Statistics and Computing (2023) 33:25
https://doi.org/10.1007/s11222-022-10192-1

ORIGINAL PAPER

Parallelized integrated nested Laplace approximations for fast Bayesian inference

Lisa Gaedke-Merzhäuser[1] · Janet van Niekerk[2] · Olaf Schenk[1] · Håvard Rue[2]

Received: 15 April 2022 / Accepted: 6 December 2022

Abstract
There is a growing demand for performing larger-scale Bayesian inference tasks, arising from greater data availability and higher-dimensional model parameter spaces. In this work we present parallelization strategies for the methodology of integrated nested Laplace approximations (INLA), a popular framework for performing approximate Bayesian inference on the class of Latent Gaussian models. Our approach makes use of nested thread-level parallelism, a parallel line search procedure using robust regression in INLA's optimization phase and the state-of-the-art sparse linear solver PARDISO. We leverage mutually independent function evaluations in the algorithm as well as advanced sparse linear algebra techniques. This way we can flexibly utilize the power of today's multi-core architectures. We demonstrate the performance of our new parallelization scheme on a number of different real-world applications. The introduction of parallelism leads to speedups of a factor 10 and more for all larger models. Our work is already integrated in the current version of the open-source R-INLA package, making its improved performance conveniently available to all users.

Keywords Bayesian inference · INLA · Parallelism · Mathematical software

1 Introduction

The methodology of integrated nested Laplace approximations (INLA) has become a widely spread framework for performing complex Bayesian inference tasks, see e.g. (Rue et al., 2017; Opitz, 2017; Bakka et al., 2018). INLA is applicable to a wide subclass of additive Bayesian hierarchical models. It employs a deterministic approximation scheme that relies on Gaussian Markov random fields (GMRFs) in the latent parameter space which enables the usage of efficient sparse linear algebra techniques (Rue et al. 2009). A user-friendly implementation of INLA is available in the form of an R-package under the same name, referred to as R-INLA. It can be downloaded and installed as described on `www.r-inla.org`, with the complete source code available on GitHub[1]. 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. (Lindgren et al. 2011; Martins et al. 2013), including a wide variety of applications, see e.g. (Batomen et al. 2020; de Rivera et al. 2019; Lu et al. 2018; Arisido et al. 2017; Bhatt et al. 2015; Konstantinoudis et al. 2021; Mielke et al. 2020; Coll et al. 2019; Martínez-Minaya et al. 2018; Isaac et al. 2020; Lindenmayer et al. 2021; Pimont et al. 2021; Pinto et al. 2020; Lillini et al. 2021; Sanyal et al. 2018; Shaddick et al. 2018; Kontis et al. 2020). 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. In this work we want to invite the reader to look at the INLA methodology with us from a more algorith-

✉ Lisa Gaedke-Merzhäuser
lisa.gaedke.merzhaeuser@usi.ch

Janet van Niekerk
janet.vanniekerk@kaust.edu.sa

Olaf Schenk
olaf.schenk@usi.ch

Håvard Rue
haavard.rue@kaust.edu.sa

[1] Faculty of Informatics, Universita della Svizzera italiana, 6900 Lugano, Switzerland

[2] CEMSE Division, King Abdullah University of Science and Technology, Thuwal 23955, Saudi Arabia

[1] https://github.com/hrue/r-inla.

Published online: 24 December 2022

mic point of view. We will present a much more performant implementation of R-INLA making use of OpenMP (Diaz et al. 2018) parallelism and the state-of-the-art sparse linear solver PARDISO (Bollhöfer et al. 2020; PARDISO 2022).

The introduction of OpenMP allows for the simultaneous execution of multiple tasks within the algorithm operating on shared memory. We have parallelized the computationally intensive operations within INLA whenever the dependency structure allowed it, which also opened the door for us to add more robust approximation strategies. Internally, the PARDISO library is also utilizing multiple hardware threads which leads to a double layer of parallelism within INLA. A large quantity of the algorithm's overall runtime is spent in an optimization routine to determine the posterior mode of the model's hyperparameters. The mode is found iteratively using a quasi-Newton method that requires gradient information of the functional being maximized. Approximating the gradient in each iteration is a computationally expensive task, which we were, however, able to parallelize. Additionally, INLA performs a line search procedure in every iteration of the optimization to find a suitable parameter configuration for the next step. Instead of sequentially looking for a good candidate we have parallelized this process using multiple threads. We now evaluate multiple candidates at once within an interval along the search direction. We use this information to construct a smooth local approximation of the objective function fitted through a robust regression scheme. The optimum of the local approximation becomes the new parameter configuration. This allows us to save time in each iteration. It additionally leads to more stability in the optimization routine as it rectifies small numerical discontinuities and can therefore also reduce the total number of iterations required. To compute the posterior marginal distributions of the latent parameters of the model partial matrix inversions are necessary. For all computations INLA generally works with the sparse precision matrices instead of their dense counterparts, the corresponding covariance matrices. However, to obtain the marginal variances, inversions of often high-dimensional precision matrices are necessary. A time-consuming task which we have adapted to be executed in parallel. The collection of these strategies forms the first or top layer parallelism.

The original implementation of INLA employs the sparse linear algebra library TAUCS (Toledo 2003) to perform the required numerically intensive core operations. Foremost, these include Cholesky decompositions of matrices with recurring sparsity patterns and the partial matrix inversions. TAUCS is a well-designed and efficient library which operates, however, sequentially and whose support was discontinued in the early 2000's. Additionally, TAUCS does not include a partial inversion routine which was therefore implemented by the authors of R-INLA, see (Rue and Martino 2007; Rue et al. 2009), in a sequential manner. While this was done with much thought, effort and consideration, providing reliable results, improved parallel implementations have emerged since (Verbosio et al. 2017; Li et al. 2008). They were developed by experts in the field of high-performance computing and sparse numerical solvers. The plan of integrating PARDISO into INLA was first described by Van Niekerk et al. (2021). The authors discuss the need for the usage of faster numerical solvers within INLA to support evolving statistical models of higher complexity and continuously growing availability of data. They also show the efficiency and scalability with which PARDISO performs the beforementioned numerically intensive core operations like Cholesky decompositions and partial matrix inversions. Especially for the latter task there are not many performant libraries available which make PARDISO a particularly great fit (Van Niekerk et al. 2021). Hence, the second layer of parallelism is formed by PARDISO as well as other parallelized linear algebra operations like matrix-matrix or matrix-vector products. We will present performance results for three different real-world applications, quantifying the improvement compared to previous implementations in terms of speedup and scalability. The case studies include a complex joint survival model containing 50 hyperparameters (Rustand et al. 2022), a brain activation model using fMRI data with very high-dimensional spatial domains (Spencer et al. 2022) and a smaller scale model describing spatial variation in Leukemia survival data (Lindgren et al. 2011). All corresponding code and data is publicly available.

The rest of the paper is organized as follows. We will begin with a brief introduction of the INLA methodology, presenting the class of applicable models and underlying statistical concepts, with a particular focus on the computations they entail. Additionally we will provide an overview of the fundamental numerical methods and sparse linear algebra operations involved in the implementation. Then we will introduce the new parallelization schemes, before demonstrating our improvements on the various applications.

