic concepts and improvements concerning performance matters of the continuously growing software library have been discussed much less, even though they are a key component to INLA's success. We aim at helping to close this gap by putting forward a much more performant implementation of R-INLA making use of multithreaded parallelism and the state-of-the-art sparse linear solver PARDISO [86]. A two-layer parallel scheme is introduced which, on the upper or first layer, primarily parallelizes the independent function evaluations as described in the previous sections. The original implementation of R-INLA employs the sparse linear algebra library TAUCS [87] to perform the required numerically intensive core operations. It is a well-designed and efficient library which operates, however, sequentially and whose support was discontinued in the early 2000s. Additionally, it does not include a partial inversion routine which was therefore implemented by the authors of R-INLA, see [88, 1]. While this was done with much thought, effort and consideration, providing reliable results, improved parallel implementations have emerged since [89, 90]. The plan of integrating PARDISO into R-INLA was first described by Nierkerk et al. in [57]. The authors discuss the need for the usage of faster numerical solvers within R-INLA to support evolving statistical models of higher complexity and continuously growing availability of data. PARDISO offers a great fit as it provides multithreaded implementations for all of INLA's required kernel operations. Hence, the second layer of parallelism is introduced by PARDISO as well as other parallelized linear algebra operations like matrix-matrix or matrix-vector products. We showcase the improved performance of R-INLA on a complex joint survival model containing 50 hyperparameters [91] and a brain activation model using fMRI data [92].

2.3.3 $\text{INLA}_{\text{DIST}}$ – A distributed Memory Implementation

In addition to the advancements of the R-INLA package, we put forward an independent novel hybrid distributed shared memory implementation, referred to as $\text{INLA}_{\text{DIST}}$, which is tailored to the previously introduced INLA-SPDE approach for spatio-temporal modeling [65]. $\text{INLA}_{\text{DIST}}$ is written in C++, makes use of the Eigen library [93] and the BFGS template library provided by [94]. It comprises a three-layer parallel scheme employing distributed and shared memory parallelism, which will be described in Chapter 4. To handle the computational key operations, we integrate two alternative numerical solvers in $\text{INLA}_{\text{DIST}}$. One option is the sparse solver PARDISO. This combination of $\text{INLA}_{\text{DIST}}$ will be referred to as $\text{INLA}_{\text{PARDISO}}$. Secondly, we put forward a GPU-accelerated novel block tridiagonal arrowhead (BTA) solver which will be presented in Section 3.2, referred to as INLA_{BTA}. The required kernel operations of the BTA solver are performed by standard state-of-the-art linear algebra libraries, namely by MAGMA [95], cuBLAS [96] and LAPACK [97]. Depending on the compute infrastructure and the problem size, the complete Cholesky factor can exceed the GPU memory. To nevertheless allow for almost arbitrarily large parameter spaces, only the currently required submatrix blocks are iteratively copied to GPU memory to compute the Cholesky factorization. The log determinants are then directly computed on GPU. When necessary, the complete Cholesky factor is stored in main memory, in which case the forward-backward substitution is then performed on CPU. To compute the selected block inversion, the necessary submatrices of the Cholesky factor are recursively copied back to GPU memory throughout the computation as needed. We present performance studies for $\text{INLA}_{\text{DIST}}$ on a family of synthetic datasets and apply it to an atmospheric temperature model [98] over the US main continental area throughout the course of 1 year using more than 1 million latent parameters and 2.5 million observations, see Section 5.2.

Chapter 3

High-Performance Operations in INLA

In this chapter we explore how to handle the linear algebra operations that arise within the framework of INLA. The first part presents an entirely sparse approach, and introduces the most important underlying concepts emerging in this context. This is motivated by the fact that the prevailing precision matrices are part of sparse GMRFs, and it is, therefore, only natural to rely on sparse solution strategies to perform the computational kernel operations. While sparse Cholesky decomposition is a widely supported functionality that is part of many sparse linear algebra libraries (and subsequently solving linear systems), efficient selected inversion routines are not frequently supported. One possibility is to employ the state-of-the-art sparse direct solver PARDISO [86]. It performs efficient Cholesky factorizations through matrix permutations, which allow for internal thread parallelization and lead to drastically reduced fill-in. It additionally offers a selected matrix inversion routine, building on the work of Takahashi [68]. We integrate PARDISO into the R-INLA package and $\text{INLA}_{\text{DIST}}$ as a CPU-based sparse direct solver.

The second part of this chapter puts forward a blocked approach for spatio-temporal models, where the prevailing precision matrices are divided into subblocks that are either completely zero or assumed to be dense. This is possible due to the particular sparsity pattern induced by the spatio-temporal nature of the problem. Tailored block algorithms for Cholesky decomposition, solving linear systems and selected matrix inversion are developed to operate on the arising block-tridiagonal arrowhead structure. This allows for employing efficient GPU-accelerated dense matrix kernels for the non-zero subblocks while completely omitting the zero subblocks of the system. This approach is integrated into $\text{INLA}_{\text{DIST}}$, and referred to as block tridiagonal arrowhead (BTA) solver as a GPU-based block dense direct solver.

3.1 A General Sparse Method

A sparse matrix is simply defined as a matrix, where the majority of its elements are equal to zero. For sparse matrix algebra, it is customary to only save the non-zero entries of the matrix in so-called compressed matrix formats, that store the non-zero values and their corresponding position instead of large quantities of zero entries. This allows to massively reduce the required memory and computations but hence, also demands for solvers that are especially tailored to sparse problems. These arise, however, very commonly and are therefore extensively researched. For introductory purposes see e.g. [99, 100].

During the first stage of the INLA algorithm, see Section 2.2.2, the log determinant of different precision matrices $\boldsymbol{Q}$ (to obtain the corresponding normalizing constant) and the solution to linear systems of equations of the form $\boldsymbol{Qx} = \boldsymbol{b}$ (to obtain the mode of the associated normal distribution) are required. Since the arising precision matrices are symmetric positive-definite, it is most efficient to perform a Cholesky decomposition, where the original matrix $\boldsymbol{Q}$ factors into $\boldsymbol{Q} = \boldsymbol{LL}^T$, with $\boldsymbol{L}$ being a lower-triangular matrix [101]. The log determinant of $\boldsymbol{Q}$ can easily be computed from the diagonal entries of $\boldsymbol{L}$ as $\sum_{i=1}^{n} 2\log(L_{ii})$ and the system $\boldsymbol{Qx} = \boldsymbol{b}$ can quickly be solved for $\boldsymbol{x}$ using forward-backward substitution. That means first solving $\boldsymbol{Ly} = \boldsymbol{b}$ for $\boldsymbol{y}$ and then $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{y}$ for $\boldsymbol{x}$. These operations can be executed efficiently due to the lower-triangular structure of $\boldsymbol{L}$.

3.1.1 Matrix Reordering

Additionally, it is in most cases advantageous to apply a symmetric permutation to the matrix $\boldsymbol{Q}$ (and hence $\boldsymbol{b}$, respectively) before computing the Cholesky decomposition. Even if the matrix $\boldsymbol{Q}$ is very sparse, $\boldsymbol{L}$ can have a large fill-in, meaning that there are many entries which are non-zero in $\boldsymbol{L}$ but equal to zero in the lower-triangular part of $\boldsymbol{Q}$. In the elimination process that is used to compute $\boldsymbol{L}$, similar to classical Gaussian elimination, one can see how these non-zeros arise [99]. As an illustrative example, we consider a symmetric positive-definite arrowhead matrix $\boldsymbol{Q}_1$ whose sparsity pattern can be seen in Figure 3.1. Its corresponding Cholesky factor $\boldsymbol{L}_1$ is computed recursively, starting from the first diagonal entry. To compute the off-diagonal entry $(\boldsymbol{L}_1)_{32}$, we use that $(\boldsymbol{L}_1)_{32} = ((\boldsymbol{Q}_1)_{32} - (\boldsymbol{L}_1)_{31}(\boldsymbol{L}_1)_{21})/(\boldsymbol{L}_1)_{22}$. Hence, even if $(\boldsymbol{Q}_1)_{32} = 0$, $(\boldsymbol{L}_1)_{32}$ is not equal to zero unless $(\boldsymbol{L}_1)_{31}$ or $(\boldsymbol{L}_1)_{21}$ are. More generally, one can observe a dependency structure on previous columns of the i-th and j-th row. By applying a symmetric permutation to $\boldsymbol{Q}_1$, we can obtain $\boldsymbol{Q}_2$. If we now compute $\boldsymbol{L}_2$, we can see that $(\boldsymbol{L}_2)_{32} = 0$, since $(\boldsymbol{Q}_2)_{32}, (\boldsymbol{L}_2)_{21}$ and $(\boldsymbol{L}_2)_{32}$ are all equal to zero.

x	x	x	x
x	x		
x		x	
x			x

(a) $\boldsymbol{Q}_1$

x			
x	x		
x	x	x	
x	x	x	x

(b) $\boldsymbol{L}_1$

x			x
	x		x
		x	x
x	x	x	x

(c) $\boldsymbol{Q}_2$

x			
	x		
		x	
x	x	x	x

(d) $\boldsymbol{L}_2$

Figure 3.1: *Sparsity patterns of the four-by-four symmetric positive-definite arrowhead matrices $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$, where each x stands for a non-zero entry and $\boldsymbol{Q}_2$ is a symmetric permutation of $\boldsymbol{Q}_1$. The sparsity patterns of their Cholesky factors are represented by $\boldsymbol{L}_1$ and $\boldsymbol{L}_2$, respectively. The matrix $\boldsymbol{L}_1$ is dense, i.e., exhibits a large fill-in while $\boldsymbol{L}_2$ preserves the sparsity pattern of $\boldsymbol{Q}_2$.*

In general, fill-in can often be drastically reduced by finding a suitable matrix reordering of $\boldsymbol{Q}$, which then lowers the overall memory and computation requirements of $\boldsymbol{L}$.

For each of the different arising precision matrices in INLA, the sparsity pattern does not change during the optimization phase but only the respective numerical values of the non-zero entries. Hence, it is sufficient to only compute suitable reorderings once throughout the entire algorithm. Finding a favorable permutation is, however, a challenging task [102] and has hence been, an active area of research [103, 104].

A simple greedy strategy is the minimum degree ordering, where columns are successively permuted to minimize the number of non-zero off-diagonal entries of the current pivot element [105]. Another popular class of reordering strategies called nested dissection employs graph partitioning techniques [106, 104]. A symmetric matrix can be associated with an undirected graph, where each diagonal matrix entry represents a node and each nonzero off-diagonal entry an edge between the corresponding nodes. The graph is partitioned into two roughly equal-sized independent subgraphs $\mathcal{G}_A, \mathcal{G}_B$ which are only connected through a separating set $\mathcal{G}_S$. The partition is chosen such that the size of the separating set is minimized. For the corresponding matrix, this means that the associated submatrices $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$ can be written as block matrices only connected through $\boldsymbol{Q}_S$. Hence, all fill-in that can occur is within $\boldsymbol{Q}_A, \boldsymbol{Q}_B$ and $\boldsymbol{Q}_S$, see Figure 3.2.

For large graphs, the computational cost is often too high to compute an optimal separating set. Therefore, multilevel strategies are used to recursively coarsen a large graph by merging connected vertices until it is reduced to a tractable size. For the coarse graph, an edge cut can be defined which is recursively refined while being propagated back to the original large graph. From the final edge cut, we can deduce a separating set $\mathcal{G}_S$ [107]. The same strategy of

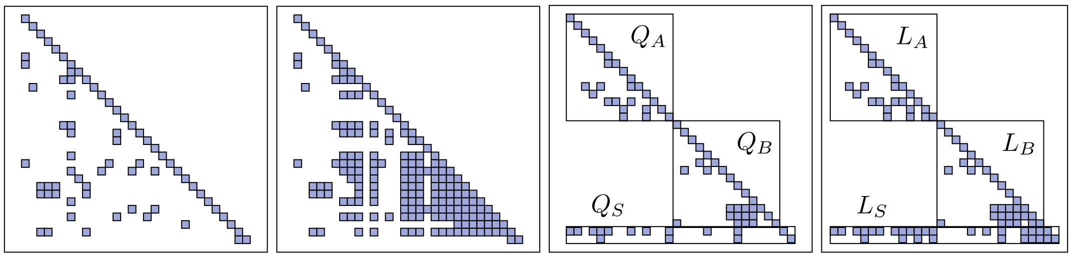


Figure 3.2: *From left to right: (a) Lower triangular part of the sparsity pattern of a random symmetric positive definite matrix* $\boldsymbol{Q}$*, with 70 non-zero entries. (b) Cholesky decomposition* $\boldsymbol{L}$ *of* $\boldsymbol{Q}$*, where L has 180 non-zero entries. (c) Permuted matrix* $\boldsymbol{Q}_{perm}$ *using (nested) dissection. (d) Cholesky decomposition* $\boldsymbol{L}_{perm}$ *of* $\boldsymbol{Q}_{perm}$*, where* $\boldsymbol{L}_{perm}$ *has only 88 non-zero entries. We can observe that* $\boldsymbol{Q}$ *was permuted such that it is separated in two independent submatrices* $\boldsymbol{Q}_A$ *and* $\boldsymbol{Q}_B$*, which are only connected through entries that are now placed in the two rows, which we refer to as* $\boldsymbol{Q}_S$*. This way there is no fill-in between* $\boldsymbol{Q}_A$ *and* $\boldsymbol{Q}_B$ *but only within the before-mentioned submatrices. This reduces the number of non-zeros in the Cholesky factorization of* $\boldsymbol{Q}_{perm}$*. The strategy can be applied recursively to each of the components* $\boldsymbol{Q}_A$ *and* $\boldsymbol{Q}_B$*.*

finding roughly balanced minimal separating sets can be independently reapplied to $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$, leading to the nested dissection approach. A popular state-of-the-art library that computes such heuristic matrix reorderings is called METIS [107] and is used by both, TAUCS as well as PARDISO. While the previously employed TAUCS library is limited to using fill-in reducing permutations, PARDISO can additionally perform much of the Cholesky factorization in parallel, making use of the independent submatrices arising from the nested dissection reordering. This will be explained in detail in Section 4.1.3.

3.1.2 Selected Inversion

In Step 4, an estimate for the variances of the posterior marginals of the latent parameters, i.e., $p(x_i|\boldsymbol{y})$ for all i, is obtained. To do so, INLA uses information from the Gaussian approximations $p_G(\boldsymbol{x}|\boldsymbol{\theta}^k, \boldsymbol{y})$ at the various evaluation points $\{\boldsymbol{\theta}^k\}_{k=1}^K$. All information about the marginal variances only exists, however, in the form of the precision matrices. As the variances Σ_{ii} correspond to $(\boldsymbol{Q}^{-1})_{ii}$, they can only be obtained through matrix inversions. More generally, the entries Σ_{ij} of interest are contained within the set of entries, where $Q_{ij} \neq 0$, see Appendix C for details. If the full precision matrices had to be inverted, this would become very time and memory consuming if not infeasible for high-dimensional latent parameter spaces as matrix inversions have a complexity of $O(n^3)$ for a matrix of size $n \times n$. Fortunately, there is an alternative method for computing the marginal variances that is much more efficient. There are two different approaches to derive this recursive strategy. One is using the conditional independence properties of GMRFs and their associated graphical structure, and was first described by Rue and Martino in [88]. The other one is based on a particular way of writing matrix identities without a further interpretation but describing the same recursion and was developed by Takahashi [68] almost 50 years ago. We will begin by considering the former. The solution $\boldsymbol{x}$ to the problem $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{z}$ where $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ is a sample from a GMRF with zero mean and precision matrix $\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{L}^T$ [50]. Since $\boldsymbol{L}^T$ is an upper triangular matrix, we can use a backward solve and recursively compute

$$
\begin{aligned}
x_n &= \frac{z_n}{L_{nn}} \\
x_i &= \frac{z_i}{L_{ii}} - \frac{1}{L_{ii}} \sum_{\substack{k>i \\ L_{ki}\neq 0}}^{n} L_{ki} x_k
\end{aligned}
\tag{3.1.1}
$$

for $i = n-1, \ldots, 1$. We exclude all terms L_{ki} from the summation directly that equate to zero. This way, it becomes clear that the more zeros we have in $\boldsymbol{L}$, the less computations are necessary. If we use the known expected value and variance of $\boldsymbol{z}$ as well as Equation 3.1.1, we can deduce the covariance matrix $\boldsymbol{\Sigma}$ of $\boldsymbol{x}$ from first principles, obtaining

$$
\Sigma_{ij} = \frac{\delta_{ij}}{L_{ii}^2} - \frac{1}{L_{ii}} \sum_{\substack{k>i \\ L_{ki}\neq 0}}^{n} L_{ki} \Sigma_{kj}, \tag{3.1.2}
$$

where δ_{ij} is one if $i = j$ and zero otherwise. The entries can only be computed recursively starting at $i = n$, traversing the matrix from the bottom right to the top left.

Equivalently, we can derive Equation 3.1.2 without using statistical properties. Instead, we consider the slightly altered decomposition $\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{L}^T = \boldsymbol{V}\boldsymbol{D}\boldsymbol{V}^T$, i.e. $\boldsymbol{L} = \boldsymbol{V}\boldsymbol{D}^{1/2}$, where $\boldsymbol{D}$ is a diagonal matrix and $\boldsymbol{V}$ a lower triangular matrix with ones on the diagonal. The following matrix identity was proposed by Takahashi in [68],

$$
\boldsymbol{\Sigma} = \boldsymbol{D}^{-1}\boldsymbol{V}^{-1} + (\boldsymbol{I} - \boldsymbol{V}^T)\boldsymbol{\Sigma}. \tag{3.1.3}
$$

As $\boldsymbol{\Sigma}$ is a symmetric matrix, it is enough to compute its upper triangular part. The term $\boldsymbol{D}^{-1}\boldsymbol{V}^{-1}$ is lower triangular with $(\boldsymbol{D}^{-1}\boldsymbol{V}^{-1})_{ii} = (\boldsymbol{D}^{-1})_{ii}$, since $\boldsymbol{V}$ has a unit diagonal. Writing out Equation 3.1.3 as sums we obtain

$$\Sigma_{ij} = \frac{\delta_{ij}}{D_{ii}} - \sum_{\substack{k>i \\ V_{ki} \neq 0}} V_{ki} \Sigma_{kj}, \quad \text{for } i \leq j \tag{3.1.4}$$

which we can compute recursively starting from Σ_{nn} and where again, δ_{ij} is equal to one for $i = j$ and zero otherwise. Equation 3.1.2 and 3.1.4 are equal since $\boldsymbol{L} = \boldsymbol{V}\boldsymbol{D}^{1/2}$. It is possible to use these recursions to compute all entries of $\boldsymbol{\Sigma}$, however, in this case they do not give us a computational advantage over traditional inversion algorithms. In both cases, we have a complexity of $O(n^3)$. If we are instead only interested in particular entries of $\boldsymbol{\Sigma}$, e.g. only the diagonal, and if additionally $\boldsymbol{L}$ (and then likewise $\boldsymbol{V}$) are sparse, a tremendous amount of computational cost can be saved, as only the entries Σ_{ij} for which L_{ij} is non-zero need to be computed. Hence, the less non-zeros we have in the factor $\boldsymbol{L}$, the smaller is the computational demand. This highlights the importance of finding suitable permutations, as described in the previous section.

PARDISO and INLA's previous selected inversion routine both employ such a partial inversion strategy. While the latter is sequential, PARDISO is using parallelized computations for a shorter time to solution, exploiting the particular matrix structure given through the permutation, see Section 4.1.3 for details. The marginal variances Σ_{ii} that we obtain are then re-permuted to correspond to the original ordering of the parameters $\boldsymbol{x}$. Afterward, they can be used as described in Step 5 to obtain estimates of the marginal posterior distributions of the latent parameters.

3.2 A Blocked Dense Method

In contrast to the previously introduced general sparse approach this section puts forward a customized blocked dense approach tailored to the spatio-temporal models at hand. We leverage the particular block tridiagonal arrowhead sparsity pattern of the spatio-temporal random field and present customized algorithms for the arising computational bottleneck operations. Finally, an analysis of their asymptotic computational complexity is presented.

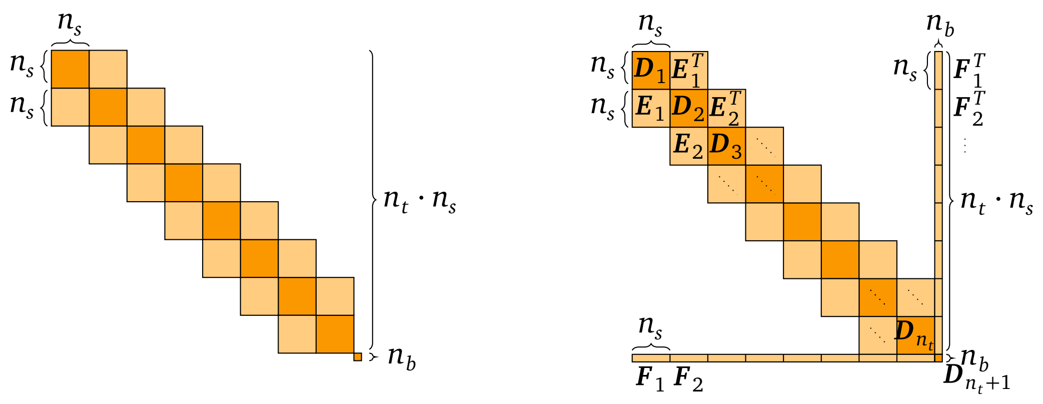
3.2.1 Sparsity Pattern of Spatio-Temporal Precision Matrices

For general latent Gaussian models the sparsity patterns of the associated precision matrices is unknown a priori. Conversely, in the spatio-temporal case, following the approach described in Section 2.2.4, a lot of structural information is known beforehand. We focus on the recurring sparsity patterns that arise in the precision matrices $\boldsymbol{Q}_x(\boldsymbol{\theta})$ as part of $p(\boldsymbol{x}|\boldsymbol{\theta})$ and $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta})$ as part of $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ which are induced by the finite element discretization of Equation 2.2.31. For brevity, we will implicitly assume the dependence of $\boldsymbol{Q}_x, \boldsymbol{Q}_{x|y}$ on $\boldsymbol{\theta}$ for from now on and omit it in the notation. We order the latent parameters $\boldsymbol{x}$ to first contain the random effects $\boldsymbol{u}$ and then the fixed effects $\boldsymbol{\beta}$, where $\boldsymbol{u}$ is internally structured to first accommodate all spatial grid points associated with the first time step, secondly all spatial grid points associated with the second time step, etc. This gives rise to a tridiagonal block structure in the precision matrices that relates to the discretized SPDE, where the off-diagonal blocks represent the coupling between two subsequent time steps. Each diagonal block refers to one instance of the discretized spatial domain. Further details on the matrix sparsity pattern can be found in Appendix B. The prior precision matrix of the latent parameters $\boldsymbol{\beta}$ is included in the last block of $\boldsymbol{Q}_x$, see Figure 3.3a. The precision matrix $\boldsymbol{Q}_{x|y}$ additionally contains covariate information that arises from conditioning on the data $\boldsymbol{y}$, as defined in Equation 2.2.4. The sparsity pattern of the prior is preserved in the spatio-temporal part, but not in rows and columns associated to the fixed effects $\boldsymbol{\beta}$, which generally become dense, see Figure 3.3b. The individual entries of the arising precision matrices are dependent on $\boldsymbol{\theta}$ whereas the sparsity pattern remains constant for a given set of covariates and spatio-temporal discretization, throughout the algorithm.

3.2.2 Block Factorization and Inversion

In this section, we develop blocked solution strategies using this structural matrix information. To compute the log determinants of $\boldsymbol{Q}_{x|y}$ and $\boldsymbol{Q}_x$, respectively, we perform block-wise Cholesky decompositions. Algorithm 1 describes the block factorization. A schematic overview of the sparsity pattern of the final Cholesky factor $\boldsymbol{L}_{x|y}$ is given in Figure 3.4. For the prior precision matrix $\boldsymbol{Q}_x$, the algorithm simplifies to factorizing a block tridiagonal matrix as the dense rows and columns referred to as $\boldsymbol{F}_i$ are equal to zero, see Figure 3.3a. Therefore, all terms in Algorithm 1 which relate to $\boldsymbol{F}_i$ equate to zero. The blocked approach bears the advantage that each step of the iterative matrix factorization only requires the submatrices associated with the current time-step t as well as the subsequent time-step $t+1$ at any given time, and the small final diagonal block $\boldsymbol{D}_{n_t+1}$.