2 Background

2.1 Latent gaussian models

The INLA methodology is applicable to the class of latent Gaussian models (LGMs). They 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. (Rue et al. 2009; Martins et al. 2013; Rue et al. 2017; Bakka et al. 2018) for details. 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. The additive linear predictor η_i is

defined as

$$\eta_i = \boldsymbol{\beta}^T \mathbf{Z}_i + \mathbf{u}_i(\mathbf{w}_i). \tag{1}$$

It has fixed effects $\boldsymbol{\beta}$ with associated covariates $\mathbf{Z}_i$ and nonlinear random effects $\mathbf{u}_i$ with associated covariates w_i representing e.g. temporal, spatial or spline effects. The observations are assumed to be conditionally independent given the parameters, such that

$$y \mid \eta, \boldsymbol{\theta} \sim \prod_{i=1}^{m} \pi(y_i|\eta_i, \boldsymbol{\theta}), \tag{2}$$

where $\boldsymbol{\theta}$ are the hyperparameters of the model. The latent parameter space of the model has traditionally been defined as $\boldsymbol{x} = (\boldsymbol{\beta}, \boldsymbol{u}, \boldsymbol{\eta})$, but can now also be formulated without the linear predictor as $\boldsymbol{x} = (\boldsymbol{\beta}, \boldsymbol{u})$, which was recently presented in Van Niekerk et al. (2022). The concepts discussed in this work are equally applicable to the two formulations. In both cases we assume that $\boldsymbol{x}$ forms a Gaussian Markov random field (GMRF) with zero mean and sparse precision matrix $\boldsymbol{Q}_x(\boldsymbol{\theta})$, i.e. $\boldsymbol{x} \sim \mathcal{N}(0, \boldsymbol{Q}_x(\boldsymbol{\theta})^{-1})$. The sparse GMRF structure is one of the key components to INLA's computational efficiency. It allows for a very high-dimensional parameter space without performance degradation due to the employed sparse linear algebra operations (Rue and Held 2005). In addition one adopts a prior $\pi(\boldsymbol{\theta})$ for the hyperparameters $\boldsymbol{\theta}$. Thus, forming a three-stage model consisting of the hyperparameter distribution, the latent field and the likelihood. It is also possible to account for constraints imposed on the latent parameters $\boldsymbol{x}$. A detailed discussion of what types of constraints can be imposed and how this is accounted for are e.g. presented in [Rue et al. 2005 Sect. 2.3.3]. For simplicity, we will not discuss the constraint case in further detail, since the same parallelization strategies are employed as in the unconstrained framework. The speedup obtained in the parallel constraint case compared to the sequential constrained case resembles the respective speedup of the unconstrained problem.

The main objective in Bayesian analysis is to compute the marginal posterior densities of the unknown parameters, which in the case of latent Gaussian models are the latent parameters $\boldsymbol{x}$ and the hyperparameters $\boldsymbol{\theta}$. For a further discussion of the model specifications or Bayesian hierarchical models see (Rue et al. 2017; Congdon 2014).

2.2 Approximating $\pi(\theta_j|\boldsymbol{y})$

INLA does not provide an estimate to the full joint posterior $\pi(\boldsymbol{x}, \boldsymbol{\theta}|\boldsymbol{y})$ but instead computes the posterior marginal distributions $\pi(\theta_j|\boldsymbol{y})$ and $\pi(x_i|\boldsymbol{y})$ as well as other relevant statistics (Rue et al. 2009). The first step is to construct an approximation to the posterior of the hyperparameters $\pi(\boldsymbol{\theta}|\boldsymbol{y})$ that can be evaluated for fixed values of $\boldsymbol{\theta}$, see Eq. (3). This is done through an approximation scheme using Bayes' rule (alternatively it can also be derived using Laplace approximations) as follows

$$\begin{aligned}\pi(\boldsymbol{\theta}|\boldsymbol{y}) = \frac{\pi(\boldsymbol{x}, \boldsymbol{\theta}|\boldsymbol{y})}{\pi(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})} &\propto \frac{\pi(\boldsymbol{\theta})\pi(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})}{\pi(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})} \\ &\approx \frac{\pi(\boldsymbol{\theta})\pi(\boldsymbol{x}|\boldsymbol{\theta})\pi(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})}{\pi_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})}\bigg|_{\boldsymbol{x}=\boldsymbol{x}^*(\boldsymbol{\theta})} := \tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y}).\end{aligned} \tag{3}$$

The goal is to find the mode of $\tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y})$, denoted by $\boldsymbol{\theta}^*$, as it is not known a priori. But first, we want to discuss the arising terms of Eq. (3) in more detail. Note that since $\pi(\boldsymbol{\theta}|\boldsymbol{y})$ is independent of $\boldsymbol{x}$, the equality holds for any $\boldsymbol{x}$. The posterior conditional distribution $\pi_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$ denotes a Gaussian approximation to $\pi(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$. It is constructed by matching the mode, which we denote by $\boldsymbol{x}^*(\boldsymbol{\theta})$, as well as the curvature at the mode, to the true conditional posterior, for details see (Rue et al. 2009). The approach preserves the sparsity of the prior precision matrix in the posterior precision matrix $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta})$ where we condition on the data. We can write the log-density of π_G as

$$\begin{aligned}\log(\pi_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})) = -\frac{1}{2}\big[&(n\log(2\pi) - \log(|\boldsymbol{Q}_{x|y}(\boldsymbol{\theta})|) \\ &+ (\boldsymbol{x}^*(\boldsymbol{\theta}) - \boldsymbol{x})^T \boldsymbol{Q}_{x|y}(\boldsymbol{\theta})(\boldsymbol{x}^*(\boldsymbol{\theta}) - \boldsymbol{x})\big].\end{aligned} \tag{4}$$

All computations are performed in log-scale as this is favorable for dependence structures and in terms of numerical robustness. To compute $\tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y})$ for a fixed value of $\boldsymbol{\theta}$, each term is evaluated individually. From a computational perspective this is relatively cheap for the prior of the hyperparameters $\pi(\boldsymbol{\theta})$ and the likelihood $\pi(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})$ due to the conditional independence assumption made in Eq. (2). The prior of the latent parameters $\pi(\boldsymbol{x}|\boldsymbol{\theta})$, on the other hand, forms a potentially high-dimensional GMRF. To evaluate this log-density one has to compute the normalizing constant which includes the log-determinant of $\boldsymbol{Q}_x(\boldsymbol{\theta})$. The computational complexity of computing the determinant of a dense matrix is in general $O(n^3)$, where n denotes the size of the matrix and corresponds in our case to the dimension of the latent parameter space. For sparse matrices, and therefore also $\boldsymbol{Q}_x(\boldsymbol{\theta})$, this cost can often be lowered significantly through employing specialized solution methods but it nevertheless continues to pose an expensive computational task in high dimensions [Rue et al. 2005 Sect. 2.3]. To evaluate $\pi_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$ one also requires its normalizing constant, and therefore the log-determinant of $\boldsymbol{Q}_{x|y}$. One, additionally, needs to find its mode $\boldsymbol{x}^*(\boldsymbol{\theta})$ (for the prior this is known a priori, as it is defined to be the zero vector) and precision matrix $\boldsymbol{Q}_{x|y}$, which is however straight-forward to construct given $\boldsymbol{\theta}$ and $\boldsymbol{y}$. If the likelihood is Gaussian, $\boldsymbol{x}^*(\boldsymbol{\theta})$ can be found through a standard update for conditional nor-

mal distributions, that requires solving a linear system of equations involving $\boldsymbol{Q}_{x|y}$. When considering the case, where the likelihood is non-Gaussian, we have to perform an iterative optimization over $\boldsymbol{x}$ to find $\boldsymbol{x}^*(\boldsymbol{\theta})$. As $\boldsymbol{Q}_{x|y}$ depends on the current vector $\boldsymbol{x}$, a new normalizing constant and a new mode are recomputed for every new vector $\boldsymbol{x}$. Thus, requiring more log-determinant evaluations and solutions to linear systems of equations of size $n \times n$, further increasing the overall computational cost.

In the first phase of the INLA framework, an optimization algorithm is employed to iteratively evaluate $\tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y})$ for different values of $\boldsymbol{\theta}$ to find the hyperparameter configuration $\boldsymbol{\theta}^*$ for which $\boldsymbol{\theta}^* = \arg\max\ \tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y})$. Further details of how this optimization problem is solved follow in Sect. 3.1. Once the maximum is determined, the area around the mode is explored. This is done through reparameterizing the parameter space using information from the Hessian at the mode, and subsequently choosing a set of K selected integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$, at which $\tilde{\pi}(\boldsymbol{\theta}^k|\boldsymbol{y})$ is evaluated. The values are then used to approximate the posterior marginals for each hyperparameter θ_j using numerical integration. For a detailed description we refer to Rue et al. (2009). From a computational perspective the most expensive and hence time-consuming kernel operations are the repeated evaluations of Eq. (3), first during the optimization phase to determine the mode $\boldsymbol{\theta}^*$, then for the Hessian approximation (see Sect. 3.1 for details) at the mode and finally at the integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$.

2.3 Approximating $\pi(x_i|\boldsymbol{y})$

The posterior marginals $\pi(x_i|\boldsymbol{y})$ can be written as

$$\pi(x_i|\boldsymbol{y}) = \int \pi(\boldsymbol{\theta}|\boldsymbol{y})\pi(x_i|\boldsymbol{\theta}, \boldsymbol{y})d\boldsymbol{\theta}, \tag{5}$$

for which an approximation to the first factor of the right-hand side and numerical integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$ are already known from the previous step. It remains to estimate $\pi(x_i|\boldsymbol{\theta}, \boldsymbol{y})$. If the likelihood is normally distributed, the Gaussian approximation $\pi_G(\boldsymbol{x}|\boldsymbol{\theta}, \boldsymbol{y})$ from Eq. (3) is exact. The posterior marginals of $\pi(x_i|\boldsymbol{\theta}, \boldsymbol{y})$ can then be approximated by extracting the latent parameter configuration $\boldsymbol{x}$ at the mode of $\pi(\boldsymbol{\theta}|\boldsymbol{y})$ from which one can directly infer the marginal means $x_i|\boldsymbol{x}_{-i}, \boldsymbol{\theta}, \boldsymbol{y}$. Here, $\boldsymbol{x}_{-i}$ denotes the vector $\boldsymbol{x}$ without the i-th entry. The marginal variances are computed from π_G, again, evaluated at the modal configuration. This, however, requires at least a partial inversion of the precision matrix of π_G, a computationally intensive operation especially, for high-dimensional latent parameters spaces. In the non-Gaussian case, the approximation procedure is more involved. An update for the correction of the conditional mean of $x_i|\boldsymbol{x}_{-i}, \boldsymbol{\theta}, \boldsymbol{y}$ for each i needs to be performed. Additionally, the marginal variances are computed at all integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$ which entails a partial matrix inversion for each precision matrix $\boldsymbol{Q}_{x|y}(\boldsymbol{\theta}^k)$. This allows INLA to more accurately account for the overall shape and potential skewness in the distributions. Using this information approximations to the marginal distributions $\pi(x_i|\boldsymbol{\theta}, \boldsymbol{y})$ are constructed, which are then used in Eq. (5) to compute the posterior marginals using numerical integration. For a detailed overview we refer to Rue et al. (2009).

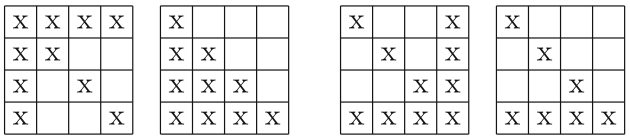
2.4 Sparse linear algebra

In this section we want to provide a quick overview of the underlying sparse linear algebra techniques leveraged by the R-INLA package. 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 (Davis 2006; Saad 2003).

During the first stage of the INLA algorithm, see Sect. 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-(semi)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 (Ascher and Greif 2011). 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 $\boldsymbol{L}$.

2.4.1 Matrix reordering

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 (Davis 2006). As an illustrative example, we consider a symmetric



(a) $\boldsymbol{Q}_1$ (b) $\boldsymbol{L}_1$ (c) $\boldsymbol{Q}_2$ (d) $\boldsymbol{L}_2$

Fig. 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 $\boldsymbol{Q}_2$

positive-definite arrowhead matrix $\boldsymbol{Q}_1$ whose sparsity pattern can be seen in Fig. 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.

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 never changes throughout the optimization phase but only the 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 (Yannakakis 1981) and has hence been an active area of research (Heath et al. 1991; Bichot and Siarry 2013).

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 (George and Liu 1989). Another popular class of reordering strategies called nested dissection employs graph partitioning techniques (George 1973; Bichot and Siarry 2013). 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 Fig. 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 then recursively refined while being propagated back to the original large graph. From the final edge cut we can deduce a separating set $\mathcal{G}_S$ (Karypis and Kumar 1998). The same strategy of 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 (Karypis and Kumar 1998) 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 Sect. 3.3.

2.4.2 Partial inversion

In the second stage of the INLA algorithm, see Sect. 2.3, an estimate for the variances of the posterior marginals of the latent parameters, i.e. $\pi(x_i|\boldsymbol{y})$ for all i, is obtained. To do so INLA uses information from the Gaussian approximations $\pi_G(\boldsymbol{x}|\boldsymbol{\theta}^k, \boldsymbol{y})$ at the various integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$. All information about the marginal variances only exists, however, in form of the precision matrices. As the variances Σ_{ii} correspond to $(\boldsymbol{Q}^{-1})_{ii}$, they can only be obtained through matrix inversions. 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 (2007). 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 (1973) 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 $\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{L}^T$ (Rue and Held 2005). Since $\boldsymbol{L}^T$ is an upper triangular matrix, we can use a backward solve and recursively compute

$$x_n = \frac{z_n}{L_{nn}}$$

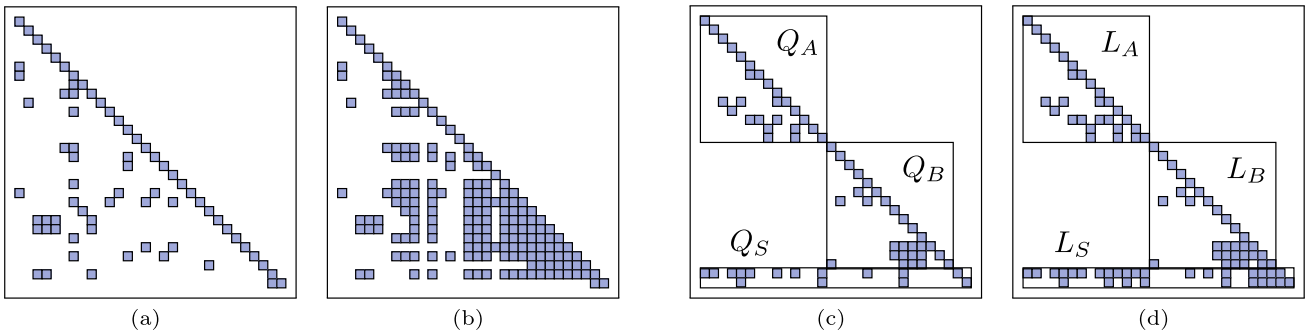


Fig. 2 From left to right: **a** Lower triangular part of the sparsity pattern of a random symmetric positive definite matrix $\boldsymbol{Q}$, with 70 non-zero entries. **b** Cholesky decomposition $\boldsymbol{L}$ of $\boldsymbol{Q}$, where L has 180 non-zero entries. **c** Permuted matrix $\boldsymbol{Q}_{\text{perm}}$ using (nested) dissection. **d** Cholesky decomposition $\boldsymbol{L}_{\text{perm}}$ of $\boldsymbol{Q}_{\text{perm}}$, where $\boldsymbol{L}_{\text{perm}}$ 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 last 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. Thus, reducing the number of non-zeros in the Cholesky factorization of $\boldsymbol{Q}_{\text{perm}}$. The strategy can be applied recursively to each of the components $\boldsymbol{Q}_A$ and $\boldsymbol{Q}_B$

$$x_i = \frac{z_i}{L_{ii}} - \frac{1}{L_{ii}} \sum_{\substack{k>i \\ L_{ki}\neq 0}}^{n} L_{ki} x_k \tag{6}$$