To efficiently compute the diagonal inverse elements $(\boldsymbol{Q}_{x|y}^{-1})_{ii}$, we derive a recursive strategy making use of the already computed Cholesky decomposition $\boldsymbol{L}_{x|y}\boldsymbol{L}_{x|y}^T = \boldsymbol{Q}_{x|y}$ and its particular nonzero structure. Our approach is sim-


n_s
n_s
n_s
$n_t \cdot n_s$
n_b
n_s
n_b
n_s
n_s
$\boldsymbol{D}_1$ $\boldsymbol{E}_1^T$
$\boldsymbol{E}_1$ $\boldsymbol{D}_2$ $\boldsymbol{E}_2^T$
$\boldsymbol{E}_2$ $\boldsymbol{D}_3$
n_s
$\boldsymbol{F}_1^T$
$\boldsymbol{F}_2^T$
$n_t \cdot n_s$
$\boldsymbol{D}_{n_t}$
n_s
n_b
$\boldsymbol{F}_1$ $\boldsymbol{F}_2$
$\boldsymbol{D}_{n_t+1}$

(a) *Sparsity structure of the precision matrices $\boldsymbol{Q_x}$. Diagonal and off-diagonal blocks have size $n_s \times n_s$, where n_s is the #spatial nodes, except for the last one which is of size $n_b \times n_b$ and contains the precision matrix of the prior related to the fixed effects. On the main diagonal are $n_t + 1$ blocks, where n_t is #time steps. This gives rise to a matrix of size $n = (n_s \cdot n_t + n_b) \times (n_s \cdot n_t + n_b)$.*

(b) *Sparsity structure of the precision matrices $\boldsymbol{Q_{x|y}}$, which are of the same size as (a), but in addition to the prior precision matrix $\boldsymbol{Q_x}$ also contain information related to the data. The sparsity structure of the spatio-temporal component is not affected by conditioning on the data while sparsity structure related to the fixed effects generally becomes dense.*

Figure 3.3

ilar to methods used in quantum transport simulations where solutions to non-equilibrium Green's functions also necessitate selected inversions, see e.g. [108, 90, 109] as well as for Kalman-Bucy filtering [110]. In both cases, the authors derive strategies to efficiently compute the block diagonal elements of the inverse of block tridiagonal matrices. We extend this to block tridiagonal arrowhead matrices, starting the derivation from the following identities

$$\boldsymbol{Q} = \boldsymbol{\Sigma}^{-1} = \boldsymbol{L}\boldsymbol{L}^T \iff \boldsymbol{\Sigma} = (\boldsymbol{L}\boldsymbol{L}^T)^{-1} \iff \boldsymbol{\Sigma}\boldsymbol{L} = \boldsymbol{L}^{-T}. \tag{3.2.1}$$

We assume $\boldsymbol{Q}$ to be a symmetric positive-definite block tridiagonal arrowhead matrix. Its inverse $\boldsymbol{\Sigma}$ is generally dense but inherits the properties of symmetry and positive-definiteness. We follow the block notation given in Figure 3.3b and write Equation 3.2.1 using this submatrix notation. The inverse of an upper triangular matrix remains upper triangular. The blocks $\boldsymbol{L}_{D_i}^{-T}$ are the inverses of

Algorithm 1 Block Cholesky Factorization

```
 1: for $i = 1,2\dots n_t - 1$ do
 2:     $L_{D_i} = \text{chol}(D_i)$
 3:     $L_{E_i} = E_i \cdot L_{D_i}^{-T}$
 4:     $L_{F_i} = F_i \cdot L_{D_i}^{-T}$
 5:     $D_{i+1} = D_{i+1} - L_{E_i} \cdot L_{E_i}^T$
 6:     $F_{i+1} = F_{i+1} - L_{F_i} \cdot L_{E_i}^T$
 7:     $D_{n_t+1} = D_{n_t+1} - L_{F_i} \cdot L_{F_i}^T$
 8: end for
 9: $L_{D_{n_t}} = \text{chol}(D_{n_t})$
10: $L_{F_{n_t}} = F_{n_t} \cdot L_{D_{n_t}}^{-T}$
11: $D_{n_t+1} = D_{n_t+1} - L_{F_{n_t}} \cdot L_{F_{n_t}}^T$
12: $L_{D_{n_t+1}} = \text{chol}(D_{n_t+1})$
```

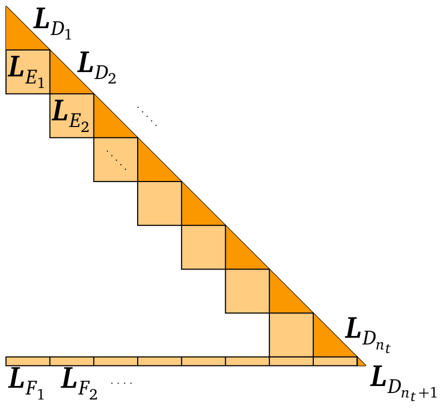


Figure 3.4: ***Left-Panel:*** *Algorithm for Block Cholesky factorization of block tridiagonal arrowhead matrices* $\mathbf{Q}_{x|y}$. ***Right-Panel:*** *Corresponding sparsity pattern and labeled subblocks of the resulting Cholesky factor* $\mathbf{L}_{x|y}\mathbf{L}_{x|y}^T = \mathbf{Q}_{x|y}$.

the individual block $L_{D_i}^T$. The $*$ denotes unknown nonzero entries.

$$\begin{bmatrix} \Sigma_{11} & \Sigma_{12}\dots & \Sigma_{1n} & \Sigma_{1n_t+1} \\ & \Sigma_{22} & & \\ \vdots & \ddots & & \vdots \\ & & \Sigma_{n_tn_t} & \Sigma_{n_tn_t+1} \\ \Sigma_{n_t+11} & \cdots & \Sigma_{n_t+1n_t} & \Sigma_{n_t+1n_t+1} \end{bmatrix} \cdot \begin{bmatrix} L_{D_1} & 0 & & \cdots & 0 \\ L_{E_1} & L_{D_2} & & & \\ & \ddots & \ddots & & \vdots \\ & & L_{E_{n_t-1}} & L_{D_{n_t}} & 0 \\ L_{F_1} & \cdots & L_{F_{n_t-1}} & L_{F_{n_t}} & L_{D_{n_t+1}} \end{bmatrix} = \begin{bmatrix} L_{D_1}^{-T} & * & * & * & * \\ 0 & L_{D_2}^{-T} & * & * & * \\ \vdots & & \ddots & * & * \\ & & & L_{D_{n_t}}^{-T} & * \\ 0 & \cdots & & 0 & L_{D_{n_t+1}}^{-T} \end{bmatrix} \tag{3.2.2}$$

Using Equation 3.2.2 we can extract the following identities.

$$\Sigma_{n_t+1n_t+1} L_{D_{n_t+1}} = L_{D_{n_t+1}}^{-T} \quad \Rightarrow \Sigma_{n_t+1n_t+1} = L_{D_{n_t+1}}^{-T} L_{D_{n_t+1}}^{-1} \tag{3.2.3}$$

$$\Sigma_{n_t+1n_t} L_{D_{n_t}} + \Sigma_{n_t+1n_t+1} L_{F_{n_t}} = 0 \quad \Rightarrow \Sigma_{n_t+1n_t} = -\Sigma_{n_t+1n_t+1} L_{F_{n_t}} L_{D_{n_t}}^{-1} \tag{3.2.4}$$

$$\Sigma_{n_tn_t} L_{D_{n_t}} + \Sigma_{n_tn_t+1} L_{F_{n_t}} = L_{D_{n_t}}^{-T} \quad \Rightarrow \Sigma_{n_tn_t} = L_{D_{n_t}}^{-T} L_{D_{n_t}}^{-1} - \Sigma_{n_tn_t+1} L_{F_{n_t}} L_{D_{n_t}}^{-1} \tag{3.2.5}$$

While for all $1 \leq i \leq n_t - 1$ we can derive the following.

$$\begin{aligned}
\boldsymbol{\Sigma}_{ii}\boldsymbol{L}_{D_i} + \boldsymbol{\Sigma}_{ii+1}\boldsymbol{L}_{E_i} + \boldsymbol{\Sigma}_{in_t+1}\boldsymbol{L}_{F_i} &= \boldsymbol{L}_{D_i}^{-T} \\
\Rightarrow \boldsymbol{\Sigma}_{ii} &= \boldsymbol{L}_{D_i}^{-T}\boldsymbol{L}_{D_i}^{-1} - \boldsymbol{\Sigma}_{ii+1}\boldsymbol{L}_{E_i}\boldsymbol{L}_{D_i}^{-1} - \boldsymbol{\Sigma}_{in_t+1}\boldsymbol{L}_{F_i}\boldsymbol{L}_{D_i}^{-1} \qquad (3.2.6)\\
\boldsymbol{\Sigma}_{i+1i}\boldsymbol{L}_{D_i} + \boldsymbol{\Sigma}_{i+1i+1}\boldsymbol{L}_{E_i} + \boldsymbol{\Sigma}_{i+1n_t+1}\boldsymbol{L}_{F_i} &= \boldsymbol{0} \\
\Rightarrow \boldsymbol{\Sigma}_{i+1i} &= -\boldsymbol{\Sigma}_{i+1i+1}\boldsymbol{L}_{E_i}\boldsymbol{L}_{D_i}^{-1} - \boldsymbol{\Sigma}_{i+1n+1}\boldsymbol{L}_{F_i}\boldsymbol{L}_{D_i}^{-1} \qquad (3.2.7)\\
\boldsymbol{\Sigma}_{n_t+1i}\boldsymbol{L}_{D_i} + \boldsymbol{\Sigma}_{n_t+1i+1}\boldsymbol{L}_{E_i} + \boldsymbol{\Sigma}_{n_t+1n_t+1}\boldsymbol{L}_{F_i} &= \boldsymbol{0} \\
\Rightarrow \boldsymbol{\Sigma}_{n_t+1i} &= -\boldsymbol{\Sigma}_{n_t+1i+1}\boldsymbol{L}_{E_i}\boldsymbol{L}_{D_i}^{-1} - \boldsymbol{\Sigma}_{n_t+1n_t+1}\boldsymbol{L}_{F_i}\boldsymbol{L}_{D_i}^{-1} \qquad (3.2.8)
\end{aligned}$$

Taking Equation 3.2.6 and substituting the unknown terms for Equation 3.2.7 and Equation 3.2.8, as well as using the fact that $\boldsymbol{\Sigma}_{ij}^T = \boldsymbol{\Sigma}_{ji}$, one obtains

$$\begin{aligned}
\boldsymbol{\Sigma}_{ii} = \boldsymbol{L}_{D_i}^{-T}\boldsymbol{L}_{D_i}^{-1} &- (-\boldsymbol{\Sigma}_{i+1i+1}\boldsymbol{L}_{E_i}\boldsymbol{L}_{D_i}^{-1} - \boldsymbol{\Sigma}_{i+1n_t+1}\boldsymbol{L}_{F_i}\boldsymbol{L}_{D_i}^{-1})^T\boldsymbol{L}_{E_i}\boldsymbol{L}_{D_i}^{-1} \\
&- (-\boldsymbol{\Sigma}_{n_t+1i+1}\boldsymbol{L}_{E_i}\boldsymbol{L}_{D_i}^{-1} - \boldsymbol{\Sigma}_{n_t+1n_t+1}\boldsymbol{L}_{F_i}\boldsymbol{L}_{D_i}^{-1})^T\boldsymbol{L}_{F_i}\boldsymbol{L}_{D_i}^{-1},
\end{aligned} \qquad (3.2.9)$$

and after rearranging

$$\begin{aligned}
\boldsymbol{\Sigma}_{ii} = \boldsymbol{L}_{D_i}^{-T}(\boldsymbol{I} + \boldsymbol{L}_{E_i}^T\boldsymbol{\Sigma}_{i+1i+1}\boldsymbol{L}_{E_i} &+ \boldsymbol{L}_{F_i}^T\boldsymbol{\Sigma}_{n_t+1n_t+1}\boldsymbol{L}_{F_i} \\
&+ \boldsymbol{L}_{F_i}^T\boldsymbol{\Sigma}_{n_t+1i+1}\boldsymbol{L}_{E_i} + \boldsymbol{L}_{E_i}^T\boldsymbol{\Sigma}_{i+1n_t+1}\boldsymbol{L}_{F_i})\boldsymbol{L}_{D_i}^{-1}.
\end{aligned} \qquad (3.2.10)$$

Using the above equations we can deduce an efficient algorithm, see Algorithm 2, to recursively compute the blocks $\boldsymbol{\Sigma}_{ii}$ starting at $i = n_t + 1$, performing an upward traversal through the matrix. During each iteration i, we make use of the previously computed diagonal blocks $\boldsymbol{\Sigma}_{i+1i+1}$ and $\boldsymbol{\Sigma}_{n_t+1n_t+1}$. One additionally requires the off-diagonal blocks $\boldsymbol{\Sigma}_{n_t+1i}$ for $2 \leq i \leq n_t$ that are computed recursively using Equation 3.2.4 and 3.2.8. This blocked approach bears, again, the advantage that each step of the iterative selected block inversion only requires the submatrices associated with the current time-step t as well as the previously computed time-step $t+1$ at any given time, besides the initially computed small diagonal block $\boldsymbol{\Sigma}_{n_t+1n_t+1}$.

Algorithm 2 Selected Block Inversion

```
1: $\boldsymbol{\Sigma}_{n_t+1 n_t+1} = \boldsymbol{L}_{D_{n_t+1}}^{-T} \cdot \boldsymbol{L}_{D_{n_t+1}}^{-1}$
2: $\boldsymbol{\Sigma}_{n_t+1 n_t} = -\boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_{n_t}} \boldsymbol{L}_{D_{n_t}}^{-1}$
3: $\boldsymbol{\Sigma}_{n_t n_t} = \boldsymbol{L}_{D_{n_t}}^{-T} (\boldsymbol{I} + \boldsymbol{L}_{F_{n_t}}^{T} \boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_{n_t}}) \boldsymbol{L}_{D_{n_t}}^{-1}$
4: for $i = n_t - 1, n_t - 2, \dots, 1$ do
5:     $\boldsymbol{\Sigma}_{ii} = \boldsymbol{L}_{D_i}^{-T} (\boldsymbol{I} + \boldsymbol{L}_{E_i}^{T} \boldsymbol{\Sigma}_{i+1 i+1} \boldsymbol{L}_{E_i} + \boldsymbol{L}_{F_i}^{T} \boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_i} + \boldsymbol{L}_{F_i}^{T} \boldsymbol{\Sigma}_{i+1 n_t+1} \boldsymbol{L}_{E_i} + \boldsymbol{L}_{E_i}^{T} \boldsymbol{\Sigma}_{n_t+1 i+1} \boldsymbol{L}_{F_i}) \boldsymbol{L}_{D_i}^{-1}$
6:     $\boldsymbol{\Sigma}_{n_t+1 i} = -(\boldsymbol{\Sigma}_{n_t+1 i+1} \boldsymbol{L}_{E_i} + \boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_i}) \boldsymbol{L}_{D_i}^{-1}$
7: end for
```

3.2.3 Computational Complexity

We dedicate this section to discussing the computational cost of the proposed block Algorithms 1 and 2, which we refer to as block tridiagonal arrowhead (BTA) approach. We compare it to the respective alternatives, entirely dense routines and entirely sparse approaches as employed in PARDISO. An overview of $\boldsymbol{Q}_{\boldsymbol{x}}, \boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}} \in \mathbb{R}^{n \times n}$, with $n = n_s \cdot n_t + n_b$, can be found in Table 3.1. The entirely dense version of both algorithms scale cubically in n. The BTA approach takes the block tridiagonal structure of $\boldsymbol{Q}_{\boldsymbol{x}}$ into account, and therefore the cost for the factorization (Algorithm 1) or selected block inversion (Algorithm 2) scales linearly in n_t but cubically in n_s. The matrix $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$ contains the additional arrowhead structure. Therefore, there is an extra cost associated to those mixed terms, which is, however, relatively small when $n_b \ll n_s$, as in our case. For PARDISO, we have that while using planar spatial meshes and by using nested dissection [111] the matrix $\boldsymbol{Q}_{\boldsymbol{x}}$ can be factorized with a complexity of $O((n_s \cdot n_t)^{1.5} + n_b^3)$. This nested dissection method can also be applied to reorder $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$, thus resulting in a complexity of $O((n_s \cdot n_t)^{1.5} + (n_s \cdot n_t + 1) n_b^3)$.

It is additionally worth noting that the dimension of $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$ is determined by the number of latent parameters in the model and does not change with the number of available observations. Similarly, the sparsity pattern of the spatio-temporal component remains unaffected with increasing numbers of observations. The remaining part of the matrix is generally assumed to be dense, and thus the complexity of all approaches, and in particular Algorithm 1 and 2 are independent of the number of observations.

	DENSE CHOL. / INV.	**BTA** CHOL / BLK. INV.	**PARDISO** CHOL. / SEL. INV.
$\mathbf{Q}_x$	$O((n_s n_t + n_b)^3)$	$O(n_s^3 n_t + n_b^3)$	$O((n_s n_t)^{3/2} + n_b^3)$
$\mathbf{Q}_{x\|y}$	$O((n_s n_t + n_b)^3)$	$O(n_s^3 n_t + (n_s n_t + 1) n_b^3)$	$O((n_s n_t)^{3/2} + (n_s n_t + 1) n_b^3)$

Table 3.1: *Overview of the computational complexity for the required Cholesky factorization and matrix inversion of* $\mathbf{Q}_x$ *and* $\mathbf{Q}_{x|y}$*, see Figure 3.3. We compare a standard dense approach with our proposed BTA approach and the sparse solver PARDISO. Here,* n_s *denotes the spatial block size,* n_t *the number of time steps and* n_b *the number of fixed effects.*

3.3 Contrasting the Two Approaches

Naturally, the question arises when one solver is preferable over the other. The CPU-based sparse approach is evidently more versatile, as it is applicable to arbitrary sparsity patterns. It is also less restrictive in terms of hardware requirements. Despite the increasing availability of GPU accelerators, many users of the R-INLA package perform their simulations on their personal laptops or small work stations which do not meet the necessary hardware specifications of the blocked approach. Of course, it would also be possible to implement a CPU-based dense block version. Its implementation would, however, not be competitive compared to the GPU variant, as GPUs excel at dense matrix-matrix operations [112]. It would, for most models, also be less performant than the sparse solver.

Thus, the BTA solver is currently a specialized approach designed for large-scale spatio-temporal inference tasks, or other inference problems whose underlying precision matrices share the same sparsity pattern, and whose computational intensities demand high-performance computing hardware architectures. The results presented in Chapter 5 will demonstrate its superior performance over the sparse approach, for exactly these types of models. Developing heuristic permutation schemes which reorder precision matrices with arbitrary sparsity patterns in block tridiagonal arrowhead form to make the BTA solver more broadly applicable, would be a valuable advancement in the future.

Chapter 4

Multi-Layer Parallelism in INLA

The algorithmic design of the INLA methodology offers various opportunities to leverage parallelism, ranging from algorithm-specific independently executable tasks to parallelizable linear algebra operations. The work presented in this chapter is dedicated to identifying these opportunities, rethinking the algorithmic design to create them, as well as discussing how to best utilize them.

Starting from the coarsest or outermost level of parallelism, we can observe that in various stages of the algorithm, one requires evaluations of $f(\boldsymbol{\theta})$ for different values of $\boldsymbol{\theta}$, see e.g. Section 2.2.2, Step 1 and 2. The former consists of the optimization phase, where one requires not only an approximation to Equation 2.2.5 but also its gradient for every iteration of the quasi-Newton method. As we use a finite difference approximation to obtain this estimate, we can perform the necessary function evaluations in parallel. In Step 2, numerous further parallel function evaluations are required for the second order finite difference approximation of the Hessian at the mode. Subsequently, and depending on the integration strategy, there are additional parallelizable function evaluations needed around the mode of Equation 2.2.1, i.e., Step 3. Likewise, the selected matrix inversions can be performed simultaneously in Step 4. In all cases, the operations are embarrassingly parallel. Further opportunities for parallelism present themselves within every function evaluation of $f(\boldsymbol{\theta})$. In some cases, the numerator and denominator of $f(\boldsymbol{\theta})$ can be computed simultaneously, i.e., allowing for parallel factorizations of $\boldsymbol{Q}_x$ and $\boldsymbol{Q}_{x|y}$. This depends, however, on the type of likelihood of the model, and will be discussed in more detail in Section 4.2.1 Additionally, the linear algebra operations of each matrix factorization and selected inversion (as well as additional standard matrix operations) are parallelized through the employed numerical solvers. We first present the shared memory model implemented in R-INLA, and subsequently discuss the hybrid shared-memory approach of $\text{INLA}_{\text{DIST}}$, which also includes a parallel copy-compute scheme for CPU-GPU data transfers.

4.1 A Shared Memory Approach for R-INLA

We put forward a two-layer multithreaded parallel scheme for R-INLA, which is realized using OpenMP [113]. The first layer directly corresponds to the coarsest level of parallelism, and consists of the parallel function evaluations that arise throughout Step 1–3, the parallel selected matrix inversions throughout Step 4 and a parallel line search scheme. A visual representation can be found in Figure 4.1.

The second level of parallelism is set within the linear algebra operations. This is within the PARDISO library, or within matrix-matrix and matrix-vector operations. PARDISO employs the graph-based matrix reordering software METIS to obtain a matrix permutation that allows for independent, and thus parallelizable, factorizations of submatrices. The same permutation is used to allow for parallelism during the forward-backward substitution and selected matrix inversion. A detailed description is provided below.

4.1.1 Level 1 – Function Evaluations

During the optimization phase a BFGS algorithm is employed to determine the optimum $\boldsymbol{\theta}^*$. A BFGS algorithm is an iterative quasi-Newton method for solving unconstrained nonlinear optimization problems [54]. Quasi-Newton methods resemble Newton methods without requiring knowledge of the second order derivative. Instead, they use gradient information to form an approximation to the Hessian in every iteration. They exhibit better convergence properties than first order methods, like gradient descent, at almost no increased cost. Among them, the BFGS algorithm is the most popular choice [54, Ch. 6]. It minimizes a differentiable function $f(\boldsymbol{\theta})$, starting from an initial guess $\boldsymbol{\theta}^0$ by using the following update formulae

$$\boldsymbol{\theta}_{l+1} = \boldsymbol{\theta}^l - \alpha^l \boldsymbol{B}_l^{-1} \nabla f_{\boldsymbol{\theta}^l}. \tag{4.1.1}$$

Here $\nabla f_{\boldsymbol{\theta}^l}$ denotes the gradient of f at $\boldsymbol{\theta}^l$, $\boldsymbol{B}_l$ is the approximation to the Hessian of f at $\boldsymbol{\theta}^l$ and $0 < \alpha^l < 1$ is the step size in iteration l. Information on the construction of $\boldsymbol{B}^l$ and a general overview can be found in [54, Ch. 6].

In each iteration the gradient $\nabla f_{\boldsymbol{\theta}^l}$ is approximated numerically using a finite difference scheme [114], where each partial derivative $\partial f / \partial \theta_i$ is approximated using either a first order forward or central difference. As an example we show the central difference approximation

$$\frac{\partial f}{\partial \theta_i}(\boldsymbol{\theta}) \approx \frac{f(\boldsymbol{\theta} + \boldsymbol{\epsilon}_i) - f(\boldsymbol{\theta} - \boldsymbol{\epsilon}_i)}{2\,||\boldsymbol{\epsilon}_i||} \quad \text{for all } i, \tag{4.1.2}$$

where the vector $\boldsymbol{\epsilon}_i$ is of the same dimension as $\boldsymbol{\theta}$ and contains only zeros except for the i-th component which contains a small value $\epsilon > 0$. This way, the finite difference approximation is computed along the coordinate axes. It is, however, also easily possible to use other bases. INLA makes use of knowledge from previous iterations to choose directional derivatives exhibiting more robust numerical properties and hence faster overall convergence, see [55] for details. Independently of the choice of basis, the directional derivatives are computed for each component of $\boldsymbol{\theta}$ and each time entail one or two function evaluations of f.

We can see from Equation 4.1.2 that all function evaluations are independent from each other. In the new parallelization scheme we are using this to our advantage by computing the function values, i.e., $f(\boldsymbol{\theta} + \boldsymbol{\epsilon}_1), f(\boldsymbol{\theta} - \boldsymbol{\epsilon}_1), f(\boldsymbol{\theta} + \boldsymbol{\epsilon}_2), \ldots$ simultaneously in each iteration, see Figure 4.1. Here, each $f(\boldsymbol{\theta}_i^l)$ denotes a different function value $f(\boldsymbol{\theta}^l \pm \boldsymbol{\epsilon}_i)$ during the l-th iteration.

For a central difference scheme we need two times as many function evaluations as there are hyperparameters, which can now all be computed in parallel instead of sequentially, while also computing $f(\boldsymbol{\theta}^l)$ itself. So, if e.g. $\dim(\boldsymbol{\theta}) = 3$, we can theoretically have a 7-fold speedup during the gradient computation. This means the introduced parallelism allows the gradient to be computed at almost no further cost in addition to the already necessary iterative evaluations of $f(\boldsymbol{\theta})$, see Figure 4.1.

There are other methods to obtain gradient information, in particular automatic differentiation [115]. In the ideal case, it offers a highly accurate solution at runtimes comparable to a single function evaluation [115], similar to the parallelized finite difference approximation. It requires, however, an adaptation of the basic linear algebra operations and then allows for parallelism within these operations. This is not trivially done for non-standard cases [116] and would require major changes in the PARDISO software library. Additionally, if the likelihood is non-Gaussian we have the inner optimization routine within each function evaluation, for which we are not aware of existing automatic differentiation methods.

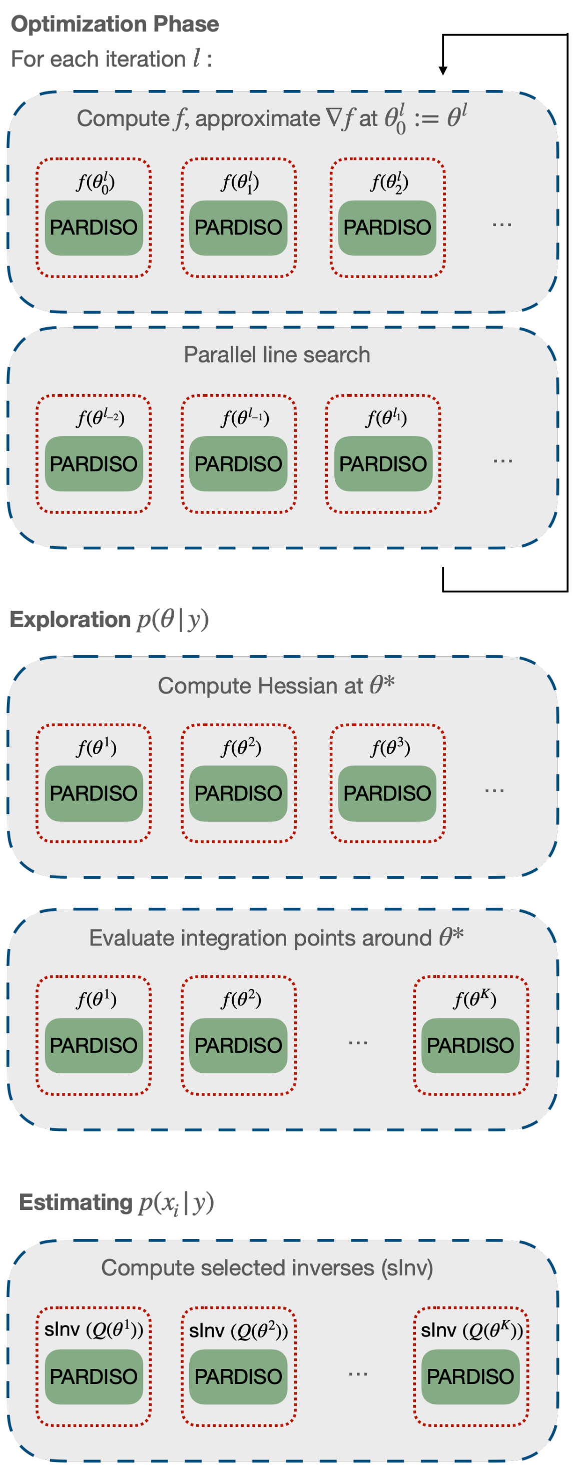

Optimization Phase
For each iteration l :
Compute f, approximate ∇f at $\theta_0^l := \theta^l$
$f(\theta_0^l)$ PARDISO
$f(\theta_1^l)$ PARDISO
$f(\theta_2^l)$ PARDISO
…
Parallel line search
$f(\theta^{l_{-2}})$ PARDISO
$f(\theta^{l_{-1}})$ PARDISO
$f(\theta^{l_1})$ PARDISO
…
Exploration $p(\theta \mid y)$
Compute Hessian at θ^*
$f(\theta^1)$ PARDISO
$f(\theta^2)$ PARDISO
$f(\theta^3)$ PARDISO
…
Evaluate integration points around θ^*
$f(\theta^1)$ PARDISO
$f(\theta^2)$ PARDISO
…
$f(\theta^K)$ PARDISO
Estimating $p(x_i \mid y)$
Compute selected inverses (sInv)
sInv $(Q(\theta^1))$ PARDISO
sInv $(Q(\theta^2))$ PARDISO
…
sInv $(Q(\theta^K))$ PARDISO

Figure 4.1: *Simplified overview of the parallelization scheme within INLA throughout the different phases of the algorithm. Each dashed blue box indicates a region of level 1 parallelism. Each dotted red box indicates a region of level 2 parallelism within the parallel level 1 regions. At the end of every parallel region (level 1 & 2) there is an intrinsic synchronization, where execution halts until the assigned threads have concluded their tasks.*

After the mode of $\tilde{p}(\boldsymbol{\theta}|\boldsymbol{y})$ is found, we compute the Hessian at the mode using a second order finite difference scheme. This requires further function evaluations that can be performed in parallel, see Figure 4.1. Next, we can also perform the function evaluations at the evaluation points $\{\boldsymbol{\theta}^k\}_{k=1}^K$ in parallel. Finally, we have to estimate the posterior marginal variances. To do so we compute the partial inverses of the precision matrices $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta}^k)$ at the integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$. We also implemented this to be executed in parallel, as the arising precision matrices are independent from each other.

Each described set of parallelizable operations are embarrassingly parallel, meaning that each task is completely independent from the rest. From a theoretical perspective, we can therefore ideally expect that the number of employed threads exactly corresponds to the observed speedup over the sequential version throughout these parallel regions. There is, of course, a maximum number of tasks within each of these parallel regions (indicated by the blue dashed boxes in Figure 4.1), e.g. the number of necessary function evaluations to compute the gradient using a forward difference, which is $d(\boldsymbol{\theta})$, in addition to evaluating f itself. Beyond this number, the increased thread count, is not beneficial anymore in all phases of the algorithm.

4.1.2 Parallel Line Search

In every iteration of the BFGS algorithm, a value for the step size α^l needs to be chosen. Determining a suitable value for α^l is crucial to the convergence of the method and is done through a line search procedure, see e.g. [54, Ch. 3] for an overview. The most common approach is to use a mixture of artificially chosen upper and lower bounds, as well as the value from the previous iteration, to propose the next step size α_l. Then, a check is performed to see if a certain condition is met to either accept or reject the suggested step size. If it is rejected, the step size is reduced, and the check is performed again, if it is accepted, $\boldsymbol{\theta}^l$ is updated as described in Equation 4.1.1 and one proceeds to the next iteration. For this check, the function f is evaluated at the potential new candidate which, as discussed previously, is an expensive operation to perform, but necessary in order to determine the validity of the new step. Hence, every time a step is rejected, another sequentially computed function evaluation is required.

To be more efficient, we introduce a parallel line search strategy to make better use of the available resources. Instead of computing different values for α^l sequentially, we define a search interval $I = (\boldsymbol{\theta}^l, \boldsymbol{\theta}^l - \gamma^l \boldsymbol{p}^l)$, where $\boldsymbol{p}^l := \boldsymbol{B}_l^{-1} \nabla f_{\boldsymbol{\theta}^l}$ and $\gamma^l > 0$ is an upper bound to α^l. Hence, I contains all possible solutions of Equation 4.1.1 for $0 < \alpha^l \leq \gamma^l$. Depending on the number of available threads, we define a number of points $\boldsymbol{\theta}^{l_i}$ on I for which we evaluate all $f(\boldsymbol{\theta}^{l_i})$ in parallel. The easiest option would be to now choose the candidate $\boldsymbol{\theta}^{l_i}$ that minimizes f as the next iterate. However, it has shown to be advantageous to fit a second order polynomial q through the newly evaluated points using robust regression [117, 118]. This way, slight inaccuracies in the function evaluations (that can e.g. arise from more complicated likelihoods and require an inner optimization loop) are counter balanced. We additionally add two more evaluation points in positive $\boldsymbol{p}^l$ direction close to $\boldsymbol{\theta}^l$ to stabilize the polynomial fitting process close to the global optimum. This does not increase the overall runtime, as they can be evaluated in parallel with the other $\boldsymbol{\theta}^{l_i}$. Robust regression differs from regular regression in the sense that each pair $(\boldsymbol{\theta}^{l_i}, f(\boldsymbol{\theta}^{l_i}))$ gets assigned a weight $w_{\boldsymbol{\theta}^{l_i}}$, which will make it more or less influential on the overall fitting process. There are a number of commonly used weighting functions w. We have chosen to use the so-called bisquare weighting, as in our experience this has been giving the best results. After finding the second order polynomial q, its minimum is determined on I and chosen as the next candidate $\boldsymbol{\theta}^{l+1}$. In Figure 4.2 we can see an illustration of the new strategy, while Figure 4.1 shows the parallel line search within the overall parallelization scheme. The advantages of the new strategy are the mitigation of inherently sequential function evaluations in the reject-accept check, as well as an improved step size choice. This can lower the number of required iterations until convergence, and hence also the overall runtime, see Section 5 for numerical results.

The theoretical speedup that can be obtained is hard to predict, as it highly depends on the properties of f. If in the sequential case a lot of new stepsize candidates get rejected, and hence require additional function evaluations or if there are larger numerical errors, the potential for speedup is higher. However, this is of course not known a priori. As a rule of thumb one can bear in mind that the more complicated the model, and in particular the likelihood (or likelihoods if there are multiple), the more significant is typically the gain through employing the parallel line search routine.

4.1.3 Level 2 – Sparse Linear Solvers

The PARDISO library offers routines to efficiently provide solutions to various problems commonly arising in the field of sparse linear algebra [119]. In R-INLA the PARDISO implementation for Cholesky decomposition, subsequent forward-backward substitution and partial inversion are invoked. PARDISO is a state-of-the-art direct solver that has been in active development for many years. To achieve excellent performance it relies on numerous strategies ranging from thor-

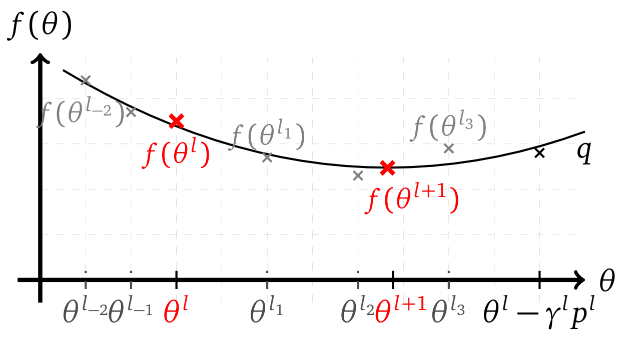


Figure 4.2: *Illustration of the parallel line search using robust regression. Let $\boldsymbol{\theta}^l$ be the current iterate with function value $f(\boldsymbol{\theta}^l)$, new search direction $\boldsymbol{p}^l$, and search interval $I = (\boldsymbol{\theta}^l, \boldsymbol{\theta}^l - \gamma^l \boldsymbol{p}^l)$ on which points $\boldsymbol{\theta}^{l_i}$ are defined, with $\boldsymbol{\theta}^{l_0} := \boldsymbol{\theta}^l$, and all $f(\boldsymbol{\theta}^{l_i})$ are evaluated in parallel. The evaluation points $\boldsymbol{\theta}^{l_{-1}}, \boldsymbol{\theta}^{l_{-2}}$ in positive $\boldsymbol{p}^l$ direction are added for numerical stabilization. The polynomial q is fitted using robust regression and its minimum becomes the next iterate $\boldsymbol{\theta}^{l+1}$.*

ough algorithm selection to detailed hardware considerations. In this work we want to solely focus on the employed parallelism and refer the interested reader to [86] for a comprehensive overview.

As described in Section 3.1.1, a symmetric permutation found through nested dissection is applied to $\boldsymbol{Q}$ before computing its Cholesky decomposition $\boldsymbol{L}$. This reordering technique does not only reduce the arising fill-in but also creates independent block matrices $\boldsymbol{Q}_A, \boldsymbol{Q}_B$, see Figure 3.2, that are only connected through a small submatrix $\boldsymbol{Q}_S$. Hence, it is possible to factorize $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$ in parallel, before factorizing $\boldsymbol{Q}_S$. The reordering generated through nested dissection has a recursive pattern, and thus $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$ themselves exhibit the same structure. Both of them, again, contain two independent blocks and a connecting submatrix which corresponds to the separating set of the associated graph. The reordered matrix can therefore be factorized in parallel, starting from the set of smallest block matrices. If sufficient compute resources are available, this introduction of parallelism drastically reduces the computational time and especially exhibits much better scaling. While it is not possible to give precise bounds on expected speedup without specifying more graph theoretical concepts and make assumptions on the sparsity pattern of the matrix, one can see that the recursive subdivisions induce a logarithmic growth. For details, see [120].

A similar principle can be applied to determine the selected inverse of $\boldsymbol{Q}$, for which we have seen in Section 3.1.2 that only the non-zero entries of $\boldsymbol{L}$ are required. We can compute the same subgraphs in parallel, as in the Cholesky decomposition. This time, however, we first compute the values of the submatrix connected to the separating set, and referred to as $\boldsymbol{Q}_S$ in Figure 3.2. Once we have recursively computed all necessary inverse elements belonging to the indices of $\boldsymbol{Q}_S$, we can determine the inverse elements of $\boldsymbol{\Sigma}$ with indices belonging to $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$ in parallel. This is possible since $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$ are only connected through $\boldsymbol{Q}_S$ and these entries have already been computed at this point. Since for larger matrices, each submatrix $\boldsymbol{Q}_A, \boldsymbol{Q}_B$ was again permuted following the same strategy, we can recursively traverse the matrix, in the opposite direction as in the Cholesky decomposition, computing the required entries of $\boldsymbol{\Sigma}$ in many parallel regions [86].

PARDISO uses OpenMP to carry out the simultaneous computations. Thus, we obtain a nested OpenMP structure, as we have multiple matrix factorizations or inversions carried out in parallel, see Section 4.1.1 and 4.1.2. Additionally to the parallelism within PARDISO, OpenMP is used on the second level to parallelize matrix operations, e.g. during the computation of matrix-matrix or matrix-vector products and while assembling matrices.

4.2 A Hybrid Memory Approach for INLA$_{\text{DIST}}$

The parallel model of the hybrid memory approach of INLA$_{\text{DIST}}$ is similar to shared memory approach of R-INLA as it leverages the same parallelizable function evaluations as R-INLA as well as parallelism within the linear algebra operations. The coarse level parallelism is realized using MPI [121], i.e. the parallel function evaluations are executed by different processes. In the Gaussian likelihood case, INLA$_{\text{DIST}}$ allows for an additional layer of parallelism that will be described in the subsequent section. When the CPU-based sparse solver is employed, INLA$_{\text{DIST}}$ uses the same multi-threading approach as provided through PARDISO as R-INLA. But since each process and thus each function evaluation can be assigned to a different node, a larger number of threads is available per matrix factorization. Each node only performs one matrix factorization, instead of multiple ones as in the shared memory approach. This leads to a performance increase, not only due to the increased number of available cores but also in terms of increased available memory bandwidth per matrix factorization. When the GPU-based BTA solver is used, there are also parallelizable linear algebra operations. Foremost, many of the dense linear algebra operations that are performed on each subblock can be executed efficiently in parallel, and in particular all dense matrix-matrix multiplications which are especially efficient on GPU accelerators. But also other operations like the triangular solve to update the off-diagonal non-zero blocks allow for parallel computations. Additionally, a considerable amount of host-device memory transfers are necessary during the matrix factorization and selected matrix inversion. These can, however, mostly be performed in parallel to the required computations. Section 4.2.2 discusses the details of how this is achieved.

4.2.1 The Gaussian Likelihood Case

When the associated data of the model is normally distributed, it is possible to make use of an additional layer of parallelism within INLA$_{\text{DIST}}$. Generally, we require Equation 2.2.3 to be evaluated at the mode $\boldsymbol{x}^*(\boldsymbol{\theta})$ of the true conditional $p(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ to yield a reasonable approximation. This stems from the fact that a second order Taylor series expansion around the mode $\boldsymbol{x}^*(\boldsymbol{\theta})$ is used to approximate the likelihood, as part of $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ which becomes inaccurate when evaluated far from this point. It implies that we first have to determine $\boldsymbol{x}^*(\boldsymbol{\theta})$ using the "inner" iteration before being able to evaluate the remaining components of $f(\boldsymbol{\theta})$ in $\boldsymbol{x}^*(\boldsymbol{\theta})$, or equivalently Equation 2.2.3. On the other hand, if the likelihood is Gaussian, $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ is exact, and therefore we can evaluate Equation 2.2.3 in an arbitrary configuration $\tilde{\boldsymbol{x}}$. The evaluation of $p(\boldsymbol{x}|\boldsymbol{\theta})$ and $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$ in any $\tilde{\boldsymbol{x}}$the computation of the Cholesky factorizations of the precision matrices $\boldsymbol{Q}_{\boldsymbol{x}}$ and $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$, as well as solve a linear system of equations to determine the conditional mean $\boldsymbol{x}^*(\boldsymbol{\theta})$ of $p_G(\boldsymbol{x}|\boldsymbol{\theta},\boldsymbol{y})$. This can, however, be done in parallel, as it is not necessary to know $\boldsymbol{x}^*(\boldsymbol{\theta})$ when evaluating $p(\boldsymbol{x}|\boldsymbol{\theta})$. This allows for the introduction of an additional layer of parallelism within the overall algorithmic design of INLA$_{\text{DIST}}$, see Figure 4.3 for a schematic overview. In terms of implementation this is realized as a shared as well as distributed memory approach, and can be chosen according to individual preferences, depending on the capacity of the available compute architecture and the size of the precision matrices, $\boldsymbol{Q}_{\boldsymbol{x}}$ and $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$.

4.2.2 A Parallel Copy-Compute scheme for GPU

Within INLA$_{\text{BTA}}$, and therefore more specifically for the GPU implementation of the block Cholesky decomposition, described in Algorithm 1 and the selected block inversion, described in Algorithm 2 intensive memory transfers between host and device memory are necessary. For large-scale applications, the block-dense Cholesky factor $\boldsymbol{L}_{x|y}$ can exceed the available GPU memory, rendering it necessary to store on host memory. As $\boldsymbol{L}_{x|y}$ is required for subsequent forward-backward substitutions as well as potential selected matrix inversions. This implies that instead of initializing the original matrix $\boldsymbol{Q}_{x|y}$ and pre-allocating stor-

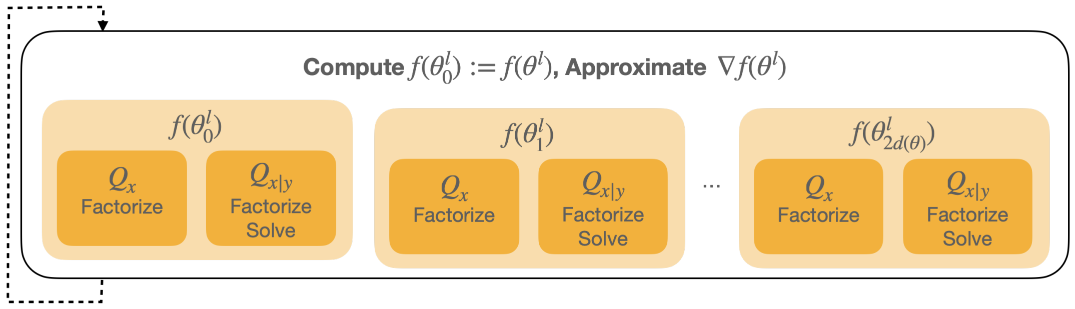

Compute $f(\theta_0^l) := f(\theta^l)$, Approximate $\nabla f(\theta^l)$
$f(\theta_0^l)$
Q_x Factorize
$Q_{x|y}$ Factorize Solve
$f(\theta_1^l)$
Q_x Factorize
$Q_{x|y}$ Factorize Solve
...
$f(\theta_{2d(\theta)}^l)$
Q_x Factorize
$Q_{x|y}$ Factorize Solve

Figure 4.3: *Schematic overview of the parallel scheme during one iteration of the optimization phase (for simplicity, without the line search routine). The finite difference gradient approximation entails $2 \cdot d(\boldsymbol{\theta}) + 1$ independent function evaluations of f which are executed by separate processes. Each process contains multiple threads that can execute the Cholesky factorizations of $\boldsymbol{Q}_x$ and $\boldsymbol{Q}_{x|y}$ independently of each other. Alternatively, each function evaluation is executed by 2 processes such that $\boldsymbol{Q}_x$ and $\boldsymbol{Q}_{x|y}$ can be factorized by separate processes. Each factorization itself is then parallelized by employing parallel linear algebra operations.*

age for the Cholesky factor that is to be computed on the device before the start of the computations, the data transfers are performed column block-wise. For ease of notation, we will refer to $\boldsymbol{Q}_{x|y}$ as $\boldsymbol{Q}$ and $\boldsymbol{L}_{x|y}$ as $\boldsymbol{L}$ for the remainder of this section. For the block Cholesky factorization, this results in the updated Algorithm 3 including the block-wise memory transfers.

Here $\boldsymbol{Q}_i$ denotes the blocks $\boldsymbol{Q}_i = \{\boldsymbol{D}_i, \boldsymbol{E}_i, \boldsymbol{F}_i\}$ and respectively for newly computed column blocks $\boldsymbol{L}_i$ of the Cholesky factor. Typically, the original sparse matrix $\boldsymbol{Q}$ only has a fraction of the non-zeros of the blocked Cholesky factor $\boldsymbol{L}$[1]. Therefore, the host to device transfers are relatively cheap to perform, whereas the device to host transfers involve the entire dense blocks and therefore bear a larger potential for speedup through parallelism. The implementation is realized using CUDA streams or MAGMA queues, respectively. It is possible to derive a similar parallel copy-compute scheme for the selected block inversion. In this case, the large and therefore time-consuming memory transfers are from host to device, when copying the previously computed Cholesky factor $\boldsymbol{L}$ back to the GPU as needed for the computations in the upwards block traversal of Algorithm 2. The set of selected inverse entries of interest, which we denote by $\mathcal{S}$, are contained within the original sparsity pattern of $\boldsymbol{Q}$ and thus much fewer than the entire dense block tridiagonal arrowhead structure. This implies that the necessary device to host transfers are less memory intensive. Algorithm 4 shows the

[1]Exact numbers are provided for the concrete applications provided in Chapter 5

Algorithm 3 Block Factorization with Parallel Host-Device Memory Transfers
Compute *Copy*

```
 1:                                                               MemCopyHostToDev($\boldsymbol{Q_1}, \boldsymbol{Q}_{n_t+1}$)
 2: for $i = 1, 2 \ldots n_t - 1$ do
 3:     $\boldsymbol{L}_{D_i} = \mathrm{chol}(\boldsymbol{D}_i)$                                     MemCopyHostToDev($\boldsymbol{Q_{i+1}}$)
 4:     $\boldsymbol{L}_{E_i} = \boldsymbol{E}_i \cdot \boldsymbol{L}_{D_i}^{-T}$
 5:     $\boldsymbol{L}_{F_i} = \boldsymbol{F}_i \cdot \boldsymbol{L}_{D_i}^{-T}$
 6:     $\boldsymbol{D}_{i+1} = \boldsymbol{D}_{i+1} - \boldsymbol{L}_{E_i} \cdot \boldsymbol{L}_{E_i}^T$                       MemCopyDevToHost($\boldsymbol{L_i}$)
 7:     $\boldsymbol{F}_{i+1} = \boldsymbol{F}_{i+1} - \boldsymbol{L}_{F_i} \cdot \boldsymbol{L}_{E_i}^T$
 8:     $\boldsymbol{D}_{n_t+1} = \boldsymbol{D}_{n_t+1} - \boldsymbol{L}_{F_i} \cdot \boldsymbol{L}_{F_i}^T$
 9: end for
10: $\boldsymbol{L}_{D_{n_t}} = \mathrm{chol}(\boldsymbol{D}_{n_t})$                                     MemCopyHostToDev($\boldsymbol{Q_{n_t+1}}$)
11: $\boldsymbol{L}_{F_{n_t}} = \boldsymbol{F}_{n_t} \cdot \boldsymbol{L}_{D_{n_t}}^{-T}$
12: $\boldsymbol{D}_{n_t+1} = \boldsymbol{D}_{n_t+1} - \boldsymbol{L}_{F_{n_t}} \cdot \boldsymbol{L}_{F_{n_t}}^T$                     MemCopyDevToHost($\boldsymbol{L}_{n_t}$)
13: $\boldsymbol{L}_{D_{n_t+1}} = \mathrm{chol}(\boldsymbol{D}_{n_t+1})$
14:                                                               MemCopyDevToHost($\boldsymbol{L}_{n_{t+1}}$)
```

updated parallel copy-compute version of the selected block inversion, where $\boldsymbol{L_i} = \{\boldsymbol{L}_{D_i}, \boldsymbol{L}_{E_i}, \boldsymbol{L}_{F_i}\}$ as before and $\boldsymbol{\Sigma_i}|_{\mathcal{S}}$ denotes the selected inverse entries of interest in $\boldsymbol{\Sigma_i}$.