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 z as well as Eq. (6), we can deduce the covariance matrix $\boldsymbol{\Sigma}$ of $\boldsymbol{x}$ from first principles, obtaining

$$\Sigma_{ij} = \frac{\delta_{ij}}{L_{ii}^2} - \frac{1}{L_{ii}} \sum_{\substack{k>i \\ L_{ki}\neq 0}}^{n} L_{ki} \Sigma_{kj}, \tag{7}$$

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 Eq. (7) 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 Takahashi (1973),

$$\boldsymbol{\Sigma} = \boldsymbol{D}^{-1}\boldsymbol{V}^{-1} + (\boldsymbol{I} - \boldsymbol{V}^T)\boldsymbol{\Sigma}. \tag{8}$$

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 Eq. (8) as sums we obtain

$$\Sigma_{ij} = \frac{\delta_{ij}}{D_{ii}} - \sum_{\substack{k>i \\ V_{ki}\neq 0}} V_{ki} \Sigma_{kj}, \quad \text{for } i \leq j \tag{9}$$

which we can compute recursively starting from Σ_{nn} and where again, δ_{ij} is equal to one for $i = j$ and zero otherwise. Eqs. (7) and (9) 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 need. 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 Sect. 3.3 for details. The marginal variances Σ_{ii} that we obtain are then repermuted to correspond to the original ordering of the parameters $\boldsymbol{x}$. Afterwards, they can be used as described in Sect. 2.3 to obtain estimates of the marginal posterior distributions of the latent parameters.

2.5 BFGS algorithm

To solve the optimization problem described in Sect. 2.2, INLA uses a BFGS algorithm. It is an iterative quasi-Newton method for solving unconstrained nonlinear optimization problems [Nocedal and Wright 2006 Ch. 6]. Quasi-Newton methods resemble Newton methods without requiring knowledge about 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 [Nocedal and Wright 2006 Ch. 6]. It minimizes a differentiable function $f(\boldsymbol{\theta})$, starting from an initial guess $\boldsymbol{\theta}_0$ by using the following update formulae

$$\boldsymbol{\theta}_{l+1} = \boldsymbol{\theta}_l - \alpha_l \boldsymbol{B}_l^{-1} \nabla f_{\boldsymbol{\theta}_l}. \tag{10}$$

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 [Nocedal and Wright 2006 Ch. 6]. Details on how this is implemented in INLA are described in the next section, including gradient approximations for $\nabla f_{\boldsymbol{\theta}_l}$ and finding appropriate step sizes α_l, as they are closely linked to opportunities for parallelism.

3 Parallelization scheme

In the following we present the novel parallelization strategies that have been added to the current R-INLA implementation. There are four significant updates in the algorithm that we want to discuss in detail. The first one comprises parallel function evaluations of Eq. (3) for different values of $\boldsymbol{\theta}$. We determined multiple occasions throughout the algorithm that allow for such parallelism, and discuss the details in Sect. 3.1. The second update consists of parallel precision matrix inversions which are required to compute the posterior marginal distributions of the latent parameters. Thirdly, we introduce a parallel line search strategy within the optimization routine of the algorithm, see Sect. 3.2 for details. The last update applies to all stages of the algorithm and consists of the incorporation of the PARDISO library into R-INLA to handle all required major sparse linear algebra operations, and will be discussed in Sect. 3.3. The parallel operations are realized through the usage of OpenMP, an application programming interface that supports multiprocessing for shared-memory architectures and allows to efficiently execute multiple tasks at once. A schematic overview of the parallelization scheme is presented in Fig. 3.

3.1 Level 1 parallelism – parallel function evaluations

Among the computationally intensive and therefore time-consuming operations in INLA are the function evaluations of the posterior marginal distribution, i.e. computing $\tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y})$ at $\boldsymbol{\theta}$, as discussed in Sect. 2.2. Instead of evaluating the posterior directly, we consider

$$f(\boldsymbol{\theta}) := -\log \tilde{\pi}(\boldsymbol{\theta}|\boldsymbol{y}). \tag{11}$$

The logarithmic scale is introduced to enhance numeric stability. Finding the mode $\boldsymbol{\theta}^*$ of the posterior becomes a minimization problem through the sign switch in Eq. (11). The employed BFGS algorithm requires knowledge of the gradient of f, see Sect. 2.5.

We estimate it numerically using a finite difference scheme (LeVeque 2007). Each partial derivative $\partial f/\partial\theta_i$ of the gradient is approximated using either a first order forward or central difference. As an example we show the central difference approximation

$$\frac{\partial f}{\partial \theta_i}(\text{þ}) \approx \frac{f(\boldsymbol{\theta} + \boldsymbol{\epsilon}_i) - f(\boldsymbol{\theta} - \boldsymbol{\epsilon}_i)}{2\,||\boldsymbol{\epsilon}_i||} \text{ for all } i, \tag{12}$$

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 (Fattah et al. 2022) 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 Eq. (12) 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. $f(\boldsymbol{\theta} + \boldsymbol{\epsilon}_1)$, $f(\boldsymbol{\theta} - \boldsymbol{\epsilon}_1)$, $f(\boldsymbol{\theta} + \boldsymbol{\epsilon}_2)$, ... simultaneously in each iteration, see Fig. 3. Here, each $f(\boldsymbol{\theta}_l^i)$ denotes a different function value $f(\boldsymbol{\theta} \pm \boldsymbol{\epsilon}_i)$ 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 $f(\boldsymbol{\theta})$ 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 the already necessary iterative evaluations of $f(\boldsymbol{\theta})$ for fixed values of $\boldsymbol{\theta}$, see Fig. 3.

There are other methods to obtain gradient information, in particular automatic differentiation (Baydin et al. 2018). In the ideal case it offers a highly accurate solution at runtimes comparable to a single function evaluation (Baydin et al. 2018), 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 (Van Merriënboer et al. 2018) and would require major changes in the PARDISO software library. Additionally, if the likelihood is non-Gaussian we have the inner optimiza-