Algorithm 4 Selected Block Inversion Parallel Host-Device Memory Transfers
Compute *Copy*

```
1:                                                                MemCopyHostToDev($\boldsymbol{L_{n_t+1}}$)
2: $\boldsymbol{\Sigma}_{n_t+1 n_t+1} = \boldsymbol{L}_{D_{n_t+1}}^{-T} \cdot \boldsymbol{L}_{D_{n_t+1}}^{-1}$                          MemCopyHostToDev($\boldsymbol{L_{n_t}}$)
3: $\boldsymbol{\Sigma}_{n_t+1 n_t} = -\boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_{n_t}} \boldsymbol{L}_{D_{n_t}}^{-1}$                MemCopyDevToHost($\boldsymbol{\Sigma_{n_t+1}}|_{\mathcal{S}}$)
4: $\boldsymbol{\Sigma}_{n_t n_t} = \boldsymbol{L}_{D_{n_t}}^{-T}(\boldsymbol{I} + \boldsymbol{L}_{F_{n_t}}^T \boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_{n_t}}) \boldsymbol{L}_{D_{n_t}}^{-1}$       MemCopyHostToDev($\boldsymbol{L_{n_t-1}}$)
5: for $i = n_t - 1, n_t - 2, \ldots, 1$ do
6:     $\boldsymbol{\Sigma}_{ii} = \boldsymbol{L}_{D_i}^{-T}(\boldsymbol{I} + \boldsymbol{L}_{E_i}^T \boldsymbol{\Sigma}_{i+1 i+1} \boldsymbol{L}_{E_i} + \boldsymbol{L}_{F_i}^T \boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_i}$    MemCopyHostToDev($\boldsymbol{L_{i-1}}$)
       $+ \boldsymbol{L}_{F_i}^T \boldsymbol{\Sigma}_{i+1 n_t+1} \boldsymbol{L}_{E_i} + \boldsymbol{L}_{E_i}^T \boldsymbol{\Sigma}_{n_t+1 i+1} \boldsymbol{L}_{F_i}) \boldsymbol{L}_{D_i}^{-1}$       MemCopyDevToHost($\boldsymbol{\Sigma_{i+1}}|_{\mathcal{S}}$)
7:     $\boldsymbol{\Sigma}_{n_t+1 i} = -(\boldsymbol{\Sigma}_{n_t+1 i+1} \boldsymbol{L}_{E_i} + \boldsymbol{\Sigma}_{n_t+1 n_t+1} \boldsymbol{L}_{F_i}) \boldsymbol{L}_{D_i}^{-1}$
8: end for
9:                                                                MemCopyDevToHost($\boldsymbol{\Sigma_1}|_{\mathcal{S}}$)
```

Chapter 5

Applications

In this chapter, we present various applications of the INLA methodology and analyze their performances, putting the previous chapters into practice. The first part focuses on case studies using R-INLA, while the second part is dedicated to performing inference with INLA$_{\text{DIST}}$. For the former three different case studies are considered, for which we perform strong scaling analyses of the two-layer, shared memory, parallel scheme. We then examine the effects of the parallel line search and their combined effects. We also discuss how to best utilize a given number of available cores respective to the model parameter dimensions. For INLA$_{\text{DIST}}$, we first validate its accuracy by using synthetic datasets, as well as by comparing it against R-INLA. We also examine its single node performance and carry out a FLOP analysis. Further, its strong scaling is discussed for both linear solvers, its temporal and spatial scaling properties and the parallel copy-compute scheme of the BTA solver. We conclude with a large-scale climate application, modeling air temperature over the area of the United States.

5.1 Case Studies using R-INLA

We present performance results of the newly introduced parallelization strategies using previously published real-world applications that make use of the R-INLA package, and are briefly introduced below. We will not go into the details of the individual application as all the details can be found in the original publications, and instead focus on the computational aspects involved. All numerical experiments for Case Study I & II were performed on a single node machine with 755 GB of main memory and 26 dual-socket Intel(R) Xeon(R) Gold 6230R CPU @ 2.10GHz, totaling 52 cores. The large number of cores allows us to demonstrate the full performance gains that can be obtained through parallelism. All numerical experiments for Case Study III were performed on an Apple M1 Mac mini with 16 GB of memory, 4 performance and 4 efficiency cores. We chose this machine to illustrate that we also obtain performance gains through parallelism on regular desktop computers, laptops and notebooks, although to a smaller extent than on larger computer architectures.

Case Study I: Joint survival modeling of randomized clinical trial. Rustand et al. presented in [91], with R package **INLAjoint**[1], a joint survival model consisting of multivariate longitudinal markers paired with competing risks of events. The different submodels are linked to each other through shared or correlated random effects. The authors consider a randomized placebo controlled trial for the treatment of the rare autoimmune disease primary biliary cholangitis (PBC) which affects the liver. They examine how different biomarkers, in combination with the treatment (medication vs. placebo), affect the competing risks of death and liver transplantation. Their model setup includes 7 different correlated likelihoods, which allows them to detect more complex dependency structures, that go unseen in simpler approaches. The resulting model has a particularly large number of hyperparameters, with dimension $d(\boldsymbol{\theta}) = 50$, see Table 5.1 for details.

Case Study II: Cortical surface modeling of human brain activation. Spencer et al. show in [92] that functional brain responses can be reliably estimated using cortical surface-based spatial Bayesian generalized linear models (GLMs). Functional magnetic resonance imaging (fMRI) data from individual subjects is used to identify areas of significant activation during a task or stimulus. The authors use the stochastic partial differential equations approach [47, 4, 122] for manifolds to define a spatial GMRF field on the surface of the brain, employing geodesic distances. This allows to flexibly encode spatial dependency structures in the model while maintaining sparsity in the precision matrices. Their approach defines a GLM that fits the framework presented in 2.1.3. The likelihood is assumed to follow a Gaussian distribution, and therefore does not require an inner optimization loop for every function evaluation. More details can be found in [92, 9] which also includes information about the corresponding R package **BayesfMRI**[2]. For this work we fit a single subject, single repetition set up using fMRI data for 4 tasks. The dimension of the hyperparameter space $d(\boldsymbol{\theta}) = 9$, two for the parameterization of the spatial field of each task and a noise term

[1] `https://github.com/DenisRustand/INLAjoint`

[2] `https://github.com/mandymejia/BayesfMRI` (ver.0.1.8), `https://github.com/danieladamspencer/BayesGLM_Validation`

for the observations. The images are preprocessed for standardization, and the surface of the brain is discretized using a triangular mesh. We use two different spatial discretizations that are determined by the resampling size, which subsequently influence the dimension of the latent parameter space and the number of considered observations.

Case Study III: Spatial variation in Leukemia survival data. As a final example we consider a study on spatial variation in Leukemia survival data from [123, 47]. The additive linear predictor of the hazard function includes 5 fixed effects, which relate to attributes like age, sex, economic status and inflammatory markers. To capture the spatial variation, it additionally includes a spatial GMRF random field, which induces a three-dimensional hyperparameter space. The fitted model provides survival estimates given the covariates and spatial locations.

		$d(\boldsymbol{\theta})$	# lat. par.	# obs.
CS I	Joint Survival Model	50	51 290	27 330
CS II	Medium Brain Activation Model	9	35 544	2 541 396
CS II	Large Brain Activation Model	9	183 624	13 129 116
CS III	Spatial Leukemia Model	3	7 945	6 174

Table 5.1: *Overview of the different case studies, showing the parameter dimensions, including the number of hyperparameters ($d(\boldsymbol{\theta})$), number of latent parameters (# lat. par) and number of observations (# obs.)*

5.1.1 Leveraging Parallel Function Evaluations

The level 1 parallelism enables simultaneous function evaluations, as described in Section 4.1.1. They arise during the gradient computation, the Hessian approximation at the mode, the objective function evaluations at the evaluation points and the partial inversion scheme around the mode. The speedup that can theoretically be obtained through parallelization depends on the number of hyperparameters. If a forward difference scheme is employed, a maximum speedup of $d(\boldsymbol{\theta})+1$, is possible during the gradient computation. For a central difference scheme, it is $2 \cdot d(\boldsymbol{\theta})$. If not specified otherwise, R-INLA employs a mix of forward and central difference approximations. The theoretical speedup during the Hessian approximation and the partial inversion scheme depends on the number of integration points, which, however, typically exceed $d(\boldsymbol{\theta})+1$ by far unless the “empirical Bayes” integration strategy is chosen. While these are the computationally most costly operations, there are other non-parallelizable steps required. Additionally, setting up a parallel OpenMP environment always creates overhead. Hence, the theoretical speedup can almost never be attained. For relatively small applications, the observed speedup through parallelism is usually not as significant as for larger problems, because the sequential parts and the induced overhead make up a larger part of the total time. On the other hand, larger applications using a very large number of threads might not exhibit further speed up after a certain thread count, as memory management can become the limiting factor. We will be able to observe both effects when looking at performance results of parallelizing level 1. In Figure 5.1 we provide scaling results with varying number of threads on level 1 and a fixed number of threads on level 2 for Case Studies I & II. Instead of showing the total runtime for each case, we divide it by the total number of function evaluations of f, as the number of evaluations can vary a bit depending on randomly set initial values and numerical imprecision due to different round-off errors. For Case Study I, we observe an almost ideal reduction in time per f evaluation until 20 threads, yielding an impressive speed of a factor of 15 at 20 threads compared to the single-threaded version. When employing more than 20 threads, the parallel efficiency starts to reduce. Case Study II has fewer hyperparameters, hence the maximum attainable speedup on level 1 is lower, nevertheless providing significant improvements for up to 10 threads with a speedup of almost 6 compared to the single-threaded version.

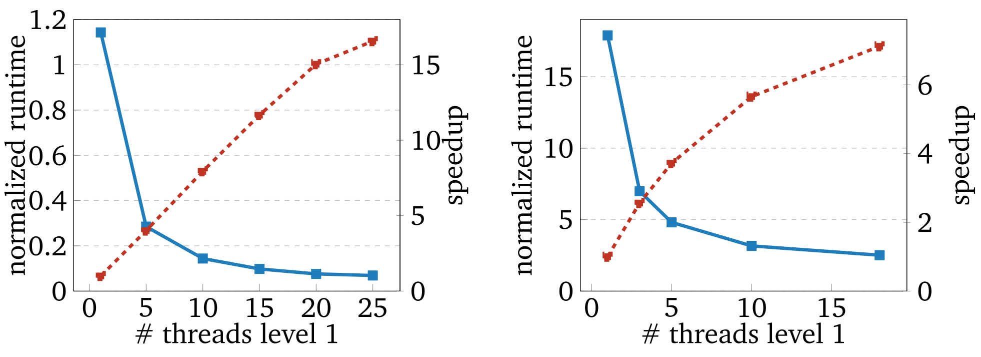
# threads level 1	1	2	3	4
time per fn in sec	0.031	0.023	0.017	0.016
speedup	1	1.4	1.8	2

Table 5.2: *Effects of using different numbers of threads on level 1 for Case Study III.*

We also analyze the effects of level 1 parallelism for Case Study III. This model is of much smaller dimension than the first two and hence a larger part of the overall runtime is dedicated to sequential operations like setting up or the initializations of the parallel regions. Nevertheless, we observe a speedup of a factor of 2 when using 4 threads over the sequential version, see Table 5.2. The results confirm the theoretical expectation, that the level 1 parallelism scales close to linearly, in particular for thread counts smaller than $d(\boldsymbol{\theta}) + 1$.

5.1.2 Leveraging Parallel Line Search

The accuracy of robust regression depends on the number of available regressors, which in turn is dependent on the number of available threads. In our experience making use of the parallel line search implementation only becomes



(a) *Joint Survival Model* (b) *Large Brain Activation Model*

Figure 5.1: *The solid blue line shows the normalized runtime (total runtime divided by number of function evaluations) over number of threads on level 1 (# threads level 2 fixed to one). The dashed red line is the speedup over the single-threaded version. On average a total number of (a) 19253 and (b) 378 function evaluations were required to fit the respective model.*

advantageous if 8 threads or more are in use on level 1. Then there are sufficiently many evaluation points in the search interval I to adequately represent the original function. In Figure 5.2, we show the total runtime of Case Study I, using varying numbers of threads on level 1 with and without enabling the parallel line search. Since this model has a large number of hyperparameters, the computation of the posterior marginals of the latent parameters using the simplified Laplace approximation strategy would be the dominating the overall compute time. In this section we, therefore, used the computationally cheaper empirical Bayes' approximation, to be able to put the emphasis on the optimization phase of the algorithm, where the parallel line search feature is relevant. We can see that it lowers the overall time to solution without requiring more threads. It additionally exhibits a more stable behavior when it comes to reduction in runtime over an increasing thread count.

5.1.3 Leveraging Scalability of Sparse Linear Solvers

The level 2 parallelism occurs within each call of the PARDISO library as well as other matrix operations like matrix-matrix multiplications, following the concepts described in Section 4.1.3. The single-threaded version of the PARDISO library is already very efficient for small to moderate sized problems. The full effects of the parallelization start becoming visible for the latent parameter spaces of dimensions in the range of 10^4 and larger, and in particular for models with a

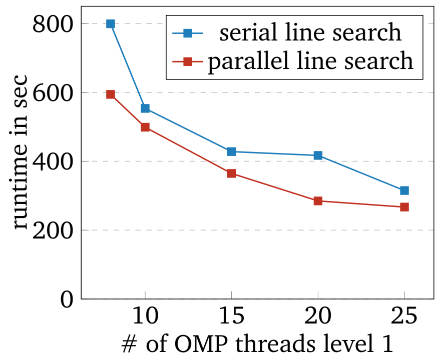


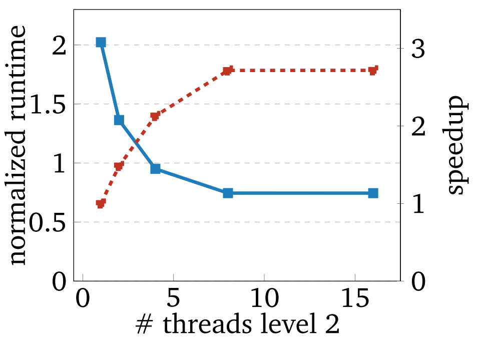
Figure 5.2: *Runtime for Case Study I, with and without parallel line search over varying numbers of threads on level 1. The number of threads on level 2 is fixed to 1. The posterior marginals of the latent parameters are approximated using the empirical Bayes integration strategy. On average the serial line search required 15286 and the parallel line search 10950 function evaluations.*

three-dimensional structure.

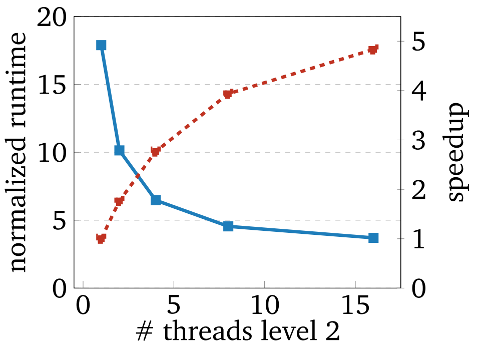
We consider both examples of Case Study II in the performance analysis of Figure 5.3. For a fixed number of threads on level 1, we observe a continuous speedup for increasing numbers of threads on level 2 up until 8 threads for the medium-sized Case Study II, leading to a maximum speedup of almost 3 over the single-threaded version. In turn, the larger-sized Case Study II continues to exhibit a speedup also beyond 8 threads, showing that larger latent parameter space benefit more from level 2 parallelism, leading to a speedup of almost 5 at 16 threads over the single-threaded version. As expected from a theoretical point of view, we do not observe a linear or close to linear speedup for the level 2 parallelism. We can see that a higher dimensional latent parameter space exhibits a higher speed up for the same number of threads. The large Case Study II also continues to benefit from higher thread counts when the medium-sized Case Study II has already reached a saturation point.

5.1.4 Combined Parallelism

In this section, we want to present results for combined parallelization strategies and discuss how to choose a favorable setup. The ideal approach depends on the available infrastructure and the problem at hand. In almost all cases, the number of available cores will be limited. Often this limit will easily be reached as the total number of threads is equal to (# threads L1) · (# threads L2). This poses the question of how best utilize the available resources. We have found

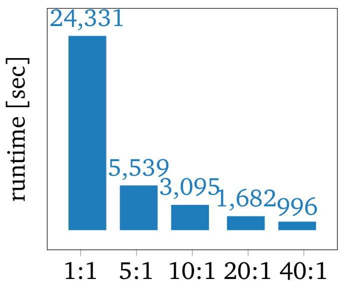


(a) *Medium Brain Activation Model*

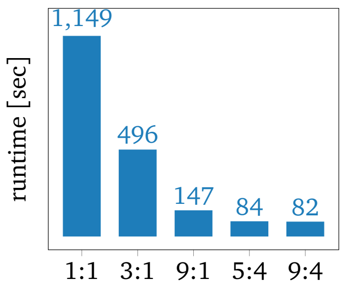


(b) *Large Brain Activation Model*

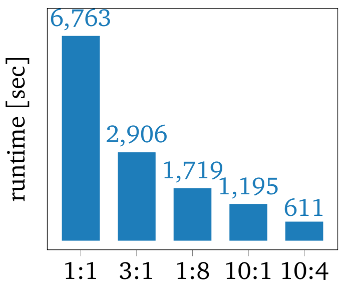
Figure 5.3: *The solid blue line shows the normalized runtime for different numbers of threads on level 2 (# threads level 1 threads fixed to one). The dashed red line shows the speedup over the single-threaded version. On average a total number of (a) 378 and (b) 452 function evaluations were required to fit the respective model.*



(a) *Joint Survival Model*



(b) *Medium Brain Act. Model*



(c) *Large Brain Act. Model*

Figure 5.4: *Runtime in seconds for different thread configurations. The first number represents the threads on level 1, the second number the threads on level 2. The total number of threads is equal to (# threads level 1) · (# threads level 2).*

that hyperthreading often does not significantly increase the performance (or is even counterproductive) and therefore only consider thread counts that are within the number of physical cores. In the previous two sections we have seen that smaller problems benefit less from larger thread counts as the overhead of thread initialization can overshadow the gain. We can see from both Figure 5.1 and 5.3 that the parallel efficiency is highest for smaller thread counts on both levels (in cases where $d(\boldsymbol{x})$ is sufficiently large). This implies that it is often favorable to distribute the available number of threads among both levels. This behavior can be observed empirically in Figure 5.4, where we present the runtime of the INLA algorithm in seconds for different thread configurations. As a rule of thumb, one might consider always assigning the first $d(\boldsymbol{\theta})+1$ threads to level 1. As both from a theoretical and empirical point of view has shown to be the best strategy. If more threads are available, one can either add threads to level 2 or further add to level 1, depending on the dimension of the latent parameter space.

In general, it becomes clear that the introduction of parallelism leads to a tremendous reduction in runtime. For Case Study I, we observe speedups of a factor 20 and more over the single-threaded version. For the medium-sized Case Study II, we observe a speedup of a factor up to 15. While for the large Case Study II, we observe a speedup of more than 10 times compared to the single-threaded version. For sufficiently large multi-core architectures, the newly introduced updates open the door to previously unfeasible modeling scales, and drastically reduce waiting times for users.

5.2 Spatio-Temporal Modeling with INLA$_{\text{DIST}}$

We present two numerical case studies using INLA$_{\text{DIST}}$, a family of synthetic datasets of varying sizes, and a real-world application concerned with continuous air temperature modeling over the United States. The numerical experiments were carried out at the Erlangen National High Performance Computing Center (NHR@FAU). The simulations using INLA$_{\text{BTA}}$ were conducted on GPGPU nodes, each equipped with 64-core dual chip AMD EPYC 7713 “Milan” processors @ 2.0 GHz and 8 Nvidia A100 GPUs. The simulations using INLA$_{\text{PARDISO}}$ were conducted on dual socket Intel Xeon Platinum 8360Y “Ice Lake” processors (36 cores per chip) @ 3.6 GHz. PARDISO was called using 32 threads, as this represents the most performant choice.

5.2.1 Experiments on Synthetic Data

We generate a family of synthetic datasets based on the model class described in Section 2.1.3, that is,

$$\boldsymbol{y} = \boldsymbol{Z}\boldsymbol{\beta} + \tilde{\boldsymbol{A}}\boldsymbol{u} + \boldsymbol{\epsilon}, \quad \text{where} \quad \epsilon_i \sim \mathcal{N}(0, \gamma_n^{-1}). \tag{5.2.1}$$

In the following we outline how the individual components of Equation 5.2.1 were chosen. The vector $\boldsymbol{\beta}$ contains the fixed effects of the model, which include an intercept. We chose a random 6-dimensional vector, as the number of fixed effects is typically small, with values between $[-5, 5]$. We sample covariates $\boldsymbol{Z}$

		n_s	n_t	n_b	n_o	n
Base Case	BC	**4002**	**250**	**6**	**2 001 000**	**1 000 506**
Temporal Scaling	TS I	4002	50	6	400 200	200 106
	TS II	4002	100	6	800 400	400 206
	TS III	4002	500	6	4 002 000	2 001 006
Spatial Scaling	SS I	4002	30	6	240 120	120 066
	SS II	10242	30	6	614 520	307 266
	SS III	16002	30	6	960 120	480 066
	SS IV	20252	30	6	1 215 120	607 566

Table 5.3: *Differently sized model problems and their corresponding key dimensions, where n_s denotes the number of spatial nodes, n_t the number of time steps, n_b the number of fixed effects, n_o the number of observations and n the number of latent parameters, thus $\boldsymbol{Q_x}, \boldsymbol{Q_{x|y}} \in \mathbb{R}^{n \times n}$ with $n = n_s \cdot n_t + n_b$.*

stemming from linear as well as nonlinear functions, with additional random uniform noise to interact with the fixed effects. The vector $\boldsymbol{u}$ is associated to the spatio-temporal field. Exemplary, we consider the entire globe as our spatial domain over equidistant time units. The discretization is done using first order finite elements as previously described. The random field $\boldsymbol{u}$ is sampled from the spatio-temporal component of $\boldsymbol{Q_x}(\boldsymbol{\theta})$ with mean zero, as by model definition. The precision matrix $\boldsymbol{Q_x}(\boldsymbol{\theta})$ is formed by the discretization of Equation 2.2.31 and independent prior variances for each fixed effect. It is parametrized by the 4-dimensional hyperparameter vector $\boldsymbol{\theta} = (\gamma_n, \gamma_s, \gamma_t, \gamma_e)^T$, which includes a noise term for the observations and the scaling parameters of the SPDE. The hyperparameters are chosen such that they give rise to a realistic system. We sample random locations over the globe from a uniform distribution to represent the measurement stations. They generally do not coincide with the nodes of the finite element mesh. For simplicity, we assume the stations to remain constant over time, which is, however, not a requirement, as each measurement is projected onto the finite element mesh using the projection matrix $\tilde{\boldsymbol{A}}$. Twice as many observations per time step are sampled as there are spatial mesh nodes.

Additionally, we select a prior for the hyperparameters $\boldsymbol{\theta}$. We choose a penalized complexity prior [124] as they are particularly suited for additive models, allow for explicitly incorporating probability statements on the parameters, and penalize unnecessary complexity in the model, as its name suggests [122]. We generate datasets of varying mesh sizes in time and space, thus giving rise to dif-

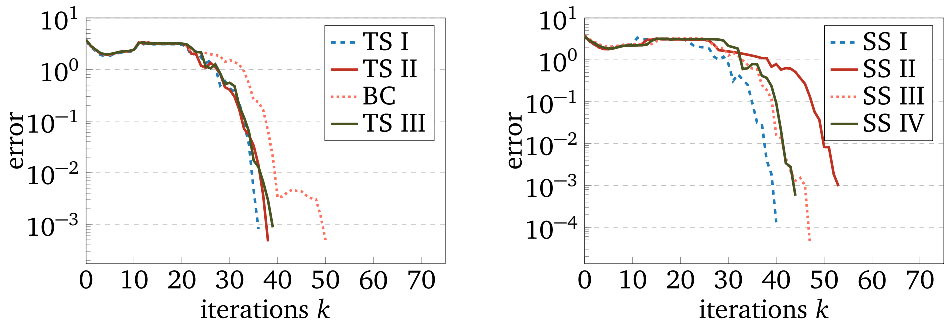


Figure 5.5: *The x-axis shows the number of iterations over the error on the y-axis, defined as $||\boldsymbol{\theta}^* - \boldsymbol{\theta}^l||$, where $\boldsymbol{\theta}^*$ denotes the optimum, $\boldsymbol{\theta}^l$ refers to the l-th iterate, and $||\cdot||$ the Euclidean norm. The results for the different spatio-temporal model sizes from Table 5.3 are shown. The same initial values were chosen for all datasets.* ***Left-Panel:*** *Model problems related to the temporal scaling, including the Base Case.* ***Right-Panel:*** *Model problems related to the spatial scaling.*

ferently sized problems, see Table 5.3. For both, the Base Case (BC) model and Temporal Scaling Case III (TS III), we have latent parameter spaces with more than 1 million unknown parameters and more than 2 million observations.

Validation

To solve the optimization problem described in Section 2.2.2, Step 1, a BFGS-algorithm in combination with a backtracking line search using Wolfe condition is employed. In Figure 5.5 we present convergence results for the different cases from Table 5.3. It can be seen that the overall convergence behavior for each of the differently sized problems is similar, while the exact number of iterations varies. In all cases the same initial guess was used.

We also compare our inference estimates to those of the R-INLA package by computing the ratio of the marginal likelihoods of the two approaches for different synthetic datasets. This ratio is called Bayes factor and is used to quantify the support of one model over another. For numeric stability, we compute the log marginal likelihoods (LMLs) and later exponentiate their difference. The LML is computed using the respectively determined modes of the hyperparameters $\boldsymbol{\theta}^*_{\text{RINLA}}$ and $\boldsymbol{\theta}^*_{\text{INLADIST}}$ and subsequently induced latent parameters $\boldsymbol{x}^*_{\text{RINLA}}$ and $\boldsymbol{x}^*_{\text{INLADIST}}$ as follows

$$\text{LML} = \log p(\boldsymbol{y}) = \log p(\boldsymbol{x}^*|\boldsymbol{\theta}^*) + \log p(\boldsymbol{y}|\boldsymbol{\theta}^*, \boldsymbol{x}^*) - \log p(\boldsymbol{x}^*|\boldsymbol{\theta}^*, \boldsymbol{y}). \quad (5.2.2)$$

According to Jeffrey's scale [125], as well as other commonly used metrics [126, 127], a Bayes factor (BF) between approximately $10^{-1/2} < \text{BF} < \sqrt{10}$, implies that the difference in approaches is "barely worth mentioning", where $\text{BF} = 1$ means that they are identically likely. Figure 5.6 shows the Bayes factor for different datasets. It can be observed that all Bayes factors' indicate an insignificant difference between approaches. We also see that for increasingly larger temporal domains, and thus more independent, identically distributed observations, the Bayes factor decreases.

Single Node Performance

We are not aware of any distributed software library that supports the model types treated in this work. The R-INLA package[3], however, provides a single node implementation which, supports nested multi-threading as presented in the previous section. Therefore, we conduct a performance comparison between R-INLA and our CPU-based $\text{INLA}_{\text{PARDISO}}$ implementation, using a set of smaller models but following the same setup as described in Section 5.2.1, on a single node machine with 52 cores (26 dual-socket Intel Xeon Gold 6230R CPU) and 755 GB of main memory, see Figure 5.6. We parallelize both versions to use the same number of cores, meaning that we cannot leverage the full multi-layer parallel scheme of our implementation, thus factorizing $\boldsymbol{Q}_x$ and $\boldsymbol{Q}_{x|y}$ sequentially. Both algorithms utilize PARDISO as an underlying solver for the computational kernel operations. We observe that for the smallest test cases they have almost identical runtimes, with $\text{INLA}_{\text{PARDISO}}$ exhibiting more consistent scaling properties as the model parameter dimensions grow. We consider normalized runtime (total runtime / # number of function evaluations) instead of absolute runtime because the number of required function evaluations is dependent on various factors. The implementation of the objective function 2.2.5 of the optimization problem differs between $\text{INLA}_{\text{DIST}}$ and R-INLA, while also changing with the problem size, dataset, mesh discretization, etc. which influences the number of required iterations. Furthermore, the METIS reordering used in PARDISO Additionally, a small total runtime might simply be attributed to a good initial guess or not very strict convergence criteria. The largest test case took less than 2 hours for $\text{INLA}_{\text{PARDISO}}$ and just over 3.5 hours for R-INLA. In the following, we will see that our methods $\text{INLA}_{\text{PARDISO}}$ and INLA_{BTA} scale well on distributed architectures (with INLA_{BTA} outperforming $\text{INLA}_{\text{PARDISO}}$ for all larger models), thus making it possible to run much larger models than the ones employed here within shorter runtimes.

[3]through the INLAspacetime package (https://github.com/eliaskrainski/INLAspacetime)

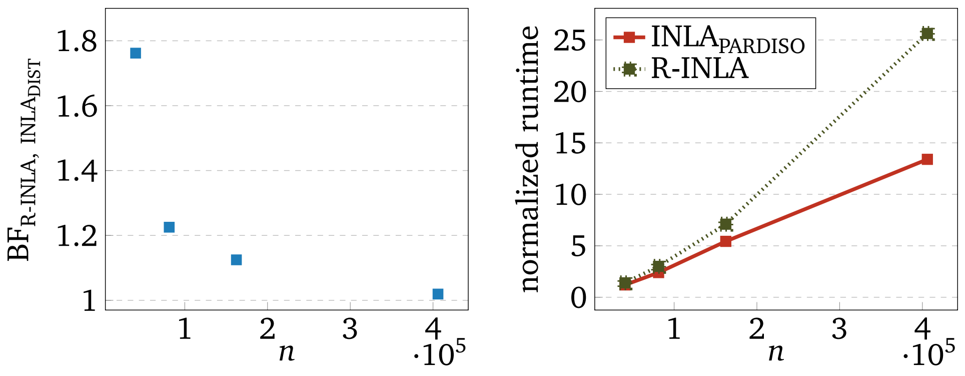
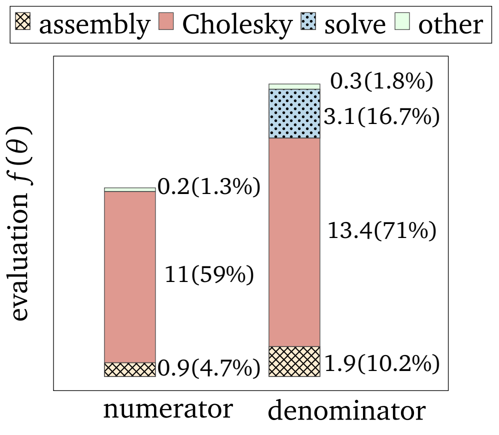


Figure 5.6: *Single node comparison of our INLA$_{DIST}$ approach and R-INLA for 4 different synthetic datasets, where the spatial grid size is fixed to $n_s = 812$ nodes with increasing number of time steps $n_t = 50, 100, 200, 500$ leading to matrix dimensions of $n \leq 4.1 \cdot 10^5$ as shown on the x-axes, and $n_o = 2 \cdot n_s n_t$ observations, respectively.* ***Left-Panel:*** *Bayes Factor (BF), i.e., ratio of the marginal likelihoods as inferred by INLA$_{DIST}$ and R-INLA over n. Bayes factors smaller than 3 (and ≥ 1), are deemed not significant, with BF $= 1$, meaning that both models are identically likely.* ***Right-Panel:*** *Run time comparison. To account for varying number of BFGS iterations until convergence, we show the normalized runtime, defined as runtime[sec]/(total # of function evaluations), over the matrix size n for INLA$_{PARDISO}$ and R-INLA, each using 36 cores.*

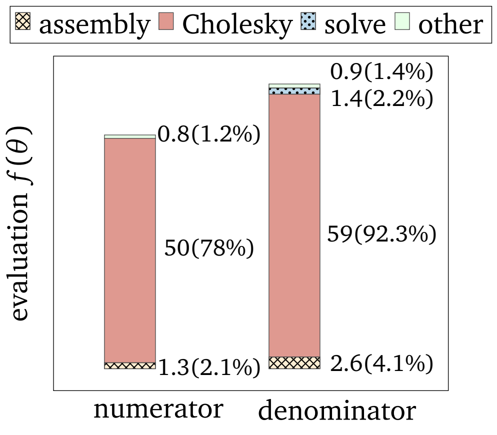
Kernel Operations and Performance

In Figure 5.7 we show the different contributions to the overall runtime for a single function evaluation of $f(\boldsymbol{\theta})$, for INLA$_{\text{BTA}}$ and INLA$_{\text{PARDISO}}$, respectively, using the Base Case from Table 5.3. The evaluation of f is split in numerator, which entails the factorization of $\boldsymbol{Q}_{\boldsymbol{x}}$ and denominator, which entails the factorization of $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$, and are executed in parallel. Therefore, the overall runtime of a function evaluation of f is determined by the maximum between the two and corresponds to 100% for each solver, respectively. The majority of the runtime for both solvers is made up by the Cholesky factorizations. They are performed on separate GPUs or nodes, for INLA$_{\text{BTA}}$ and INLA$_{\text{PARDISO}}$, respectively. The sparse matrix assembly for $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$ takes longer as it includes the dense arrowhead structure. All other operations like the evaluation of the prior of the hyperparameters and the likelihood take up a negligible amount of time with contributions of a few percent.

Figure 5.8 shows the achieved GFLOP/S (GigaFLOP per Second) of the numerical factorization of $\boldsymbol{Q}_{\boldsymbol{x}}$ and $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$, respectively, for the BTA solver and PARDISO. As expected, the former performs many more floating point operations per second than the latter as it operates entirely on dense submatrices as compared



(a) *INLA$_{BTA}$*



(b) *INLA$_{PARDISO}$*

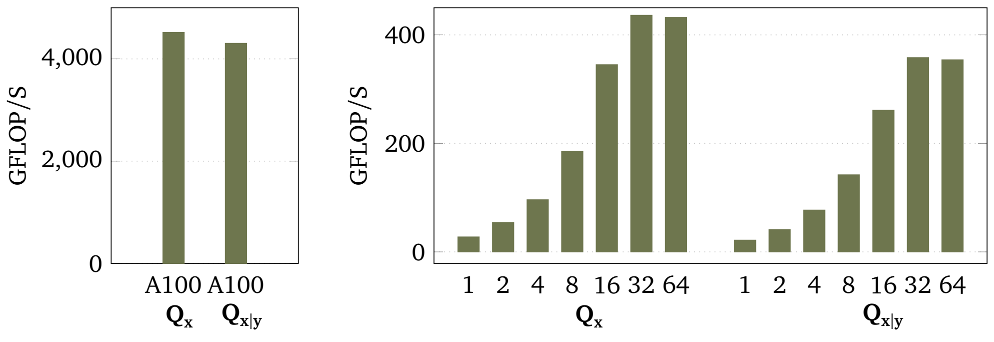
Figure 5.7: *Computational kernel operations of $f(\boldsymbol{\theta})$ for the Base Case, see Table 5.3, and split in the parallel evaluation of numerator and denominator, see Equation 2.2.3, using (a) INLA$_{BTA}$ and (b) INLA$_{PARDISO}$. For each solver type max(runtime(numerator), runtime(denominator)) denotes* 100%, *respectively. The size of the bars is relative to this maximum. The first number next to the bar denotes the respective absolute runtime in seconds for each task, the second number denotes the respective percentage. For both solvers the Cholesky decompositions of $\mathbf{Q}_x$ (numerator) and $\mathbf{Q}_{x|y}$ (denominator) dominate the runtimes. INLA$_{BTA}$ outperforms INLA$_{PARDISO}$ by a factor of 3.5.*

to sparse structures. Additionally, we can see how the number of GFLOP/S increases with the number of cores used by PARDISO up to 32 cores, where it peaks. The rate of increase over the sequential version illustrates its parallel scalability, as the number of floating point operations is independent of the core count.

Strong Scaling

In this section, we discuss performance results with respect to increasing numbers of GPUs/cores. The majority of the overall runtime of the algorithm is spent on solving the optimization problem described in Section 2.2.2, Step 1. In each iteration of the BFGS solver, $2d(\boldsymbol{\theta})+1$ parallel function evaluations are required to compute $f(\boldsymbol{\theta}^k)$ and $\nabla f(\boldsymbol{\theta}^k)$. One can observe in Figure 5.9 that our algorithm actually exhibits ideal scaling for up to 9 processes ($\dim(\boldsymbol{\theta}) = 4$) as expected.

As previously discussed, further parallelism is introduced by simultaneously factorizing $\mathbf{Q}_x$ and $\mathbf{Q}_{x|y}$. Since $\mathbf{Q}_x$ has less nonzero entries and a simpler sparsity structure, its factorization takes less time than of $\mathbf{Q}_{x|y}$. Therefore, the theoretical maximum speedup going from sequentially factorizing these two matrices



(a) *BTA Solver* (b) *PARDISO*

Figure 5.8: ***Left-Panel:*** *BTA Solver. GFLOP/S during the numerical factorization of $\mathbf{Q}_x$ and $\mathbf{Q}_{x|y}$, respectively, on an A100 GPU.* ***Right-Panel:*** *PARDISO. GFLOP/S relative to the number of cores during the numerical factorization of $\mathbf{Q}_x$ and $\mathbf{Q}_{x|y}$. It can be seen that using 64 cores no longer increases the performance. The matrices $\mathbf{Q}_x$ and $\mathbf{Q}_{x|y}$ stem from the Base Case.*

to parallel factorizations is less than a factor 2, but rather around a factor 1.6 for INLA$_{\text{BTA}}$ and 1.8 for INLA$_{\text{PARDISO}}$, see Figure 5.7 for details. This also becomes apparent in our numerical experiments. When parallelizing the Cholesky decompositions within each function evaluation on top of the parallel function evaluations, we can observe a further speed up that corresponds approximately to the theoretically achievable factor of 1.6 and 1.8 respectively, as indicated by the black lines.

In terms of wall-clock time, we can see that INLA$_{\text{BTA}}$ outperforms PARDISO by roughly a factor 3-4, independently of the number of processes. To fit the Base Case model, INLA$_{\text{BTA}}$ requires just under $1 \cdot 10^3$ seconds, i.e., about 16 minutes, when using 18 GPUs. INLA$_{\text{PARDISO}}$ requires just over 50 minutes when using 576 cores.

Temporal and Spatial Scaling

We will now discuss how our proposed methods scale with increasing numbers of n_s spatial and n_t temporal nodes, respectively. The normalized runtimes are shown with respect to the matrix size in Figure 5.10. We note that for both scaling studies, the INLA$_{\text{BTA}}$ outperforms INLA$_{\text{PARDISO}}$ by roughly a factor of 3-4 for larger problems in terms of total runtime. The analysis in Section 3.2.3 showed that the BTA solver has linear complexity in time for a single matrix factorization or selected inversion. As these are the dominating subroutines of INLA$_{\text{BTA}}$, we would ideally expect this property to be inherited. We can see that this holds true

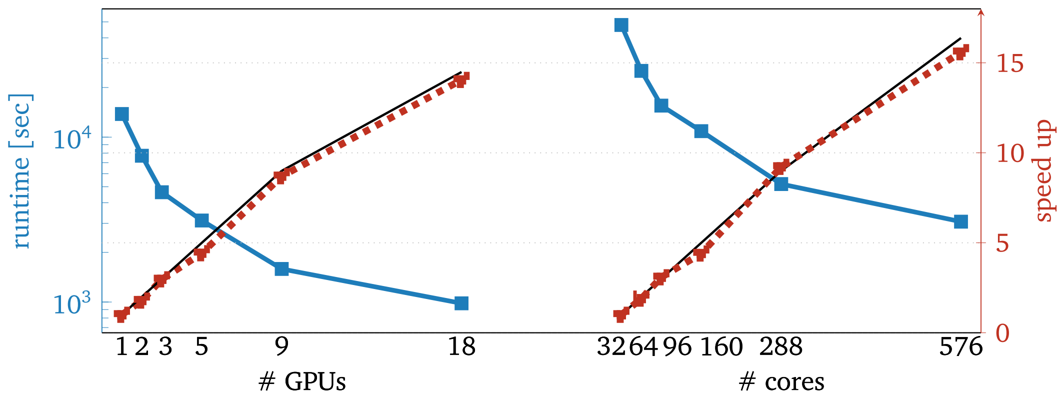
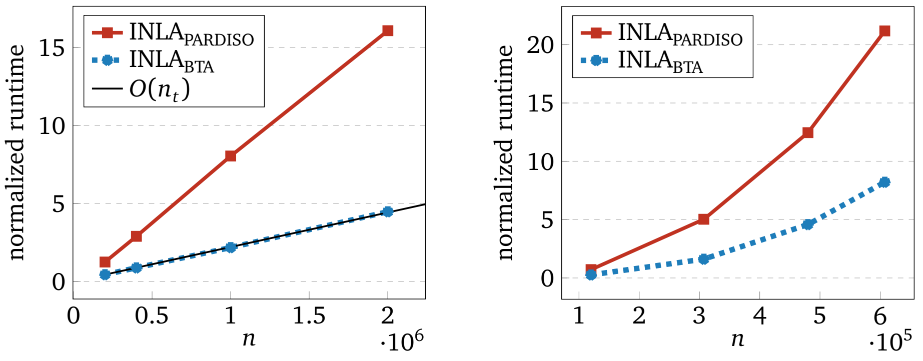


Figure 5.9: *Strong scaling plot of fitting the Base Case model. The blue line indicates the runtime in seconds using different numbers of GPUs or cores, respectively. The red line indicates the speedup over the respective base version (1 GPU/32 cores). The black dashed line represents the ideally achievable speed up.* ***Left-Panel:*** *INLA$_{BTA}$.* ***Right-Panel:*** *INLA$_{PARDISO}$.*

almost perfectly, as indicated in the graph. PARDISO's complexity with respect to temporal nodes n_t grows with a larger factor, see Table 3.1. This is in line with the observed temporal scaling behavior of INLA$_{\mathrm{PARDISO}}$. In the spatial scaling case, the complexity for both solvers is superlinear which is consistent with the results shown in Figure 5.10b. Due to the sparsity preserving SPDE approach our method scales independently of the number of observations, i.e., increasing the number of observations does not increase the runtime, see Section 2.2.1 for details.

Parallel Copy-Compute

In this section, we explore the impact of overlapping copy-compute operations within INLA$_{\mathrm{BTA}}$ compared to a sequential implementation as outlined in Section 4.2.2, utilizing two distinct test cases from Table 5.3. We analyze the runtime distribution within a single loop iteration of Algorithm 3, split into host-device memory transfers, dense Cholesky factorization of a diagonal block, triangular solves for updating the matrix rows below, and the matrix-matrix multiplications for updating subsequent coupled blocks. The results are shown in Figure 5.11. We can see that for the first example, Base Case I, which has a smaller spatial domain ($n_s = 4002$), the device to host memory transfer is relatively larger compared to the Spatial Scaling Case IV, which has a larger spatial domain ($n_s = 20252$).



(a) *Temporal Scaling* (b) *Spatial Scaling*

Figure 5.10: *The normalized runtime (runtime / (total # of function evaluations)) over the precision matrix dimension n.* ***Left Panel:*** *Fixed spatial grid size of $n_s = 4002$ nodes with increasing number of time steps, corresponding to TS I-II, BC & TS III from Table 5.3. The black dotted line (coinciding with INLA$_{BTA}$) indicates the linear complexity in n_t as discussed in Table 3.1.* ***Right Panel:*** *Fixed temporal grid size of $n_t = 30$ nodes with increasing number of spatial nodes, corresponding to SS I-IV.*

First, we discuss how the two case studies compare and where they relatively spend more time. It can be seen that when the spatial domain, and thus the diagonal block to be factorized, is smaller (BC I), relatively more time is spent within the Cholesky factorization (POTRF), the memory transfer from device to host (CpyToHost), and less time within the matrix-matrix multiplication (GEMM). For a larger spatial domain (SS IV) the matrix-matrix multiplication becomes the dominating operation whereas the previous two operations require relatively less time. This is attributed to the fact that the computational complexity of the associated GEMM is approximately $2n_s^3$, whereas POTRF is also of cubic complexity but scales with a smaller constant and the number of entries to be copied scales with $2n_s^2$. Thus, the former increasingly dominates the overall runtime as n_s increases. Going from the sequential to the parallel copy-compute version, we can see that in both cases we see a significant improvement through the parallel copy-compute implementation. In the first case, we see a reduction of almost 20%, whereas it is about 13% for the second case.

5.2.2 Case Study on Atmospheric Air Temperature

In this section, we utilize our proposed method to model the daily average air temperatures over the United States for the entire year of 2022. Temperature

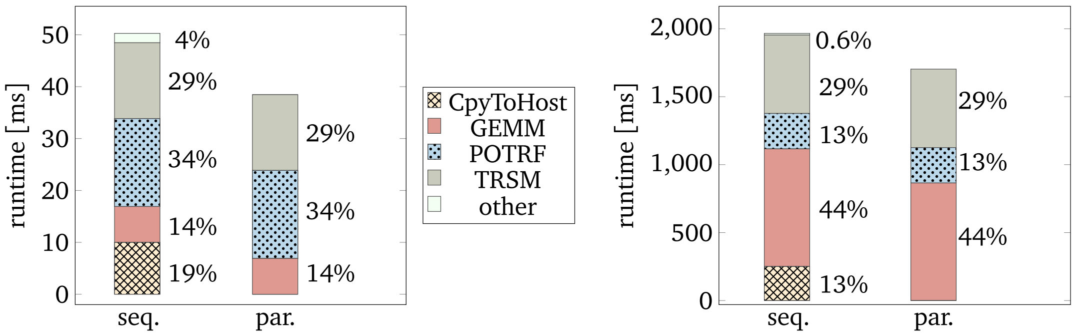


Figure 5.11: *Comparison of parallel copy-compute (Algorithm 3) compared to sequential (seq.) version (Algorithm 1) for two different case studies. We consider one loop iteration of diagonal block factorization (POTRF), the triangular solve to update the rows below (TRSM) and the matrix multiplication to update the coupled blocks (GEMM). In the parallel version the copy to host (CpyToHost) operation completely overlaps with the computation (GEMM, POTRF, TRSM).* ***Left-Panel:*** *Base Case I according to Table 5.3.* ***Right-Panel:*** *Spatial Scaling IV according to Table 5.3.*

n_s	n_t	n_b	n_o	n
2865	365	4	2 472 561	1 045 729

Table 5.4: *Model parameter dimensions for the air temperature dataset, following the notation of Table 5.3.*

measurements from different meteorological stations throughout the country are supplied by the National Centers of Environmental Information and publicly available [98]. We define the linear predictor $\eta(s,t)$ over the spatio-temporal domain as

$$\eta(s,t) = \beta_0 + \beta_1 \,\text{elevation}(s) + \beta_2 \,\text{z}(t) + \beta_3 \,\text{north}(s) + u(s,t), \tag{5.2.3}$$

where β_0 represents an overall temperature mean and β_1 is the effect of elevation at a location s. The term $z(t) = \sin(\pi t/365)$ describes a normalized seasonal trend and β_2 is the effect of this term. The variable β_3 represents the effect of the projected latitude with units in kilometers from the southeast data location, and $u(s,t)$ is the spatio-temporal field. The unknown posterior parameter distributions in the linear predictor are the fixed effects $\boldsymbol{\beta} = (\beta_0, ..., \beta_3)^T$, as well as the random field $u(s,t)$. The spatio-temporal domain is discretized using first order finite elements, whose basis functions are defined on the spatio-temporal mesh nodes, as detailed in [49]. For simplicity, we keep the same spatial discretization

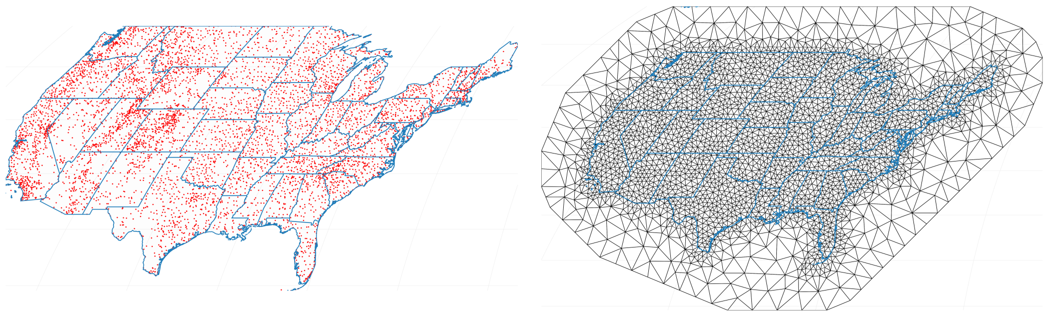


Figure 5.12: ***Left-Panel:*** *The stations where measurements were collected. Not every station collects measurements at every time step.* ***Right-Panel:*** *A map of the different US states is outlined in blue. The employed spatial discretization is shown in gray including the coarser peripheral mesh with a total of $n_s = 2865$ mesh nodes.*

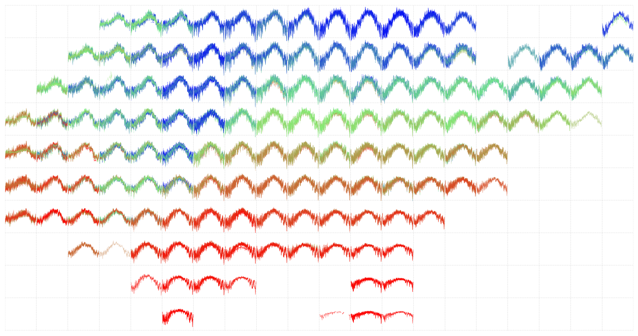


Figure 5.13: *The data shown as time series grouped around its locations and colored as function of its average from colder (blue) to warmer (red).*

at each time step. We embed the contiguous US territory, i.e., omitting Hawaii and Alaska, in a two-dimensional convex domain that is discretized using a Delauney triangulation and add a coarser peripheral mesh to buffer boundary effects. The different measurement locations and the spatial mesh are shown in Figure 5.12. The temporal mesh consists of $n_t = 365$ nodes, one for each day. It is worth noting that not all stations have measurements at every time step. There are about 2.5 million observations available throughout the entire year, see Figure 5.13. The arising latent parameter space has a dimension of over 1 million, see Table 5.4 for details.