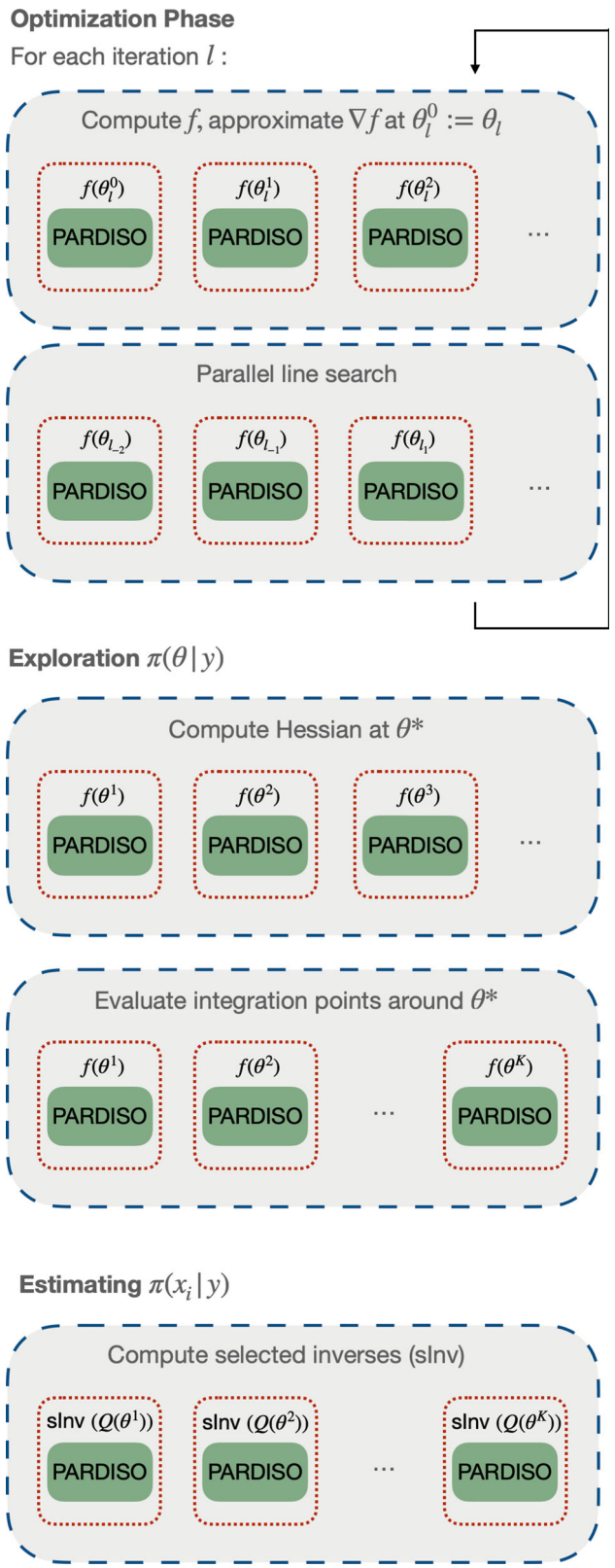


Fig. 3 Simplified overview of the parallelization scheme within INLA throughout the different phases of the algorithm. Each dashed blue box indicates a region of level 1 parallelism. Each dotted red box indicates a region of level 2 parallelism within the parallel level 1 regions. At the end of every parallel region (level 1 & 2) there is an intrinsic synchronization, where execution halts until all assigned threads have concluded their tasks.

tion routine within each function evaluation, for which we are not aware of existing automatic differentiation methods.

After the maximum of $\tilde{\pi}(\boldsymbol{\theta} \mid \boldsymbol{y})$ 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 Fig. 3. Next, we can also perform the function evaluations at the integration points $\{\boldsymbol{\theta}^k\}_{k=1}^K$ in parallel, see Sect. 2.2. 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 completely 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 Fig. 3), 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.

3.2 Parallel line search

In addition to approximating the gradient, a value for the step size α_l needs to be chosen in every iteration of the optimization routine. Determining a suitable value for α_l is crucial to the convergence of the overall algorithm and is done through a line search procedure, see e.g. [Nocedal and Wright 2006 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 Eq. (10) 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 Eq. (10) 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 now be to 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 (Rousseeuw and Leroy 2005; Atkinson et al. 2000). This way slight inaccuracies in the function evaluations (that can e.g. arise from more complicated choices of likelihood 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 iterate $\boldsymbol{\theta}_{l+1}$. In Fig. 4 we can see an illustration of the new strategy, while Fig. 3 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 Sect. 4 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.

3.3 Level 2 parallelism – parallelization within PARDISO

The PARDISO library offers routines to efficiently provide solutions to various problems commonly arising in the field of sparse linear algebra (Demmel 1997). 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 thorough algorithm selection to detailed hardware considerations.

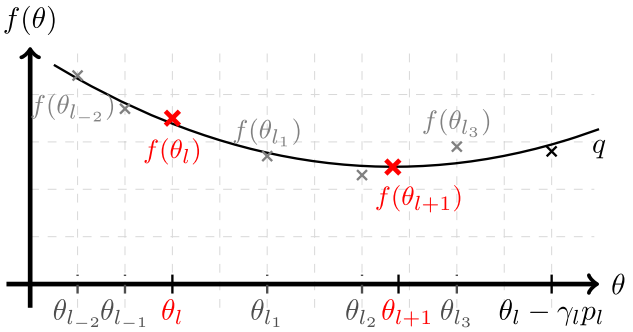


Fig. 4 Illustration of the parallel line search using robust regression. Let $\boldsymbol{\theta}_l$ be the current iterate with function value $f(\boldsymbol{\theta}_l)$, new search direction $\boldsymbol{p}_l$, with search interval $I = (\boldsymbol{\theta}_l, \boldsymbol{\theta}_l - \gamma_l \boldsymbol{p}_l)$ 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 $\boldsymbol{\theta}_{l+1}$

In this work we want to solely focus on the employed parallelism and refer the interested reader to Bollhöfer et al. (2020) for a comprehensive overview.

As described in Sect. 2.4.1, 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 Fig. 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 a 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 (Pan and Reif 1985).

A similar principle can be applied to determine the partial inverse of $\boldsymbol{Q}$, for which we have seen in Sect. 2.4.2 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 Fig. 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 $\sum$ 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 $\sum$ in many parallel regions (Bollhöfer et al. 2020).

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 Sects. 3.1 and 3.2. Additionally to the parallelism within PARDISO, OpenMP is used on the second level to parallelize other matrix operations e.g. during the computation of matrix-matrix or matrix-vector products and while assembling matrices.

4 Results & benchmarks

In this section we present performance results of the newly introduced parallelization strategies for various applications. They are based on previously published models that make use of the R-INLA package. 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, of course, to a smaller extent than compared to a larger computer architecture.

Case Study I : Joint survival modeling of randomized clinical trial. Recently, Rustand et al. (2022) presented a joint survival model consisting of multivariate longitudinal markers paired with competing risks of events, available in the R package **INLAjoin** [2]. 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 hyperparamter, with dimension $d(\boldsymbol{\theta}) = 50$, see Table 1

Table 1 Overview parameters Case Study I

	d($\boldsymbol{\theta}$)	# Lat. par	# of obs.
CS I	50	51 290	27 330

Table 2 Overview parameters Case Study II

	d($\boldsymbol{\theta}$)	Res. size	# Lat. par	# Of obs.
Med. CSII	9	5 000	35 544	2 541 396
Large CSII	9	25 000	183 624	13 129 116

Case Study II : Cortical surface modeling of human brain activation. Spencer et al. show 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 (Lindgren et al. 2011, 2022; Krainski et al. 2018) 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. The likelihood is assumed to follow a Gaussian distribution and we therefore do not require an inner optimization loop for every function evaluation. More details can be found in Spencer et al. (2022); Mejia et al. (2020) which also includes information about the corresponding R package **BayesfMRI**[3]. For this work we fit a single subject, single repetition model 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 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, as shown in Table 2.

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 Henderson et al.

[2] https://github.com/DenisRustand/INLAjoint.

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

Table 3 Overview parameters Case Study III

	d($\boldsymbol{\theta}$)	# Lat. par	# of obs.
CS III	3	7945	6174

(2002); Lindgren et al. (2011). 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, see Table 3

The fitted model provides survival estimates given the covariates and spatial locations.

4.1 Leveraging level 1 parallelism

The level 1 parallelism enables simultaneous function evaluations as described in Sect. 3.1 which are relevant for the gradient computation during the optimization, the Hessian approximation at the mode, the evaluation of at the integration points as well as the partial inversion scheme around the mode.

The speedup that can theoretically be obtained through parallelization depends on the number of hyperparamters. 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 $2 \cdot d(\boldsymbol{\theta}) + 1$. 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 portion of the total runtime. On the other hand, larger applications using a very large number of threads might not exhibit further speedup 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 Fig. 5 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 f 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 speedup 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 less 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.

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 instead of 1 thread, see Table 4. The results confirm the theoretical expectation, that the level 1 parallelism scales close to linearly, in particular for thread counts smaller than $d(\boldsymbol{\theta}) + 1$.

4.2 Leveraging parallel line search

The accuracy of robust regression clearly 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 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 Fig. 6, 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 scaling behavior when it comes to reduction in runtime over an increasing thread count.

4.3 Leveraging level 2 parallelism

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 Sect. 3.3. 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 becom-

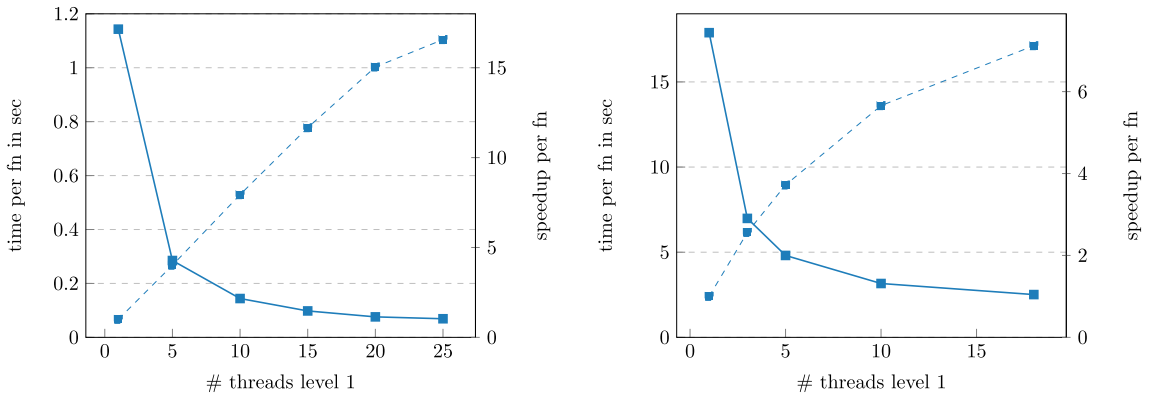


Fig. 5 The solid line shows the total runtime in seconds divided by the number of function evaluations for different numbers of threads on level 1 with the number of level 2 threads fixed to one and without the parallel line search enabled. The dashed line shows the relative speedup over the single-threaded version

Table 4 Effects of using different numbers of threads on level 1 for Case Study III

# 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

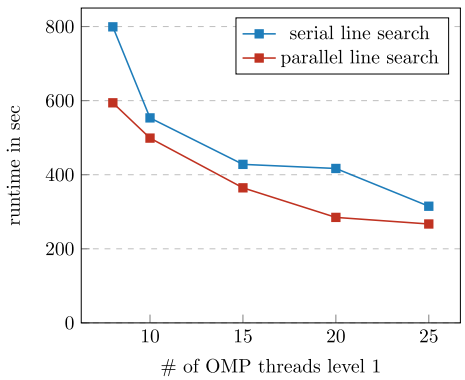


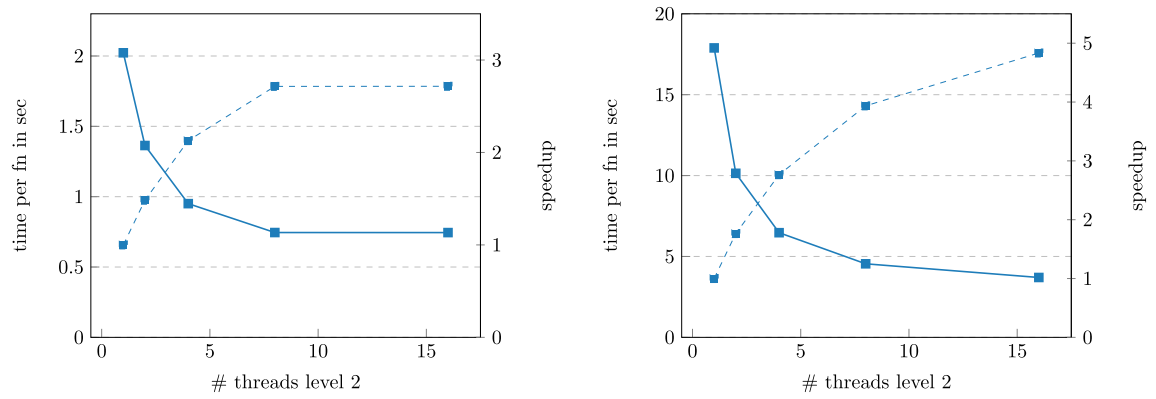
Fig. 6 Runtime for Case Study I, with and without parallel line search over varying numbers of threads on level 1. The number of threads on level 2 is fixed to 1. The posterior marginals of the latent parameters are approximated using the empirical Bayes integration strategy

ing visible for the latent parameter spaces of dimensions in the range of 10^4 and larger, especially for models with a three-dimensional structure.

We consider both examples of Case Study II in the performance analysis of Fig. 7. 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 spaces benefit more from level 2 parallelism, leading to a speedup of almost 5 using 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 speedup 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.

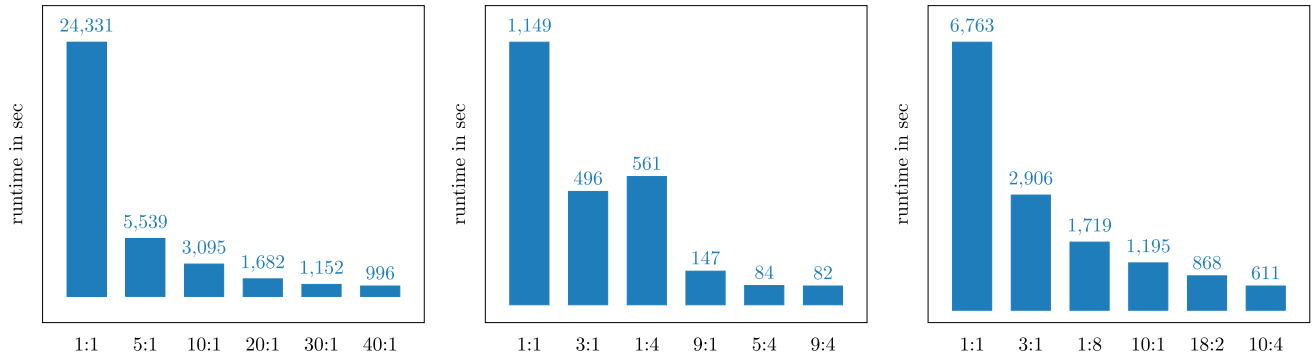
4.4 Leveraging combined parallelism

In this section we want to present results for combined parallelization strategies and discuss how to choose a favorable set up. The ideal approach clearly 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 to best utilize the available resources. We have found that hyperthreading often does not significantly increase the performance (or is even counter productive) 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 Figs. 5 & 7 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 Fig. 8, 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. This has from a theoretical, as well



(a) medium Case Study II (b) large Case Study II

Fig. 7 The solid line shows the total runtime in seconds divided by the number of function evaluations for different numbers of threads on level 2 with the number of level 1 threads fixed to one and without the parallel line search enabled. The dashed line shows the relative speedup over the single-threaded version



(a) Case Study I (b) medium Case Study II (c) large Case Study II

Fig. 8 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)

as an empirical point of view, 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 Discussion

In this paper we presented novel parallelization strategies for the INLA methodology. While INLA has always been using optimized sparse linear algebra operations to be computationally efficient, we have taken this a step further by introducing parallelism using OpenMP. This allows for the simultaneous execution of computationally expensive tasks which are distributed over a specified number of cores. Our approach contains two layers of parallelism. On the first layer we have parallelized independent function evaluations during INLA's optimization phase to determine and explore the posterior mode of the hyperparameters.