The first inference step is to estimate the hyperparameter distributions of $\boldsymbol{\theta}$, as described in Section 2.2.4, which parametrize the noise term and the spatio-temporal field. From this the posterior mean and standard deviation for the fixed effects $\boldsymbol{\beta}$ are derived and presented in Figure 5.16. They give rise to the following model interpretation. The estimated temperature in 2022 was 10.6 degrees

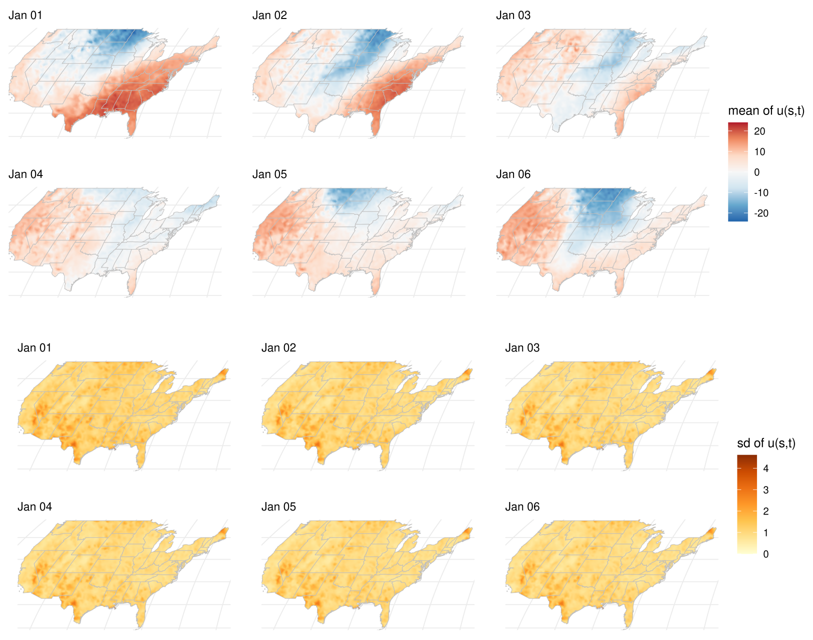


Figure 5.14: *The posterior mean and standard deviation for the $u(s,t)$ field for the first 6 days, that is January 1-6.*

Celsius, in the beginning and at the end of the year, at the southeast station location at sea level. It decreased 3.8 degrees per kilometer of elevation. On average, the temperature increased by 25.7 degrees by the middle of the year, and decreases 7.37 degrees per thousand kilometers going north from the most southeastern point. While general trends are captured by the fixed effects, the spatio-temporal field $u(s,t)$ describes the local deviations from this overall behavior. In Figure 5.14 the estimated posterior mean and standard deviation of $u(s,t)$ are shown for the first six days of the year 2022. It can be seen that on January 1st it was around 10 to 20 Celsius degrees warmer, than the fixed effects would stipulate in the southeastern part of the domain. The warmer region shrinks in size over the following days, showing the ability of this term to indeed capture short-term variations. We note that the standard deviation of $u(s,t)$ is lower in areas with a higher density of measurement stations and larger in areas with fewer measurement stations, see for example the northeastern part of the country as well as in some areas of Texas. In Figure 5.15 we show the posterior mean and standard deviation for $u(s,t)$ for selected days over the year which are around periods of two months apart. In these far apart in time estimates we can see a different spatial pattern for the posterior mean as they are related

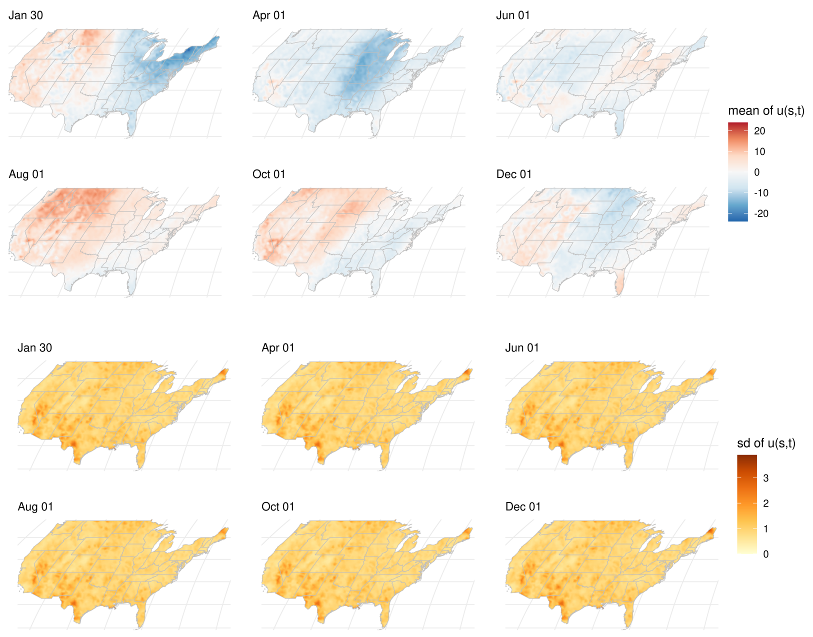


Figure 5.15: *The posterior mean and standard deviation for the $u(s,t)$ field at selected times.*

	β_0	β_1	β_2	β_3
est. mean	10.58	-3.84	25.70	-7.37
est. sd	0.35	0.006	0.34	0.16

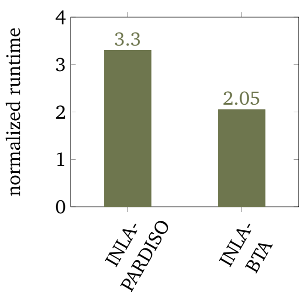


Figure 5.16: ***Left-Panel:*** *Estimated posterior fixed effects.* ***Right-Panel:*** *Normalized runtime in seconds per function evaluation for INLA$_{\mathrm{PARDISO}}$ and INLA$_{\mathrm{BTA}}$ using 576 cores or 18 GPUs, respectively.*

to the weather at these days which are reasonable to be assumed uncorrelated. We notice that with the spatial resolution used to solve the model, it was able to capture the local behavior of the weather. The posterior standard deviation looks similar for each shown time steps, which corresponds to the fact that most stations record observations daily.

The normalized runtimes are shown in Figure 5.16 using the most performant configuration for each version, that is 576 cores or 18 GPUs, respectively. The overall runtimes were just over 25 for $\mathrm{INLA}_{\mathrm{PARDISO}}$ and 15 minutes for $\mathrm{INLA}_{\mathrm{BTA}}$. The memory footprint for each (permuted) sparse Cholesky factor as computed by $\mathrm{INLA}_{\mathrm{PARDISO}}$ equates to 25.4 GB, while the $\mathrm{INLA}_{\mathrm{BTA}}$ approach requires 47.9 GB.

Chapter 6

Conclusion

The work presented in this thesis focuses on combining the methodology of integrated nested Laplace approximations with scalable high performance algorithms and their implementation for providing accurate, fast and reliable inference estimates. The main contributions are

- The conception and development of the high-performance implementation INLA$_{\text{DIST}}$ of the INLA methodology for large-scale spatio-temporal models.
- Identifying, developing, enhancing and implementing efficient algorithms handling the arising computational bottleneck operations within the existing R-INLA package as well as INLA$_{\text{DIST}}$.
- Introduction of massive parallelism through multi–layer parallel schemes, realized as a shared memory approach in R-INLA and a distributed–shared memory approach in INLA$_{\text{DIST}}$.

The INLA methodology offers a deterministic approximation scheme for estimating the posterior marginal distributions of the unknown parameters and is applicable to the class of latent Gaussian models. From a computational perspective, the methodology comprises three main steps. Firstly, it requires identifying the minimum of a non-linear optimization problem, whose objective function is an approximation to the posterior distribution of the hyperparameters and computationally expensive to evaluate. Secondly, additional objective function evaluations in proximity to the optimum are needed when accommodating non-Gaussian posterior distributions. The arising computational core operations during the objective function evaluations are Cholesky decomposition and solving linear systems of high-dimensional symmetric positive definite matrices with recurring sparsity patterns. Thirdly, the estimation of the posterior marginal variances of the latent parameters, which involves computing selected entries of the full inverse of the associated precision matrices. For the selected matrix inversion, we are interested in the entries of the full inverse that correspond to nonzero entries in the original sparse matrix. The CPU-based state-of-the-art sparse direct solver PARDISO was integrated into R-INLA and INLA$_{\text{DIST}}$ as it provides efficient routines to perform all computational bottleneck operations, i.e., Cholesky factorizations, solving linear systems and selected matrix inversions. The sparse solver leverages intelligent matrix reordering techniques which allow for large parts of the otherwise sequential computations to be parallelized and employs a separate symbolic reordering phase which can be reused.

The representation of processes which include spatial or spatio-temporal data often leads to particularly high-dimensional models. The INLA methodology in combination with the SPDE approach provide an efficient framework for performing spatio-temporal Bayesian modeling. We rely on a non-separable space-time model to capture spatio-temporal phenomena, where the spatio-temporal discretization of the model induces block tridiagonal arrowhead sparsity structure in its precision matrices. We have derived algorithmic solution strategies to handle the computational core operations, which are tailored to the recurring sparsity patterns. In particular, an efficient selected block inversion routine is proposed to compute the marginal variances of the latent parameters, without having to compute the full inverse. A GPU-accelerated implementation of these routines is put forward which is entirely based on dense block operations, while maintaining sparsity in the overall system.

The multi-layer parallel approaches in R-INLA as well as INLA$_{\text{DIST}}$ take advantage of mutually independent objective function evaluations, identify parallelizable kernel operations, and exploit concurrency within each subroutine. During the optimization phase, the parallelizable function evaluations arise in the finite difference approximation of the gradient. Additionally, a parallel line search routine is introduced using robust regression, to parallelize and stabilize the search for each next step size, being able to lower the overall number of required iterations. Subsequently, parallel function evaluations arise in the exploration of the posterior distribution of the hyperparameters. The computation of the marginal variances necessitates the selected matrix inversions in different evaluation points, which are embarrassingly parallel operations as they are independent of each other.

We empirically demonstrate the performance and scalability of the augmented R-INLA implementation on three case studies. Each of them uses a different type of model, hence posing different computational challenges. For all larger applications, we observed speedups of a factor of 10-25 times compared to the single-threaded version. Hence, lowering runtimes for some models from almost seven hours to less than 30 minutes or from more than two hours to just over ten minutes. For smaller applications we still observe speedups but to a smaller extend. In our performance analysis of the high-performance $\text{INLA}_{\text{DIST}}$ package, we show that it exhibits almost ideal strong scaling up to two times the hyperparameter dimension. Its blocked GPU-based variant scales linearly with the number of time steps, and quasilinearly for the sparse CPU-based variant. We utilize $\text{INLA}_{\text{DIST}}$ for a large-scale air temperature modeling application, using more than 1 million latent parameters and 2.5 million observations. We exhibit runtimes in the order of tens of minutes, outperforming existing approaches.

Leveraging the strengths of today's multi-core, multi-node and GPU-accelerated computers, these algorithmic advancements allow for performing inference using more complex models at higher resolution in shorter runtimes, making previously unfeasible inference tasks feasible.

Appendix A

Variational Bayes Correction – Derivation of the Derivatives

We derive the first and second order derivatives of $I(\mu)$ with respect to μ, i.e., Equation 2.2.18 and 2.2.19, including a change of variables. We remind ourselves that

$$I(\mu) = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\, dx. \tag{A.0.1}$$

For ease of notation we define

$$h(x,\mu) := \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x) \tag{A.0.2}$$

We assume that $h(x,\mu)$ is continuous and that $\int_{-\infty}^{\infty} \frac{\partial}{\partial\mu} h(x,\mu)dx$ converges uniformly on a closed interval J for $\mu \in J$ as well as that $I(\mu)$ converges for all μ. Then $I(\mu)$ is differentiable and we can differentiate under the integral sign. For details and proofs, see, e.g., Chapter 13 [128] or other standard analysis textbooks. Thus, we can derive the following

$$\begin{aligned} \frac{d}{d\mu} I(\mu) &= \frac{d}{d\mu}\left(\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\, dx\right) \\ &= \int_{-\infty}^{+\infty} \frac{dI}{d\mu}\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\right) dx \\ &= \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} \frac{(x-\mu)}{\sigma^2} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\, dx. \end{aligned} \tag{A.0.3}$$

Assuming that the same regularity assumptions are still satisfied for $\frac{d}{d\mu}I(\mu)$, one can differentiate again with respect to μ. Using the product rule, this yields

$$\begin{aligned}
\frac{d^2 I}{d\mu d\mu}(\mu) &= \frac{dI}{d\mu}\left(\int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma}\frac{(x-\mu)}{\sigma^2}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\,dx\right) \\
&= \int_{-\infty}^{+\infty} \frac{dI}{d\mu}\left(\frac{1}{\sqrt{2\pi}\sigma}\frac{(x-\mu)}{\sigma^2}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\right) dx \\
&= \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma}\left(\frac{(x-\mu)}{\sigma^2}\frac{(x-\mu)}{\sigma^2} - \frac{1}{\sigma^2}\right)\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) f(x)\,dx
\end{aligned} \tag{A.0.4}$$

We now perform a change of variables with $z = \frac{x-\mu}{\sqrt{2}\sigma} \iff x = \sqrt{2}\sigma z + \mu$, with $\frac{dz}{dx} = \frac{1}{\sqrt{2}\sigma} \iff dx = \sqrt{2}\sigma dz$, for all three functions. For a general discussion on integration by substitution, see e.g. [129]. This results in

$$I(\mu) = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{\pi}} \exp\left(-z^2\right) f(\sigma z + \mu)\,dz \tag{A.0.5}$$

$$\frac{d}{d\mu}I(\mu) = \int_{-\infty}^{+\infty} \frac{z}{\sigma}\frac{1}{\sqrt{\pi}} \exp\left(-z^2\right) f(\sigma z + \mu)\,dz \tag{A.0.6}$$

$$\frac{d^2}{d\mu d\mu}I(\mu) = \int_{-\infty}^{+\infty} \frac{(z^2-1)}{\sigma^2}\frac{1}{\sqrt{\pi}} \exp\left(-z^2\right) f(\sigma z + \mu)\,dz, \tag{A.0.7}$$

which is equivalent to Equation 2.2.17, 2.2.18 and 2.2.19. One can now easily apply Gauss-Hermite quadrature to approximate the above integrals, as discussed in Section 2.2.3. It is worth noting that both derivatives do not require the differentiation of f.

Appendix B

Discretization and Parameter Scales of the DEMF Model

Following Lindgren et al. in [49], the general form of Equation 2.2.31, assuming appropriate boundary conditions, non-negative smoothness parameters $(\alpha_t, \alpha_s, \alpha_e)$ and $L_s = (\gamma_s^2 - \Delta)$ is

$$\left(\gamma_t \frac{\partial}{\partial t} + L_s^{\alpha_s/2}\right)^{\alpha_t} u(\boldsymbol{s}, t) = \mathrm{d}\mathcal{E}_{Q,\gamma_e}(\boldsymbol{s}, t), \quad (\boldsymbol{s}, t) \in \mathcal{D} \times \mathbb{R}. \tag{B.0.1}$$

In Theorem 1 in [49], the authors show that for $\alpha_t \in \mathbb{N}$, Equation B.0.1 is equivalent to

$$\left(-\gamma_t^2 \frac{\partial^2}{\partial t^2} + L_s^{\alpha_s}\right)^{\alpha_t/2} u(\boldsymbol{s}, t) = \mathrm{d}\mathcal{E}_{Q,\gamma_e}(\boldsymbol{s}, t), \quad (\boldsymbol{s}, t) \in \mathcal{D} \times \mathbb{R}. \tag{B.0.2}$$

For a further theoretical discussion we refer to [49, 46] and instead consider the DEMF(1,2,1) model of interest in this thesis, i.e., $\alpha_t = 1, \alpha_s = 2, \alpha_e = 1$, and discuss various properties more relevant to its implementation and in particular motivate the arising tridiagonal block structure of the spatio-temporal prior precision matrix.

Finite Element Discretization

A finite element approach was chosen for the discretization of Equation 2.2.31 (or more generally Equation B.0.2) , as they are computationally efficient, easily account for unstructured spatial data locations and are well-defined on mani-

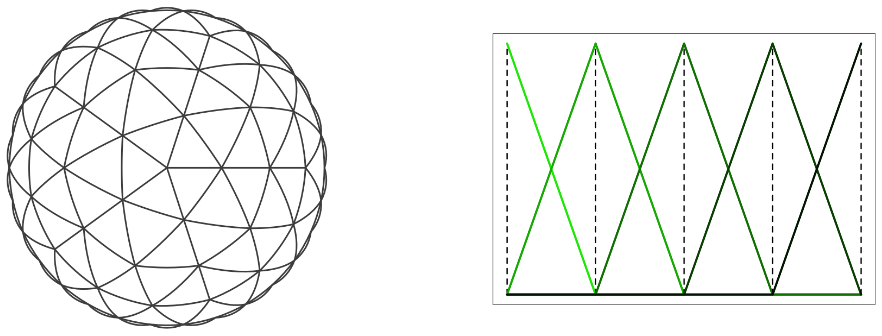


Figure B.1: *Left: Spatial mesh with 92 nodes on $\mathbb{S}^2$. Right: Temporal equidistant mesh with 5 nodes and piecewise linear hat functions.*

folds. A Kronecker product basis expansion of the form

$$u(\boldsymbol{s},t)=\sum_{i=1}^{n_s}\sum_{j=1}^{n_t} u_{ij}\,\psi_i(\boldsymbol{s})\phi_j(t), \tag{B.0.3}$$

is considered, where $\{\psi_i(\boldsymbol{s}) \mid i=1,\dots,n_s\}$ and $\{\phi_j(t) \mid j=1,\dots,n_t\}$ are sets of basis functions on a spatial domain $\mathcal{D}$ and time interval $T=[t_0,t_1]\subset\mathbb{R}$, respectively, with space-time coefficients u_{ij}. While there are various possible choices of spatial and temporal basis functions, we restrict ourselves to piecewise linear basis functions, see [49] for a comprehensive discussion. Therefore, let $\boldsymbol{\Psi}(\boldsymbol{s})=[\psi_1(\boldsymbol{s}),\dots,\psi_{n_s}(\boldsymbol{s})]$ be a set of spatial basis function on $\mathcal{D}$ with $\sum_{i=1}^{n_s}\psi_i(\boldsymbol{s})\equiv 1$. Similarly, let $\boldsymbol{\Phi}(t)=[\phi_1(t),\dots,\phi_{n_t}(t)]$ be a set of one-dimensional temporal basis function on $[t_0,t_1]$ with $\sum_{i=1}^{n_t}\phi_i(t)\equiv 1$.

We follow Theorem 2 and its necessary assumptions in [49], for the discretization of the differential operator. Using the above basis representation and $\alpha_t=1, \alpha_s=2$ and $\alpha_e=1$, this gives rise to the precision matrix $\boldsymbol{Q}_{\boldsymbol{u}}$ for the spatio-temporal random field with

$$\begin{aligned}\boldsymbol{Q}_{\boldsymbol{u}} =&\gamma_e^2\sum_{k=0}^{2\alpha_t}\gamma_t^k\boldsymbol{J}_{\alpha_t,k/2}\otimes\boldsymbol{K}_{\alpha_s(\alpha_t-k/2)+\alpha_e}\\ =&\gamma_e^2\left(\gamma_t^0\boldsymbol{J}_{\alpha_t,0}\otimes\boldsymbol{K}_{\alpha_s(\alpha_t-0)+\alpha_e}+\gamma_t^1\boldsymbol{J}_{\alpha_t,1/2}\otimes\boldsymbol{K}_{\alpha_s(\alpha_t-1/2)+\alpha_e}+\gamma_t^2\boldsymbol{J}_{\alpha_t,1}\otimes\boldsymbol{K}_{\alpha_s(\alpha_t-1)+\alpha_e}\right)\\ =&\gamma_e^2\left(\boldsymbol{J}_{1,0}\otimes\boldsymbol{K}_3+\gamma_t\boldsymbol{J}_{1,1/2}\otimes\boldsymbol{K}_2+\gamma_t^2\boldsymbol{J}_{1,1}\otimes\boldsymbol{K}_1\right).\end{aligned} \tag{B.0.4}$$

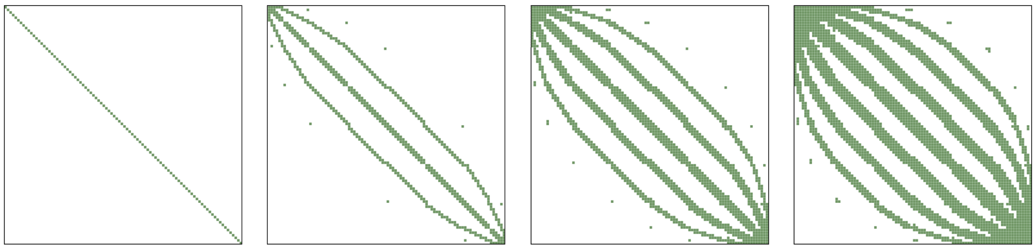


Figure B.2: *From left to right: Respective spatial precision matrices from a triangular mesh on a sphere using $n_s = 92$ nodes (a) Lumped mass matrix $\boldsymbol{C}$ (b) Stiffness matrix $\boldsymbol{G}$ or equivalently $\boldsymbol{K}_1$ (c) $\boldsymbol{G}^{(2)} = \boldsymbol{G}\boldsymbol{C}^{-1}\boldsymbol{G}$ or equivalently $\boldsymbol{K}_2$ (d) $\boldsymbol{G}^{(3)} = \boldsymbol{G}(\boldsymbol{C}^{-1}\boldsymbol{G})^{(2)}$ or equivalently $\boldsymbol{K}_3$.*

Here all finite element matrices $\boldsymbol{K}_i$ relate to the spatial component while the matrices $\boldsymbol{J}_{\alpha_t,i}$ are associated with the temporal component. Thus, each addend is separable, due to the separable basis functions B.0.3, but the overall sum is not.