Each evaluation requires the Cholesky factorization, a computationally expensive task, of at least one sparse precision matrix whose size is tied to the potentially very high-dimensional latent parameter space of the model. We have additionally introduced a parallel line search routine using robust regression to parallelize and stabilize the search for each next step size, thus being able to lower the overall number of required iterations. We were also able to parallelize large parts of the computation of the posterior marginal variances of the latent parameters which require a computationally expensive partial matrix inversion. The state-of-the-art sparse solver PARDISO was included in INLA to handle the most expensive linear algebra operations and makes up the second layer of parallelism. PARDISO makes use of intelligent matrix reordering techniques that allow large parts of the otherwise sequential computations to be parallelized. Thus, PARDISO internally employs OpenMP for simultaneous operations within Cholesky factorizations, solutions to linear systems of equations and partial matrix inversions.

We empirically demonstrated the performance and scalability of the parallelization strategies on three case studies. Each of them uses a different type of model, hence posing different computational challenges. For all larger applications we observed speeds up 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. Given that larger multi-core computer architectures are becoming more and more accessible, these computational advancements allow users to perform inference using more complex models at higher resolution in shorter runtimes with INLA.

Acknowledgements We would like to thank Prof. D. Bolin and Dr. D. Rustand for their support with the different case studies. The work of L. Gaedke-Merzhäuser has been supported by the SNF SINERGIA Project No. CRSII5_189942.

Funding The authors have no relevant financial or non-financial interests to disclose.

References

Arisido, M.W., Gaetan, C., Zanchettin, D., Rubino, A.: A Bayesian hierarchical approach for spatial analysis of climate model bias in multi-model ensembles. Stoch. Environ. Res. Risk Assess. **31**(10), 2645–2657 (2017). https://doi.org/10.1007/s00477-017-1383-2

Ascher, U.M., Greif, C.: A first course on numerical methods. SIAM (2011). https://doi.org/10.1137/9780898719987

Atkinson, A.C., Riani, M., Riani, M.: Robust diagnostic regression analysis, Volume 2. Springer (2000). https://doi.org/10.1007/978-1-4612-1160-0

Bakka, H., Rue, H., Fuglstad, G.A., Riebler, A., Bolin, D., Illian, J., Krainski, E., Simpson, D., Lindgren, F.: Spatial modelling with R-INLA: a review. WIREs Comput. Stat. **10**(6), e1443 (2018). https://doi.org/10.1002/wics.1443

Batomen, B., Irving, H., Carabali, M., Carvalho, M.S., Ruggiero, E.D., Brown, P.: Vulnerable road-user deaths in Brazil: a Bayesian hierarchical model for spatial-temporal analysis. Int. J. Injury Cont. Safety Promot. (2020). https://doi.org/10.1080/17457300.2020.1818788

Baydin, A.G., Pearlmutter, B.A., Radul, A.A., Siskind, J.M.: Automatic differentiation in machine learning: a survey. J. Mach. Learn. Res. **18**, 1–43 (2018). https://doi.org/10.5555/3122009.3242010

Bhatt, S., Weiss, D., Cameron, E., Bisanzio, D., Mappin, B., Dalrymple, U., Battle, K., Moyes, C., Henry, A., Eckhoff, P., et al.: The effect of malaria control on plasmodium falciparum in Africa between 2000 and 2015. Nature **526**(7572), 207–211 (2015). https://doi.org/10.1038/nature15535

Bichot, C.-E., Siarry, P.: Graph partitioning. Wiley, Hobroken (2013). https://doi.org/10.1007/978-3-319-63962-8_312-1

Bollhöfer, M., Schenk, O., Janalik, R., Hamm, S., Gullapalli, K.: State-of-the-art sparse direct solvers. In Parallel algorithms in computational science and engineering, pp. 3–33. Springer. (2020) https://doi.org/10.1007/978-3-030-43736-7_1

Coll, M., Pennino, M.G., Steenbeek, J., Solé, J., Bellido, J.M.: Predicting marine species distributions: complementarity of food-web and bayesian hierarchical modelling approaches. Ecol. Modell. **405**, 86–101 (2019). https://doi.org/10.1016/j.ecolmodel.2019.05.005

Congdon, P.: Applied Bayesian modelling, Volume 595. Wiley, Hobroken (2014). https://doi.org/10.1002/9781118895047

Davis, T.A.: Direct methods for sparse linear systems. SIAM (2006). https://doi.org/10.1137/19780898718881

de Rivera, O.R., Blangiardo, M., López-Quílez, A., Martín-Sanz, I.: Species distribution modelling through Bayesian hierarchical approach. Theoret. Ecol. **12**(1), 49–59 (2019). https://doi.org/10.1007/s12080-018-0387-y

Demmel, J.W.: Applied numerical linear algebra. Soci. Ind. Appl. Math. https://doi.org/10.1137/19781611971446 (1997)

Diaz, J.M., Pophale, S., Hernandez, O., Bernholdt, D.E., Chandrasekaran, S. (2018) Openmp 4.5 validation and verification suite for device offload. In B. R. de Supinski, P. Valero-Lara, X. Martorell, S. Mateo Bellido, and J. Labarta (Eds.), Evolving OpenMP for Evolving Architectures, pp. 82–95. Springer, Cham https://www.openmp.org

Fattah, E.A., Niekerk, J.V., Rue, H.: Smart gradient - an adaptive technique for improving gradient estimation. Found. Data Sci. **4**(1), 123–136 (2022). https://doi.org/10.3934/fods.2021037

George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**(2), 345–363 (1973). https://doi.org/10.1137/0710032

George, A., Liu, J.W.: The evolution of the minimum degree ordering algorithm. SIAM Rev. **31**(1), 1–19 (1989). https://doi.org/10.1137/1031001

Heath, M.T., Ng, E., Peyton, B.W.: Parallel algorithms for sparse linear systems. SIAM Rev. **33**(3), 420–460 (1991). https://doi.org/10.1137/1033099

Henderson, R., Shimakura, S., Gorst, D.: Modeling spatial variation in leukemia survival data. J. Am. Stat. Assoc. **97**(460), 965–972 (2002). https://doi.org/10.1198/016214502388618753

Isaac, N.J., Jarzyna, M.A., Keil, P., Dambly, L.I., Boersch-Supan, P.H., Browning, E., Freeman, S.N., Golding, N., Guillera-Arroita, G., Henrys, P.A., et al.: Data integration for large-scale models of species distributions. Trend. Ecol. Evolut. **35**(1), 56–67 (2020). https://doi.org/10.1016/j.tree.2019.08.006

Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Scient. Comp. **20**(1), 359–392 (1998). https://doi.org/10.5555/305219.305248

Konstantinoudis, G., Padellini, T., Bennett, J., Davies, B., Ezzati, M., Blangiardo, M.: Long-term exposure to air-pollution and covid-

19 mortality in England: a hierarchical spatial analysis. Environ. Int. **146**, 106316 (2021). https://doi.org/10.1016/j.envint.2020.106316

Kontis, V., Bennett, J.E., Rashid, T., Parks, R.M., Pearson-Stuttard, J., Guillot, M., Asaria, P., Zhou, B., Battaglini, M., Corsetti, G., 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. Nat. Med. **26**(12), 1919–1928 (2020) https://www.nature.com/articles/s41591-020-1112-0

Krainski, E.T., Gómez-Rubio, V., Bakka, H., Lenzi, A., Castro-Camilio, D., Simpson, D., Lindgren, F., Rue, H. (2018, December) Advanced spatial modeling with stochastic partial differential equations using R and INLA. CRC press, Cambridge. Github version www.r-inla.org/spde-book

LeVeque, R.J.: Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems. SIAM **10**(1137/1), 9780898717839 (2007)

Li, S., Ahmed, S., Klimeck, G., Darve, E.: Computing entries of the inverse of a sparse matrix using the FIND algorithm. J. Comput. Phys. **227**(22), 9408–9427 (2008). https://doi.org/10.1016/j.jcp.2008.06.033