In the following, we discuss the sparsity patterns of the matrices arising in Equation B.0.4. Exemplarily, we consider a space-time domain $\mathcal{D} \times T$ with $\mathcal{D} = \mathbb{S}^2$ and $T = [1, 5]$, see Figure B.1. The spatial domain is discretized using a triangulation with 92 mesh nodes on which the piecewise linear basis functions are defined. The temporal mesh assumes 5 equidistant time steps with a step size of one. The sparsity patterns of the different matrices that arise in the construction of the discretized precision operator are shown in Figure B.2–B.4. Starting with the spatial component, one can form the lumped mass matrix $\boldsymbol{C}$ with $C_{ii} = \langle \psi_i, 1 \rangle$ and zero otherwise, and stiffness matrix $\boldsymbol{G}$ with $G_{ij} = \langle \nabla\psi_i, \nabla\psi_j \rangle$ for $i, j = 1, \ldots, n_s$, see also Figure B.2. The discretization of the Whittle-Matérn operator $L_s^{\alpha_s} = (\gamma_s^s - \Delta)^{\alpha_s}$, as shown in [47], is then given by

$$\boldsymbol{K}_1 = \gamma_s^2 \boldsymbol{C} + \boldsymbol{G}, \tag{B.0.5}$$

$$\boldsymbol{K}_2 = \gamma_s^4 \boldsymbol{C} + 2\gamma_s^2 \boldsymbol{G} + \boldsymbol{G}^{(2)} \qquad \text{with} \ \ \boldsymbol{G}^{(2)} = \boldsymbol{G}\boldsymbol{C}^{-1}\boldsymbol{G}, \tag{B.0.6}$$

$$\boldsymbol{K}_3 = \gamma_s^6 \boldsymbol{C} + 3\gamma_s^4 \boldsymbol{G} + 3\gamma_s^2 \boldsymbol{G}^{(2)} + \boldsymbol{G}^{(3)} \qquad \text{with} \ \ \boldsymbol{G}^{(3)} = \boldsymbol{G}(\boldsymbol{C}^{-1}\boldsymbol{G})^{(2)}, \tag{B.0.7}$$

and more generally for $l \in \mathbb{N}, l > 2$,

$$\boldsymbol{K}_l = \boldsymbol{K}\boldsymbol{C}^{-1}\boldsymbol{K}_{l-2}\boldsymbol{C}^{-1}\boldsymbol{K}, \qquad \text{with } \boldsymbol{G}^{(k)} = \boldsymbol{G}(\boldsymbol{C}^{-1}\boldsymbol{G})^{(k-1)}. \tag{B.0.8}$$

where l depends on the exponent α_s. The discretization of the temporal matrices

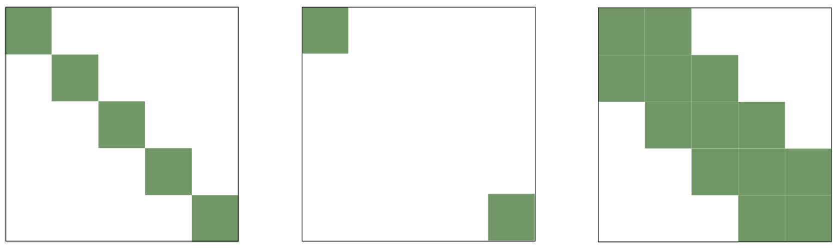


Figure B.3: *From left to right: Respective temporal precision matrices using 5 equidistant time steps (a) Lumped one-dimensional mass matrix $\boldsymbol{J}_{1,0}$ (b) Boundary matrix $\boldsymbol{J}_{1,1/2}$ (c) One-dimensional stiffness matrix $\boldsymbol{J}_{1,1}$*

$\boldsymbol{J}_{\alpha_t,k/2}$ for odd k, i.e., $k = 1, 3, \dots, 2\alpha_t - 1$ is defined as

$$[\boldsymbol{J}_{\alpha_t,k/2}]_{ij} = \begin{cases} \text{boundary correction term}, & \text{if } (i,j) \in \{(1,1),(n_t,n_t)\} \\ 0, & \text{else} \end{cases} \tag{B.0.9}$$

which imposes temporal Neumann boundary conditions. For even k, i.e., $k = 0, 2, \dots, 2\alpha_t$ this means

$$[\boldsymbol{J}_{\alpha_t,k/2}]_{ij} = \langle(-\Delta)^{k/2}\phi_i, (-\Delta)^{k/2}\phi_j\rangle. \tag{B.0.10}$$

Thus, for $\alpha_t = 1$, we consider $k = 0, 1, 2$. For $k = 0$ we obtain a one-dimensional mass matrix, for which we perform mass lumping as before, i.e. $[\boldsymbol{J}_{\alpha_t,0}]_{ii} = \langle\phi_i, 1\rangle$. For $k = 1$, one obtains a zero matrix with Neumann boundary conditions, i.e., in $[\boldsymbol{J}_{\alpha_t,1/2}]_{11} = [\boldsymbol{J}_{\alpha_t,1/2}]_{n_t n_t} = 1/(2h)$, where h is the distance between mesh nodes, otherwise $[\boldsymbol{J}_{\alpha_t,1/2}]_{ij} = 0$. For $k = 2$, we obtain a one-dimensional stiffness matrix $[\boldsymbol{J}_{\alpha_t,1}]_{ij} = \langle\nabla\phi_i, \nabla\phi_j\rangle$ for $i, j = 1, \dots, n_t$, see also Figure B.3. The block tridiagonal structure is induced by the Kronecker product between the tridiagonal temporal stiffness matrix $\boldsymbol{J}_{1,1}$ and $\boldsymbol{K}_1$, as shown in Figure B.4. From this it also becomes clear, that if the temporal process is discretized using a higher-order scheme that, e.g., induces a pentagonal structure this would result in a block pentagonal precision matrix $\boldsymbol{Q}_u$. The additional non-zeros in the diagonal blocks are induced by the Kronecker product between the diagonal matrix $\boldsymbol{J}_{1,0}$ and $\boldsymbol{K}_3$.

Conditioning on Data

Let $\mathbb{E}(\boldsymbol{y}) = g^{-1}(\boldsymbol{\eta})$, where $\mathbb{E}(\cdot)$ denotes the expected value, $g(\cdot)$ a link function and $\boldsymbol{\eta}$ the linear predictor. Let $\boldsymbol{\eta} = \tilde{\boldsymbol{A}}\boldsymbol{u} + \boldsymbol{Z}\boldsymbol{\beta}$, where we assume that $\boldsymbol{u}$ denotes

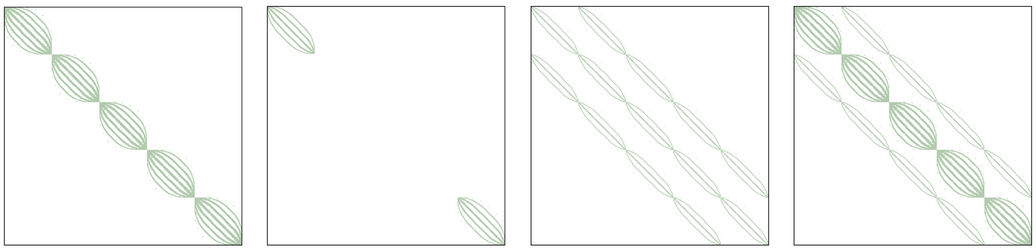


Figure B.4: *From left to right: The respective sparsity patterns of (a)* $\boldsymbol{J}_{1,0} \otimes \boldsymbol{K}_3$ *(b)* $\boldsymbol{J}_{1,1/2} \otimes \boldsymbol{K}_2$ *(c)* $\boldsymbol{J}_{1,1} \otimes \boldsymbol{K}_1$ *(d)* $\boldsymbol{Q}_u = \boldsymbol{J}_{1,0} \otimes \boldsymbol{K}_3 + \boldsymbol{J}_{1,1/2} \otimes \boldsymbol{K}_2 + \boldsymbol{J}_{1,1} \otimes \boldsymbol{K}_1$

the spatio-temporal random effects $\boldsymbol{u} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{Q}_u^{-1})$ as defined previously. The matrix $\boldsymbol{Z}$ is assumed to contain the covariates related to the fixed effects $\boldsymbol{\beta}$. The projection matrix $\tilde{\boldsymbol{A}} \in \mathbb{R}^{m \times (n_s n_t)}$ contains the coefficients of the finite element representation of the spatio-temporal locations of the observations. Each observation y_i is associated with a time t and a spatial location s_i. We represent this in the basis formulation B.0.3 and store the arising coefficients in $\tilde{\boldsymbol{A}}$. Due to the mesh triangulation of the spatial domain each location s_i is represented by either one (if the mesh node and the observation location coincide), two (if s_i is on an edge) or three (for all other cases) spatial basis functions. In all cases the sum of their coefficients is 1. We assume that each measurement is taken at a given time step and thus associated with exactly one of the n_t time steps. Exemplary, we show the sparsity pattern of a projection matrix $\tilde{\boldsymbol{A}}$ for 200 randomly sampled observations within the previously described spatio-temporal domain. We additionally assume that the model has $n_b = 4$ fixed effects and $\boldsymbol{Z}$ is a dense matrix. Then we obtain the following sparsity pattern for projection matrix $\boldsymbol{A} = [\tilde{\boldsymbol{A}}, \boldsymbol{Z}]$, see Figure B.5. If we consider the priors of the fixed effects to be independent, the prior precision matrix $\boldsymbol{Q}_\beta$ becomes a diagonal matrix, which we combine to $\boldsymbol{Q}_x = \text{blockdiag}(\boldsymbol{Q}_u, \boldsymbol{Q}_\beta)$. Finally, we construct the conditional posterior precision matrix $\boldsymbol{Q}_{x|y} = \boldsymbol{Q}_x + \boldsymbol{A}^T \boldsymbol{D} \boldsymbol{A}$, where $\boldsymbol{D}$ is a diagonal matrix. We observe that the sparsity pattern of $\boldsymbol{Q}_{x|y}$ restricted to the spatio-temporal part coincides with the sparsity pattern of $\boldsymbol{Q}_u$, i.e. no additional non-zeros are introduced by conditioning on the observations due to the finite element basis representation. The covariate matrix $\boldsymbol{Z}$ introduces additional non-zeros. This number does, however, not depend on the number of observations but only on the number of fixed effects as shown in Figure B.5. This results in a block tridiagonal arrowhead sparsity structure in $\boldsymbol{Q}_{x|y}$.

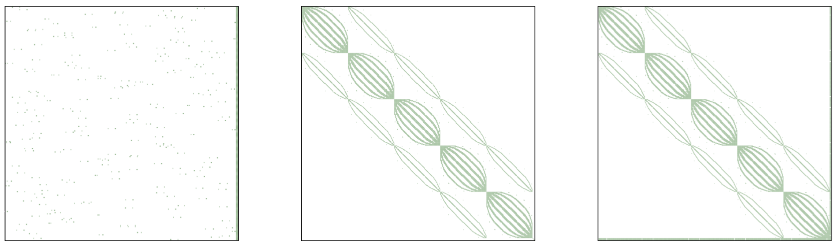


Figure B.5: *From left to right: The respective sparsity patterns of (a) projection matrix $\boldsymbol{A} \in \mathbb{R}^{m\times(n_s n_t)}$ with $m = 200, n_s = 92, n_t = 5$ (b) $\boldsymbol{Q_x} \in \mathbb{R}^{n\times n}$ with $n = n_s n_t + n_b$, where $n_b = 4$ (c) $\boldsymbol{Q_{x|y}} = \boldsymbol{Q_x} + \boldsymbol{A}^T \boldsymbol{DA}$. One can see the arising block tridiagonal structure, where each diagonal block refers to a discretized spatial domain at a different time step.*

Model Scale Parameters

The scaling parameters $(\gamma_s, \gamma_t, \gamma_e)$ naturally arise as part of the SPDE, Equation B.0.2, and are also convenient from an implementation perspective. They enter the LGM as part of the hyperparameters, where they do not have an intuitive interpretation. For increased model interpretability, one considers instead (r_s, r_t, σ_{st})[1]. Here, r_s represents the spatial correlation range, giving a correlation in space of approximately 0.13 while keeping time fixed. The parameter r_t describes the temporal correlation range which, for a separable model, is defined as the spatial range. In the non-separable case, the temporal correlation will also be dependent on the spatial range. And finally, σ_{st} which describes the marginal standard deviation of the spatial-temporal field. The conversion between the two parameter scales is presented below, where the topology of the domain $\mathcal{D}$ has to be taken into account. We first consider $\mathcal{D} \subset \mathbb{R}^d$, where we have that

$$\begin{aligned} &\alpha = \alpha_e + \alpha_s(\alpha_t - 0.5), \quad \nu_s = \alpha - 1, \\ &C_{\mathbb{R},\alpha_t} = \frac{\Gamma(\alpha_t - 1/2)}{\Gamma(\alpha_t)(4\pi)^{1/2}}, \quad C_{\mathbb{R}^d,\alpha} = \frac{\Gamma(\alpha - d/2)}{\Gamma(\alpha)(4\pi)^{d/2}}, \end{aligned} \tag{B.0.11}$$

[1]which also corresponds to the way the DEMF models are implemented in the INLAspacetime package.

and where $\Gamma(a)$ is the Gamma function. The parameters are typically considered in log-scale. Given $(\gamma_s, \gamma_t, \gamma_e)$, we compute (r_s, r_t, σ_{st}) as

$$\begin{aligned}
r_s = \frac{\sqrt{8\nu_s}}{\gamma_s} &\iff \log(r_s) = \frac{1}{2}\log(8\nu_s) - \log(\gamma_s), \\
r_t = \frac{\gamma_t\sqrt{8(\alpha_t - \frac{1}{2})}}{\gamma_s^{\alpha_s}} &\iff \log(r_t) = \log(\gamma_t) + \frac{1}{2}\log(8(\alpha_t - \frac{1}{2})) - \alpha_s\log(\gamma_s), \\
\sigma_{st} = \frac{\sqrt{C_{\mathbb{R}^d,\alpha}C_{\mathbb{R},\alpha_t}}}{\gamma_e\sqrt{\gamma_t}\gamma_s^{\alpha-d/2}} &\iff \log(\sigma_{st}) = \frac{1}{2}\log(C_{\mathbb{R}^d,\alpha}) + \frac{1}{2}\log(C_{\mathbb{R},\alpha_t}) - \log(\gamma_e) \\
&\qquad - \frac{1}{2}\log(\gamma_t) - (\alpha - \frac{d}{2})\log(\gamma_s).
\end{aligned} \tag{B.0.12}$$

Conversely, given (r_s, r_t, σ_{st}) we can rearrange the above to

$$\begin{aligned}
\gamma_s = \frac{\sqrt{8\nu_s}}{r_s} &\iff \log(\gamma_s) = \frac{1}{2}\log(8\nu_s) - \log(r_s), \\
\gamma_t = \frac{r_t\gamma_s^{\alpha_s}}{\sqrt{8(\alpha_t - 1/2)}} &\iff \log(\gamma_t) = \log(r_t) + \alpha_s\log(\gamma_s) - 0.5\log(8(\alpha_t - \frac{1}{2})), \\
\gamma_e = \frac{\sqrt{C_{\mathbb{R}^d,\alpha}C_{\mathbb{R},\alpha_t}}}{\sigma_{st}\sqrt{\gamma_t}\gamma_s^{\alpha-d/2}} &\iff \log(\gamma_e) = \frac{1}{2}\log(C_{\mathbb{R}^d,\alpha}) + \frac{1}{2}\log(C_{\mathbb{R},\alpha_t}) - \log(\sigma_{st}) \\
&\qquad - \frac{1}{2}\log(\gamma_t) - (\alpha - \frac{d}{2})\log(\gamma_s).
\end{aligned} \tag{B.0.13}$$

If the spatial domain is a compact manifold, e.g. a sphere, that is $\mathcal{D} = \mathbb{S}^d$, we have a change in the constants involving γ_e and σ_{st}, respectively. The constant $C_{\mathbb{R}^d,\alpha}/\gamma_s^{2\alpha-d}$ is replaced by $C_{\mathbb{S},\alpha}(\gamma_s)$.

$$C_{\mathbb{R},\alpha_t} = \frac{\Gamma(\alpha_t - 1/2)}{\Gamma(\alpha_t)(4\pi)^{1/2}}, \quad C_{\mathbb{S},\alpha}(\gamma_s) = \sum_{k=0}^{\infty}\frac{2k+1}{4\pi(\gamma^2 + k(k+1))^\alpha} \tag{B.0.14}$$

Given $(\gamma_s, \gamma_t, \gamma_e)$, this results in

$$\sigma_{st} = \frac{\sqrt{C_{\mathbb{R},\alpha_t} C_{\mathbb{S},\alpha}(\gamma_s)}}{\gamma_e \sqrt{\gamma_t}} \iff \log(\sigma_{st}) = \frac{1}{2}\log(C_{\mathbb{R},\alpha_t}) + \frac{1}{2}\log(C_{\mathbb{S},\alpha}(\gamma_s)) - \log(\gamma_e) - \frac{1}{2}\log(\gamma_t). \tag{B.0.15}$$

Or alternatively, given (r_s, r_t, σ_{st}), we can compute

$$\gamma_e = \frac{\sqrt{C_{\mathbb{R},\alpha_t} C_{\mathbb{S},\alpha}(\gamma_s)}}{\sigma_{st} \sqrt{\gamma_t}} \iff \log(\gamma_e) = \frac{1}{2}\log(C_{\mathbb{R},\alpha_t}) + \frac{1}{2}\log(C_{\mathbb{S},\alpha}(\sigma_{st})) - \log(\gamma_e) - \frac{1}{2}\log(\gamma_t). \tag{B.0.16}$$

Appendix C

Marginal Variances and Matrix-Matrix Multiplication

Let $\boldsymbol{Q}_{x|y}$ be the precision matrix of the latent parameters conditioned on the data, as defined in Equation 2.2.7, for a fixed parameter configuration of $\boldsymbol{\theta}$. We denote its covariance matrix by $\boldsymbol{\Sigma}_{x|y} = \boldsymbol{Q}_{x|y}^{-1}$. The marginal variance of each latent parameter x_i is $(\boldsymbol{\Sigma}_{x|y})_{ii}$. Under the assumption that the data is normally distributed, the expected value of the observations is defined through the linear predictor $\boldsymbol{E}(\boldsymbol{y}) = \boldsymbol{A}\boldsymbol{x}$. This implies that the covariance matrix of the observations $\text{Cov}(\boldsymbol{y}) = \boldsymbol{A}\,\boldsymbol{\Sigma}_{x|y}\,\boldsymbol{A}^T$ and their marginal variances σ_i^2 are

$$\sigma_i^2 = (\boldsymbol{A}\,\boldsymbol{\Sigma}_{x|y}\,\boldsymbol{A}^T)_{ii}. \tag{C.0.1}$$

While this is relatively straight-forward to derive from a theoretical perspective, from a computational perspective multiple issues emerge. First, the full inverse $\boldsymbol{\Sigma}_{x|y}$ is never computed as extensively discussed in Chapter 3. This is due to the fact that we do not require all entries and the computational cost as well as memory requirements are prohibitive, especially when considering a multitude of $\boldsymbol{\theta}$ configurations. Second, the matrix $\boldsymbol{A}\,\boldsymbol{\Sigma}_{x|y}\,\boldsymbol{A}^T$ arising in Equation C.0.1 is dense and therefore, computationally expensive to compute and store, especially since its final dimensions are equal to the number of observations and therefore potentially very large. To address the challenges that would arise in the case of standard matrix multiplication, we bring forward the following considerations.

Foremost we are interested in the marginal variances σ_i^2 and thus only require the diagonal values of Equation C.0.1, rewriting them as matrix sums one obtains

$$(\boldsymbol{A}\,\boldsymbol{\Sigma})_{ik} = \sum_{j=1}^{n} A_{ij}\Sigma_{jk} \tag{C.0.2}$$

$$(\boldsymbol{A}\,\boldsymbol{\Sigma}_{\boldsymbol{x}|\boldsymbol{y}}\,\boldsymbol{A}^T)_{ii} = \sum_{k=1}^{n}(\boldsymbol{A}\,\boldsymbol{\Sigma})_{ik}A^T_{ki} = \sum_{\substack{k=1\\A_{ik}\neq 0}}^{n}\left(\sum_{\substack{j=1\\A_{ij}\neq 0}}^{n} A_{ij}\Sigma_{jk}\right)A_{ik} \quad \text{for all } i = 1,\ldots,m. \tag{C.0.3}$$

This implies that for every entry of interest, we only require the set of entries of $\boldsymbol{\Sigma}$ for whose indices (j,k), there exists $i = 1,\ldots,m$ with $\boldsymbol{A}_{ij} \neq 0$ and $\boldsymbol{A}_{ik} \neq 0$. This dramatically reduces the amount of required computations as they scale with the number of non-zeros in A, which is typically $O(m)$ where m is the number of observations. As $\boldsymbol{A}$ is a projection matrix linking the observations through the linear predictor to the latent variables, we have that the required indices (j,k) are contained within in the original sparsity pattern of $\boldsymbol{Q}_{\boldsymbol{x}|\boldsymbol{y}}$.

Compute Resource Acknowledgments

We gratefully acknowledge the great scientific support and HPC resources provided by the Erlangen National High Performance Computing Center(NHR@FAU) of the Friedrich-Alexander-Universität Erlangen Nürnberg (FAU) under the NHR project 286745. NHR funding is provided by federal and Bavarian state authorities. NHR@FAU hardware is partially funded by the German Research Foundation (DFG) – 440719683. Thank you for all the constructive support, the prompt replies and access to the hardware resources.

Bibliography

[1] Håvard Rue, Sara Martino, and Nicolas Chopin. Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society: Series b (statistical methodology)*, 71(2):319–392, 2009. 10.1111/j.1467-9868.2008.00700.x.

[2] Håvard Rue, Andrea Riebler, Sigrunn H Sørbye, Janine B Illian, Daniel P Simpson, and Finn K Lindgren. Bayesian computing with INLA: a review. *Annual Review of Statistics and Its Application*, 4:395–421, 2017. 10.1146/annurev-statistics-060116-054045.

[3] Thomas Opitz. Latent Gaussian modeling and INLA: A review with focus on space-time applications. *Journal de la Société Française de Statistique*, 158(3):62–85, 2017. hal.archives-ouvertes.fr/hal-01394974.

[4] H. Bakka, H. Rue, G. A. Fuglstad, A. Riebler, D. Bolin, J. Illian, E. Krainski, D. Simpson, and F. Lindgren. Spatial modelling with R-INLA: A review. *WIREs Computational Statistics*, 10:e1443(6), 2018. (Invited extended review), `10.1002/wics.1443`.

[5] Maeregu Woldeyes Arisido, Carlo Gaetan, Davide Zanchettin, and Angelo Rubino. A Bayesian hierarchical approach for spatial analysis of climate model bias in multi-model ensembles. *Stochastic Environmental Research and Risk Assessment*, 31(10):2645–2657, 2017. 10.1007/s00477-017-1383-2.

[6] Win Wah, Susannah Ahern, and Arul Earnest. A systematic review of Bayesian spatial–temporal models on cancer incidence and mortality. *International Journal of Public Health*, 65(5):673–682, 2020. 10.1007/s00038-020-01384-5.

[7] Yuzi Zhang, Howard H Chang, A Danielle Iuliano, and Carrie Reed. Application of Bayesian spatial-temporal models for estimating unrecognized COVID-19 deaths in the United States. *Spatial Statistics*, page 100584, 2022. 10.1016/j.spasta.2021.100584.

[8] Judith Lehnert, Christoph Kolbitsch, Gerd Wübbeler, Amedeo Chiribiri, Tobias Schaeffter, and Clemens Elster. Large-scale Bayesian spatial-temporal regression with application to cardiac MR-perfusion imaging. *SIAM Journal on Imaging Sciences*, 12(4):2035–2062, 2019. 10.1137/19M1246274.

[9] Amanda F Mejia, Yu Yue, David Bolin, Finn Lindgren, and Martin A Lindquist. A Bayesian general linear modeling approach to cortical surface fMRI data analysis. *Journal of the American Statistical Association*, 115(530):501–520, 2020. 10.1080/01621459.2019.1611582.

[10] Brice Batomen, Hyacinth Irving, Mabel Carabali, Marilia Sá Carvalho, Erica Di Ruggiero, and Patrick Brown. Vulnerable road-user deaths in Brazil: a Bayesian hierarchical model for spatial-temporal analysis. *International Journal of Injury Control and Safety Promotion*, pages 1–9, 2020. 10.1080/17457300.2020.1818788.

[11] Yimeng Duan, Shen Zhang, and Zhuoran Yu. Applying Bayesian spatio-temporal models to demand analysis of shared bicycle. *Physica A: Statistical Mechanics and its Applications*, 583:126296, 2021. 10.1016/j.physa.2021.126296.