Lillini, R., Tittarelli, A., Bertoldi, M., Ritchie, D., Katalinic, A., Pritzkuleit, R., Launoy, G., Launay, L., Guillaume, E., Žagar, T., et al.: Water and soil pollution: ecological environmental study methodologies useful for public health projects. a literature review. Rev. Environ. Contaminat. Toxicol. **256**, 179–214 (2021). https://doi.org/10.1007/398_2020_58

Lindenmayer, D., Taylor, C., Blanchard, W.: Empirical analyses of the factors influencing fire severity in southeastern australia. Ecosphere **12**(8), e03721 (2021). https://doi.org/10.1002/ecs2.3721

Lindgren, F., Bolin, D., Rue, H.: The SPDE approach for gaussian and non-gaussian fields: 10 years and still running. Spat. Stat. (2022). https://doi.org/10.1016/j.spasta.2022.100599

Lindgren, F., Rue, H., Lindström, J.: An explicit link between gaussian fields and gaussian markov random fields: the stochastic partial differential equation approach. J. Royal Stat. Soc.: Series B (Stat. Methodol.) **73**(4), 423–498 (2011). https://doi.org/10.1111/j.1467-9868.2011.00777.x

Lu, N., Liang, S., Huang, G., Qin, J., Yao, L., Wang, D., Yang, K.: Hierarchical Bayesian space-time estimation of monthly maximum and minimum surface air temperature. Remote Sens. Environ. **211**, 48–58 (2018). https://doi.org/10.1016/j.rse.2018.04.006

Martínez-Minaya, J., Cameletti, M., Conesa, D., Pennino, M.G.: Species distribution modeling: a statistical review with focus in spatio-temporal issues. Stoch. Environ. Res. Risk Assess. **32**(11), 3227–3244 (2018). https://doi.org/10.1007/s00477-018-1548-7

Martins, T.G., Simpson, D., Lindgren, F., Rue, H.: Bayesian computing with inla: new features. Comput. Stat. & Data Anal. **67**, 68–83 (2013). https://doi.org/10.1016/j.csda.2013.04.014

Mejia, A.F., Yue, Y., Bolin, D., Lindgren, F., Lindquist, M.A.: A bayesian general linear modeling approach to cortical surface FMRI data analysis. J. Am. Stat. Assoc. **115**(530), 501–520 (2020). https://doi.org/10.1080/01621459.2019.1611582

Mielke, K.P., Claassen, T., Busana, M., Heskes, T., Huijbregts, M.A., Koffijberg, K., Schipper, A.M.: Disentangling drivers of spatial autocorrelation in species distribution models. Ecography **43**(12), 1741–1751 (2020). https://doi.org/10.1111/ecog.05134

Nocedal, J., Wright, S.: Numerical optimization. Springer, Berlin (2006). https://doi.org/10.1007/978-0-387-40065-5

Opitz, T. (2017). Latent gaussian modeling and inla: A review with focus on space-time applications. J. de la société française de statistique 158(3), 62–85. https://hal.archives-ouvertes.fr/hal-01394974

Pan, V., Reif, J. (1985) Efficient parallel solution of linear systems. In Proceedings of the seventeenth annual ACM symposium on Theory of computing, pp. 143–152. https://doi.org/10.1145/22145.22161

PARDISO (2022). Version 7.2. Lugano, Switzerland: Panua Technologies. http://www.panua.ch

Pimont, F., Fargeon, H., Opitz, T., Ruffault, J., Barbero, R., Martin-StPaul, N., Rigolot, E., Rivière, M., Dupuy, J.-L.: Prediction of regional wildfire activity in the probabilistic bayesian framework of firelihood. Ecol. Appl. **31**(5), e02316 (2021). https://doi.org/10.1002/eap.2316

Pinto, G., Rousseu, F., Niklasson, M., Drobyshev, I.: Effects of human-related and biotic landscape features on the occurrence and size of modern forest fires in Sweden. Agricult. Forest Meteorol. **291**, 108084 (2020). https://doi.org/10.1016/j.agrformet.2020.108084

Rousseeuw, P.J., Leroy, A.M.: Robust regression and outlier detection, Volume 589. Wiley, Hobroken (2005). https://doi.org/10.1002/0471725382

Rue, H., Held, L.: Gaussian Markov random fields: theory and applications. CRC Press, Cambridge (2005). https://doi.org/10.1201/9780203492024

Rue, H., Martino, S.: Approximate bayesian inference for hierarchical gaussian markov random field models. J. Stat. Plann. Infer. **137**(10), 3177–3192 (2007). https://doi.org/10.1016/j.jspi.2006.07.016

Rue, H., Martino, S., Chopin, N.: Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. J. Royal Stat. Soc.: Series b (Stat. Methodol.) **71**(2), 319–392 (2009). https://doi.org/10.1111/j.1467-9868.2008.00700.x

Rue, H., Riebler, A., Sørbye, S.H., Illian, J.B., Simpson, D.P., Lindgren, F.K.: Bayesian computing with INLA: a review. Ann. Rev. Stat. Appl. **4**, 395–421 (2017). https://doi.org/10.1146/annurev-statistics-060116-054045

Rustand, D., Van Niekerk, J., Krainski, E.T., Rue, H., Proust-Lima, C. (2022) Fast and flexible inference approach for joint models of multivariate longitudinal and survival data using integrated nested Laplace approximations. arxiv:2203.06256

Saad, Y.: Iterative methods for sparse linear systems. SIAM **10**(1137/1), 9780898718003 (2003)

Sanyal, S., Rochereau, T., Maesano, C.N., Com-Ruelle, L., Annesi-Maesano, I.: Long-term effect of outdoor air pollution on mortality and morbidity: a 12-year follow-up study for metropolitan france. Int. J. Environ. Res. Public Health. **15**(11), 2487 (2018). https://doi.org/10.3390/ijerph15112487

Shaddick, G., Thomas, M.L., Amini, H., Broday, D., Cohen, A., Frostad, J., Green, A., Gumy, S., Liu, Y., Martin, R.V., et al.: Data integration for the assessment of population exposure to ambient air pollution for global burden of disease assessment. Environ. Sci Technol. **52**(16), 9069–9078 (2018). https://doi.org/10.1021/acs.est.8b02864

Spencer, D., Yue, Y.R., Bolin, D., Ryan, S., Mejia, A.F.: Spatial bayesian GLM on the cortical surface produces reliable task activations in individuals and groups. NeuroImage (2022). https://doi.org/10.1016/j.neuroimage.2022.118908

Takahashi, K.: Formation of sparse bus impedance matrix and its application to short circuit study. In Proc. PICA Conference, June, (1973)

Toledo, S. (2003). Taucs: a library of sparse linear solvers. https://www.tau.ac.il/~stoledo/taucs/

Van Merriënboer, B., Breuleux, O., Bergeron, A., Lamblin, P. (2018) Automatic differentiation in ML: Where we are and where we should be going. Advances in neural information processing systems 31. https://proceedings.neurips.cc/paper/2018/file/770f8e448d07586afbf77bb59f698587-Paper.pdf

Van Niekerk, J., Bakka, H., Rue, H., Schenk, O.: New frontiers in Bayesian modeling using the INLA package in R. J. Stat. Softw. **100**(2), 1–28 (2021). https://doi.org/10.18637/jss.v100.i02

Van Niekerk, J., Bakka, H., Rue, H., Schenk, O.: New frontiers in Bayesian modeling using the INLA package in R. J. Stat. Softw. 100(2), 1–28 (2021).https://doi.org/10.18637/jss.v100.i02

Van Niekerk, J., E. Krainski, D. Rustand, and H. Rue (2022). A new avenue for bayesian inference with INLA. arXiv preprint arXiv:2204.06797

Yannakakis, M.: Computing the minimum fill-in is np-complete. SIAM J. Algebr. Discr. Meth. **2**(1), 77–79 (1981). https://doi.org/10.1137/0602010