[12] Chiara Forlani, Samir Bhatt, Michela Cameletti, Elias Krainski, and Marta Blangiardo. A joint Bayesian space–time model to integrate spatially misaligned air pollution data in R-INLA. *Environmetrics*, 31(8):e2644, 2020. 10.1002/env.2644.

[13] Denis D Patterson, Simon A Levin, Carla Staver, and Jonathan D Touboul. Probabilistic foundations of spatial mean-field models in ecology and applications. *SIAM journal on Applied Dynamical Systems*, 19(4):2682–2719, 2020. 10.1137/19M1298329.

[14] Cornelius Senf, Dirk Pflugmacher, Marco Heurich, and Tobias Krueger. A Bayesian hierarchical model for estimating spatial and temporal variation in vegetation phenology from Landsat time series. *Remote Sensing of Environment*, 194:155–160, 2017. 10.1016/j.rse.2017.03.020.

[15] Gregorio A Millett, Austin T Jones, David Benkeser, Stefan Baral, Laina Mercer, Chris Beyrer, Brian Honermann, Elise Lankiewicz, Leandro Mena, Jeffrey S Crowley, et al. Assessing differential impacts of COVID-19 on black communities. *Annals of Epidemiology*, 47:37–44, 2020. 10.1016/j.annepidem.2020.05.00.

[16] Local Burden of Disease Educational Attainment Collaborators. Mapping disparities in education across low-and middle-income countries. *Nature*, 577(7789):235–238, 2020. 10.1038/s41586-019-1872-1.

[17] Edwin T Jaynes. *Probability theory: The Logic of Science*. Cambridge university press, 2003. 10.1007/BF02985800.

[18] Rens van de Schoot, Sarah Depaoli, Ruth King, Bianca Kramer, Kaspar Märtens, Mahlet G Tadesse, Marina Vannucci, Andrew Gelman, Duco Veen, Joukje Willemsen, et al. Bayesian statistics and modelling. *Nature Reviews Methods Primers*, 1(1):1, 2021. 10.1038/ s43586-020-00001-2.

[19] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. 10.1063/1.1699114.

[20] Christian Robert and George Casella. A Short History of Markov Chain Monte Carlo: Subjective Recollections from Incomplete Data. *Statistical Science*, 26(1):102–115, 2011. https://www.jstor.org/stable/23059158.

[21] Wilfred Keith Hastings. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika*, pages 97–109, 1970. 10.1063/1.1699114.

[22] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid Monte Carlo. *Physics Letters B*, 195(2):216–222, 1987. 10.1016/0370-2693(87)91197-X.

[23] Mark Girolami and Ben Calderhead. Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 73(2):123–214, 2011. 10.1111/j.1467-9868.2010.00765.xX.

[24] Arnaud Doucet, Adam M Johansen, et al. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12 (656-704):3, 2009.

[25] James Carpenter, Peter Clifford, and Paul Fearnhead. Improved particle filter for nonlinear problems. *IEE Proceedings-Radar, Sonar and Navigation*, 146(1):2–7, 1999. 10.1049/ip-rsn:19990255 .

[26] Paul Fearnhead, Omiros Papaspiliopoulos, and Gareth O Roberts. Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 70(4):755–777, 2008. 10.1111/j.1467-9868.2008.00661.x.

[27] Dani Gamerman and Hedibert F Lopes. *Markov chain Monte Carlo: stochastic simulation for Bayesian inference*. CRC Press, 2006. 10.1201/9781482296426.

[28] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC Press, 2011. 10.1201/b10905.

[29] David Lunn, David Spiegelhalter, Andrew Thomas, and Nicky Best. The BUGS project: Evolution, critique and future directions. *Statistics in Medicine*, 28(25):3049–3067, 2009. 10.1002/sim.3680.

[30] Martyn Plummer et al. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, volume 124, pages 1–10. Vienna, Austria, 2003.

[31] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2: e55, 2016. https://peerj.com/articles/cs-55/.

[32] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76, 2017. 10.18637/jss.v076.i01.

[33] C Bishop. *Pattern recognition and machine learning*, volume 2. Springer, 2006. 10.1117/1.2819119.

[34] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37:183–233, 1999. 10.1023/A:1007665907178.

[35] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017. 10.1080/01621459.2017.1285773.

[36] Michael L Stein. *Interpolation of Spatial Data: Some Theory for Kriging*. Springer Science & Business Media, 2012. 10.1007/978-1-4612-1494-6.

[37] Jie Wang. An intuitive tutorial to Gaussian processes regression. *Computing in Science & Engineering*, 2023. 10.1109/MCSE.2023.3342149.

[38] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian Processes for Machine Learning*, volume 2. MIT Press Cambridge, MA, 2006. 10.5555/1162254.

[39] Christian P Robert, George Casella, and George Casella. *Monte Carlo Statistical Methods*, volume 2. Springer, 1999. 10.1007/978-1-4757-4145-2.

[40] Jean-Paul Chiles and Pierre Delfiner. *Geostatistics: Modeling Spatial Uncertainty*, volume 713. John Wiley & Sons, 2012. 10.1002/9781118136188.

[41] Joaquin Quinonero-Candela and Carl Edward Rasmussen. A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959, 2005. www.jmlr.org/papers/v6/quinonero-candela05a.

[42] Wolfgang Hackbusch et al. *Hierarchical matrices: algorithms and analysis*, volume 49. Springer, 2015. 10.1007/978-3-662-47324-5.

[43] Alexander Litvinenko, Ying Sun, Marc G Genton, and David E Keyes. Likelihood approximation with hierarchical matrices for large spatial datasets. *Computational Statistics & Data Analysis*, 137:115–132, 2019. 10.1016/j.csda.2019.02.002.

[44] Helmut Harbrecht, Michael Multerer, Olaf Schenk, and Ch Schwab. Multiresolution kernel matrix algebra. *Numerische Mathematik*, 156(3):1085–1114, 2024. 10.1007/s00211-024-01409-8.

[45] Noel Cressie and Christopher K. Wikle. *Statistics for Spatio-Temporal data*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., Hoboken, NJ, 2011.

[46] Finn Lindgren, David Bolin, and Håvard Rue. The SPDE approach for Gaussian and non-Gaussian fields: 10 years and still running. *Spatial Statistics*, 50:100599, 2022. 10.1016/j.spasta.2022.100599.

[47] Finn Lindgren, Håvard Rue, and Johan Lindström. An explicit link between Gaussian fields and Gaussian Markov random fields: the stochastic partial differential equation approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(4):423–498, 2011. 10.1111/j.1467-9868.2011.00777.x.

[48] F. Sigrist, Künsch H. R., and W. A. Stahel. Stochastic partial differential equation based modelling of large spacetime data sets. *Journal of the Royal Statistical Society, Series B*, 77(1):3–33, 2015. 10.1111/rssb.12061.

[49] Finn Lindgren, Haakon Bakka, David Bolin, Elias Krainski, and Håvard Rue. A diffusion-based spatio-temporal extension of Gaussian Matérn fields. *SORT*, 48(1):3–66, 2024. 10.57645/20.8080.02.13.

[50] Havard Rue and Leonhard Held. *Gaussian Markov random fields: Theory and Applications*. CRC Press, 2005. 10.1201/9780203492024.

[51] Thiago G Martins, Daniel Simpson, Finn Lindgren, and Håvard Rue. Bayesian computing with INLA: new features. *Computational Statistics & Data Analysis*, 67:68–83, 2013. 10.1016/j.csda.2013.04.014.

[52] Janet Van Niekerk, Elias Krainski, Denis Rustand, and Håvard Rue. A new avenue for Bayesian inference with INLA. *Computational Statistics & Data Analysis*, 2023. 10.1016/j.csda.2023.107692.

[53] Janet van Niekerk and Haavard Rue. Low-rank variational bayes correction to the laplace method. *Journal of Machine Learning Research*, 25(62): 1–25, 2024. https://www.jmlr.org/papers/v25/21-1405.html.

[54] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Science & Business Media, 2006. 10.1007/978-0-387-40065-5.

[55] Esmail Abdul Fattah, Janet Van Niekerk, and Håvard Rue. Smart Gradient - An adaptive technique for improving gradient estimation. *Foundations of Data Science*, 4(1):123–136, 2022. 10.3934/fods.2021037.

[56] Pierre Simon Laplace. Memoir on the probability of the causes of events. *Statistical science*, 1(3):364–378, 1986. 10.1214/ss/1177013621.

[57] Janet van Niekerk, Haakon Bakka, Haavard Rue, and Olaf Schenk. New frontiers in Bayesian modeling using the INLA package in R. *Journal of Statistical Software*, 100(1), 2022. 10.18637/jss.v100.i02.

[58] Arnold Zellner. Optimal information processing and Bayes's theorem. *The American Statistician*, 42(4):278–280, 1988. 10.2307/2685143.

[59] Nigel Waters. Tobler's First Law of Geography. *International Encyclopedia of Geography: People, the Earth, Environment and Technology*, pages 1–15, 2016. 10.1002/9781118786352.wbieg1011.pub2.

[60] Peter Whittle. On stationary processes in the plane. *Biometrika*, pages 434–449, 1954. 10.1093/biomet/41.3-4.434.

[61] JN Reddy. An Introduction to the Finite Element Method. *Journal of Pressure Vessel Technology*, 1989. 10.1115/1.3265687.

[62] Richard H Jones and Yiming Zhang. Models for continuous stationary space-time processes. In *Modelling Longitudinal and Spatially Correlated data*, pages 289–298. Springer, 1997. 10.1007/978-1-4612-0699-6_25.

[63] Michael L Stein. Space–time covariance functions. *Journal of the American Statistical Association*, 100(469):310–321, 2005. 10.1198/016214504000000854.

[64] Lisa Gaedke-Merzhäuser, Janet van Niekerk, Olaf Schenk, and Håvard Rue. Parallelized integrated nested Laplace approximations for fast Bayesian inference. *Statistics and Computing*, 33(25), 2023. 10.1007/s11222-022-10192-1.

[65] Lisa Gaedke-Merzhäuser, Elias Krainski, Radim Janalik, Håvard Rue, and Olaf Schenk. Integrated Nested Laplace Approximations for Large-Scale Spatio-Temporal Bayesian Modeling. *SIAM Journal on Scientific Computing (accepted)*, 2024. abs/2303.15254.

[66] Mathias Jacquelin, Lin Lin, and Chao Yang. PSelInv – a distributed memory parallel algorithm for selected inversion: The non-symmetric case. *Parallel Computing*, 74:84–98, 2018. ISSN 0167-8191. 10.1016/j.parco.2017.11.009.

[67] Lin Lin, Chao Yang, Jianfeng Lu, Lexing Ying, and Weinan E. A fast parallel algorithm for selected inversion of structured sparse matrices with application to 2d electronic structure calculations. *SIAM Journal on Scientific Computing*, 33(3):1329–1351, 2011. 10.1137/09077432X.

[68] Kazuhiro Takahashi. Formation of sparse bus impedance matrix and its application to short circuit study. In *Proceedings PICA Conference, June, 1973*, 1973.

[69] TOP500. TOP500 Supercomputer Sites. `https://www.top500.org/`. Accessed: Feb 20, 2024.

[70] H Rue, S Martino, and N Chopin. *INLA: Approximate Bayesian Inference using Integrated Nested Laplace Approximations*, 2011. `www.r-inla.org` (version 22.02.16-2).

[71] Oscar Rodríguez de Rivera, Marta Blangiardo, Antonio López-Quílez, and Ignacio Martín-Sanz. Species distribution modelling through Bayesian hierarchical approach. *Theoretical Ecology*, 12(1):49–59, 2019. 10.1007/s12080-018-0387-y.

[72] Ning Lu, Shunlin Liang, Guanghui Huang, Jun Qin, Ling Yao, Dongdong Wang, and Kun Yang. Hierarchical Bayesian space-time estimation of monthly maximum and minimum surface air temperature. *Remote Sensing of Environment*, 211:48–58, 2018. 10.1016/j.rse.2018.04.006.

[73] Samir Bhatt, DJ Weiss, E Cameron, D Bisanzio, B Mappin, U Dalrymple, KE Battle, CL Moyes, A Henry, PA Eckhoff, et al. The effect of malaria control on Plasmodium falciparum in Africa between 2000 and 2015. *Nature*, 526(7572):207–211, 2015. 10.1038/nature15535.

[74] Garyfallos Konstantinoudis, Tullia Padellini, James Bennett, Bethan Davies, Majid Ezzati, and Marta Blangiardo. Long-term exposure to air-pollution and COVID-19 mortality in England: a hierarchical spatial analysis. *Environment International*, 146:106316, 2021. 10.1016/j.envint.2020.106316.

[75] Konrad P Mielke, Tom Claassen, Michela Busana, Tom Heskes, Mark AJ Huijbregts, Kees Koffijberg, and Aafke M Schipper. Disentangling drivers of spatial autocorrelation in species distribution models. *Ecography*, 43 (12):1741–1751, 2020. 10.1111/ecog.05134.

[76] Marta Coll, M Grazia Pennino, Jeroen Steenbeek, Jordi Solé, and José M Bellido. Predicting marine species distributions: complementarity of food-web and Bayesian hierarchical modelling approaches. *Ecological Modelling*, 405:86–101, 2019. 10.1016/j.ecolmodel.2019.05.005.

[77] Joaquín Martínez-Minaya, Michela Cameletti, David Conesa, and Maria Grazia Pennino. Species distribution modeling: a statistical review with focus in spatio-temporal issues. *Stochastic Environmental Research and Risk Assessment*, 32(11):3227–3244, 2018. 10.1007/s00477-018-1548-7.

[78] Nick JB Isaac, Marta A Jarzyna, Petr Keil, Lea I Dambly, Philipp H Boersch-Supan, Ella Browning, Stephen N Freeman, Nick Golding, Gurutzeta Guillera-Arroita, Peter A Henrys, et al. Data integration for large-scale models of species distributions. *Trends in ecology & evolution*, 35(1):56–67, 2020. 10.1016/j.tree.2019.08.006.

[79] David Lindenmayer, Chris Taylor, and Wade Blanchard. Empirical analyses of the factors influencing fire severity in southeastern Australia. *Ecosphere*, 12(8):e03721, 2021. 10.1002/ecs2.3721.

[80] François Pimont, Hélène Fargeon, Thomas Opitz, Julien Ruffault, Renaud Barbero, Nicolas Martin-StPaul, Eric Rigolot, Miguel Rivière, and Jean-Luc Dupuy. Prediction of regional wildfire activity in the probabilistic Bayesian framework of Firelihood. *Ecological applications*, 31(5):e02316, 2021. 10.1002/eap.2316.

[81] GASJ Pinto, F Rousseu, M Niklasson, and I Drobyshev. Effects of human-related and biotic landscape features on the occurrence and size of modern forest fires in Sweden. *Agricultural and Forest Meteorology*, 291: 108084, 2020. 10.1016/j.agrformet.2020.108084.

[82] Roberto Lillini, Andrea Tittarelli, Martina Bertoldi, David Ritchie, Alexander Katalinic, Ron Pritzkuleit, Guy Launoy, Ludivine Launay, Elodie Guillaume, Tina Žagar, et al. Water and Soil Pollution: Ecological Environmental Study Methodologies Useful for Public Health Projects. A Literature Review. *Reviews of Environmental Contamination and Toxicology Volume 256*, pages 179–214, 2021. 10.1007/398_2020_58.

[83] Shreosi Sanyal, Thierry Rochereau, Cara Nichole Maesano, Laure Com-Ruelle, and Isabella Annesi-Maesano. Long-term effect of outdoor air

pollution on mortality and morbidity: a 12-year follow-up study for metropolitan France. *International journal of Environmental Research and Public Health*, 15(11):2487, 2018. 10.3390/ijerph15112487.

[84] Gavin Shaddick, Matthew L Thomas, Heresh Amini, David Broday, Aaron Cohen, Joseph Frostad, Amelia Green, Sophie Gumy, Yang Liu, Randall V Martin, et al. Data integration for the assessment of population exposure to ambient air pollution for global burden of disease assessment. *Environmental Science & Technology*, 52(16):9069–9078, 2018. 10.1021/acs.est.8b02864.

[85] Vasilis Kontis, James E Bennett, Theo Rashid, Robbie M Parks, Jonathan Pearson-Stuttard, Michel Guillot, Perviz Asaria, Bin Zhou, Marco Battaglini, Gianni Corsetti, et al. Magnitude, demographics and dynamics of the effect of the first wave of the COVID-19 pandemic on all-cause mortality in 21 industrialized countries. *Nature Medicine*, 26(12):1919–1928, 2020. s41591-020-1112-0.

[86] Matthias Bollhöfer, Olaf Schenk, Radim Janalik, Steve Hamm, and Kiran Gullapalli. State-of-the-Art Sparse Direct Solvers. In *Parallel Algorithms in Computational Science and Engineering*, pages 3–33. Springer, 2020. 10.1007/978-3-030-43736-7_1.

[87] Sivan Toledo. Taucs: A library of sparse linear solvers. 2003. https://www.tau.ac.il/ stoledo/taucs/.

[88] Håvard Rue and Sara Martino. Approximate Bayesian inference for hierarchical Gaussian Markov random field models. *Journal of Statistical Planning and Inference*, 137(10):3177–3192, 2007. 10.1016/j.jspi.2006.07.016.

[89] Fabio Verbosio, Arne De Coninck, Drosos Kourounis, and Olaf Schenk. Enhancing the scalability of selected inversion factorization algorithms in genomic prediction. *Journal of Computational Science*, 22:99–108, 2017. ISSN 1877-7503. 10.1016/j.jocs.2017.08.013.

[90] S. Li, S. Ahmed, G. Klimeck, and E. Darve. Computing entries of the inverse of a sparse matrix using the FIND algorithm. *Journal of Computational Physics*, 227(22):9408–9427, 2008. ISSN 0021-9991. 10.1016/j.jcp.2008.06.033.

[91] Denis Rustand, Janet van Niekerk, Elias Teixeira Krainski, Håvard Rue, and Cécile Proust-Lima. Fast and flexible inference for joint models of multivariate longitudinal and survival data using integrated nested Laplace approximations. *Biostatistics*, 2023. 10.1093/biostatistics/kxad019.

[92] Daniel Spencer, Yu Ryan Yue, David Bolin, Sarah Ryan, and Amanda F Mejia. Spatial Bayesian GLM on the cortical surface produces reliable task activations in individuals and groups. *NeuroImage*, 2022. 10.1016/j.neuroimage.2022.118908.

[93] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

[94] Naoaki Okazaki, Dirk Toewe, and Yixuan Qiu. LBFGSpp. `https://github.com/yixuan/LBFGSpp/`, 2021.

[95] Mohammed Al Farhan, Ahmad Abdelfattah, Stanimire Tomov, Mark Gates, Dalal Sukkari, Azzam Haidar, Robert Rosenberg, and Jack Dongarra. MAGMA templates for scalable linear algebra on emerging architectures. *The International Journal of High Performance Computing Applications*, 34 (6):645–658, 2020. 10.1177/1094342020938421.

[96] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. CUDA, release: 10.2.89, 2020. URL `https://developer.nvidia.com/cuda-toolkit`.

[97] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback). 10.1137/1.9780898719604.

[98] Matthew J Menne, Imke Durre, Bryant Korzeniewski, Shelley McNeal, Kristy Thomas, Xungang Yin, Steven Anthony, Ron Ray, Russell S Vose, Byron E Gleason, et al. Global historical climatology network-daily (GHCN-Daily), Version 3. *NOAA National Climatic Data Center*, 10:V5D21VHZ, 2012. www.ncdc.noaa.gov.

[99] Timothy A Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006. `https://doi.org/10.1137/1.9780898718881`.

[100] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003. 10.1137/1.9780898718003.

[101] Uri M Ascher and Chen Greif. *A First Course on Numerical Methods*. SIAM, 2011. https://doi.org/10.1137/9780898719987.

[102] Mihalis Yannakakis. Computing the minimum fill-in is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981. https://doi.org/10.1137/0602010.

[103] Michael T Heath, Esmond Ng, and Barry W Peyton. Parallel algorithms for sparse linear systems. *SIAM Review*, 33(3):420–460, 1991. https://doi.org/10.1137/1033099.

[104] Charles-Edmond Bichot and Patrick Siarry. *Graph partitioning*. John Wiley & Sons, 2013. https://doi.org/10.1007/978-3-319-63962-8_312-1.

[105] Alan George and Joseph WH Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989. https://doi.org/10.1137/1031001.

[106] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. 10.1137/0710032.

[107] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998. 10.5555/305219.305248.

[108] Andrey Kuzmin, Mathieu Luisier, and Olaf Schenk. Fast Methods for Computing Selected Elements of the Green's Function in Massively Parallel Nanoelectronic Device Simulations. In *Proceedings of the 19th International Conference on Parallel Processing*, Euro-Par'13, page 533–544, Berlin, Heidelberg, 2013. Springer-Verlag. 10.1007/978-3-642-40047-6_54.

[109] Alexandros Nikolaos Ziogas, Tal Ben-Nun, Guillermo Indalecio Fernández, Timo Schneider, Mathieu Luisier, and Torsten Hoefler. A Data-Centric Approach to Extreme-Scale Ab Initio Dissipative Quantum Transport Simulations. SC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362290. 10.1145/3295500.3357156.

[110] Amir Asif and José MF Moura. Inversion of block matrices with block banded inverses: Application to Kalman-Bucy filtering. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*, volume 1, pages 608–611. IEEE, 2000. 10.1109/ICASSP.2000.862055.

[111] Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. 10.1137/0710032.

[112] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36(5-6):232–240, 2010. 10.1016/j.parco.2009.12.005.

[113] Jose Monsalve Diaz, Swaroop Pophale, Oscar Hernandez, David E. Bernholdt, and Sunita Chandrasekaran. Openmp 4.5 validation and verification suite for device offload. In Bronis R. de Supinski, Pedro Valero-Lara, Xavier Martorell, Sergi Mateo Bellido, and Jesus Labarta, editors, *Evolving OpenMP for Evolving Architectures*, pages 82–95, Cham, 2018. Springer International Publishing. https://www.openmp.org.

[114] Randall J LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-state and Time-dependent Problems*. SIAM, 2007. 10.1137/1.9780898717839.

[115] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018. https://dl.acm.org/doi/abs/10.5555/3122009.3242010.

[116] Bart Van Merriënboer, Olivier Breuleux, Arnaud Bergeron, and Pascal Lamblin. Automatic differentiation in ML: Where we are and where we should be going. *Advances in Neural Information Processing Systems*, 31, 2018. https://proceedings.neurips.cc/paper/2018/file/770f8e448d07586afbf77bb59f698587-Paper.pdf.

[117] Peter J Rousseeuw and Annick M Leroy. *Robust Regression and Outlier Detection*, volume 589. John Wiley & Sons, 2005. 10.1002/0471725382.

[118] Anthony Curtis Atkinson, Marco Riani, and M Riani. *Robust Diagnostic Regression Analysis*, volume 2. Springer, 2000. https://doi.org/10.1007/978-1-4612-1160-0.

[119] James W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997. 10.1137/1.9781611971446.

[120] Victor Pan and John Reif. Efficient parallel solution of linear systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*, pages 143–152, 1985. https://doi.org/10.1145/22145.22161.

[121] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 4.0*, June 2021. URL `https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf`.

[122] Elias T Krainski, Virgilio Gómez-Rubio, Haakon Bakka, Amanda Lenzi, Daniela Castro-Camilo, Daniel Simpson, Finn Lindgren, and Håvard Rue. *Advanced Spatial modeling with Stochastic Partial Differential Equations using R and INLA*. CRC Press, 2018. 10.1201/9780429031892.

[123] Robin Henderson, Silvia Shimakura, and David Gorst. Modeling spatial variation in leukemia survival data. *Journal of the American Statistical Association*, 97(460):965–972, 2002. `https://doi.org/10.1198/016214502388618753`.

[124] Daniel Simpson, Håvard Rue, Andrea Riebler, Thiago G Martins, and Sigrunn H Sørbye. Penalising model component complexity: A principled, practical approach to constructing priors. *Statistical Science*, 32(1): 1–28, 2017. 10.1214/16-STS576.

[125] Harold Jeffreys. *The theory of probability*. OuP Oxford, 1998.

[126] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995. 10.1080/01621459.1995.10476572.

[127] Michael D Lee and Eric-Jan Wagenmakers. *Bayesian Cognitive Modeling: A Practical Course*. Cambridge university press, 2014. 10.1017/CBO9781139087759.

[128] Serge Lang. *Undergraduate analysis*. Springer Science & Business Media, 2013. 10.1007/978-1-4757-2698-5.

[129] James Stewart. *Essential Calculus: Early Transcendentals*. Brooks/Cole, a part of the Thomson Corporation, 2007.