Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ORIGINAL ARTICLE

Investigating mesh based approximation methods for the normalization constant in the log Gaussian Cox process likelihood

Martin Jullum

[1]Norwegian Computing Center, Oslo, Norway

Correspondence
*Martin Jullum, Norwegian Computing Center, P.O. Box 114 Blindern, NO-0314 Oslo, Norway. Email: jullum@nr.no

Funding Information
This research was supported by the The Research Council of Norway, grant 240838 "Model selection and model verification for point processes"

The Log Gaussian Cox Process (LGCP) is a frequently applied method for modeling point pattern data. The normalization constant of the LGCP likelihood involves an integral over a latent field. That integral is computationally costly to compute, making it troublesome to perform inference with standard methods. The so-called SPDE-INLA framework enables fast approximate inference for a range of hierarchical models, where a key component is to approximate the latent field to lie on a triangulated mesh. Recent research has made it possible to fit LGCP models with this framework using an approximate integration method to compute the troublesome integral. We carefully describe several alternative variants of that approximate integration method and derive an analytical formula for the integral in question which actually is exact under the triangular mesh assumption used by SPDE-INLA. We compare the different integration strategies through a comprehensive simulation study and find that the analytical formula is often more accurate, but not always. Among the approximate integration methods, we recommend a simple extension to a method implemented in an R-package for fitting LGCP models.

KEYWORDS:
Cox processes, Deterministic integration scheme, Likelihood normalization constant, Triangular mesh, Simulation study

1 | INTRODUCTION

Several phenomena in the world we live in may be viewed as stemming from a spatial point process. Examples range from the appearances of earthquakes (Eberhard, Zechar, & Wiemer 2012), terror attacks and crimes (Mohler, Short, Brantingham, Schoenberg, & Tita 2011), the locations of certain plant and animal species both at sea and on land (Jullum, Thorarinsdottir, & Bachl 2020; Waller et al. 2011; Yuan et al. 2017), all the way to stars and planets in the outer space (Babu & Feigelson 1996; Stoica, Tempel, Liivamägi, Castellan, & Saar 2014).

The most fundamental type of point process model is the Poisson point process. Assuming an observation domain $\Omega \subset \mathbb{R}^2$, this model is fully determined by a deterministic intensity function $\lambda : \Omega \mapsto [0, \infty)$. Under this model, the number of points $N(B)$ in any Borel set $B \subseteq \Omega$ is Poisson distributed with mean $\mu(B) = \int_B \lambda(\mathbf{s})\, d\mathbf{s}$, and independent of any other non-overlapping Borel set B^*. For this model, the likelihood of an observed point process (point pattern) Y with observations at locations $\mathbf{s}_1, \ldots, \mathbf{s}_n$, takes the form

$$p(Y|\lambda) = \exp(|\Omega| - \int_\Omega \lambda(\mathbf{s})\, d\mathbf{s}) \prod_{i=1}^{n} \lambda(\mathbf{s}_i), \tag{1}$$

where $\exp(|\Omega| - \int_\Omega \lambda(\mathbf{s})\, d\mathbf{s})$ is the normalization constant. Carrying out maximum likelihood type of inference based on (1) is not too hard when $\lambda(\mathbf{s})$ is restricted to be constant (homogeneous Poisson process), or takes a simplified (possibly covariate dependent) parametric form with few

parameters such that the integral $\int_\Omega \lambda(\mathbf{s})\,\mathrm{d}\mathbf{s}$ can be solved analytically. In general, performing traditional inference based on (1) requires numerical integration to compute the normalization constant, which is often computationally costly.

To appropriately model natural phenomenons as those mentioned above, standard (homogeneous or inhomogeneous) Poisson processes are typically not sufficient to capture the complexities involved. The Cox process (Cox 1955) is a natural extension of the Poisson process, where the intensity function λ is no longer deterministic, but rather a (latent) random field. The most popular type of Cox Process model is the Log Gaussian Cox Process (LGCP), where $\log(\lambda(\mathbf{s})) = \mathsf{Z}(\mathbf{s})$, and Z is a Gaussian random field. Even with the simplifying and well-studied structure of Gaussian fields, the fitting of such models is troublesome and has for a long time required time consuming MCMC procedures (Guttorp & Thorarinsdottir 2012). This is particularly related to the computational complexity of the normalization constant, where the integral $\int_\Omega \lambda(\mathbf{s})\,\mathrm{d}\mathbf{s}$ is the troublesome part. However, recently Simpson, Illian, Lindgren, Sørbye, and Rue (2016) made approximate inference for LGCP models computationally inexpensive. The procedure relies on the stochastic partial differential equation (SPDE) approach of the integrated nested Laplace approximation method (INLA) (Lindgren, Rue, & Lindström 2011; Rue, Martino, & Chopin 2009). The INLA framework allows for computationally feasible approximate Bayesian inference for hierarchical models with a discrete latent Gaussian Markov random field (GMRF), by utilizing several (nested) Laplace approximations and clever numerical integration schemes (Rue et al. 2009). The SPDE approach essentially extends this to continuous latent Gaussian fields by approximating the (continuous) latent field Z by a triangular mesh, which through a certain SPDE solution may be mapped to a GMRF that can be handled by INLA. The SPDE-INLA framework and the associated `INLA` `R`-package cannot, however, handle the conditional likelihood (1) out of the box. The solution of Simpson et al. (2016) is to apply a deterministic integration method to the integral $\int \lambda(\mathbf{s})\,\mathrm{d}\mathbf{s}$, which by some re-structuring transforms the full conditional likelihood formula to a product of conditionally independent Poisson observations. That approximate likelihood formula may be handled efficiently within the INLA framework.

There are numerous variants of the deterministic integration scheme outlined in Simpson et al. (2016), mainly differing in the way they assign weights to the deterministic integration points. In this paper, we show that under the triangular mesh approximation underlying the approach of Simpson et al. (2016), one may, in fact, derive an exact analytical formula for the integral $\int \lambda(\mathbf{s})\,\mathrm{d}\mathbf{s}$, and thereby the normalization constant in (1). This suggests that more accurate inference than that achieved by Simpson et al. (2016) is possible. We perform a comprehensive simulation study investigating the performance of the different integration schemes, compared to the true value of the integral and the analytical formula which is exact under the triangular mesh assumption. We find that the analytical formula is typically the most accurate, but that it is preferable to use a deterministic integration method in some situations. Moreover, which of the deterministic integration schemes that perform best varies greatly depending on the specification of the mesh and underlying latent field to integrate over. We do, however, recommend a stable and decent performing method being a simple extension to a method implemented in a user-friendly software package for fitting LGCP models. All methods have been coded in the `R` programming language. The source code for the simulation experiments and our implementations and usage of all the integration methods, in addition to user-friendly tables with summaries of all simulation results and permutation tests performed in the study, are publicly available through the GitHub repository github.com/martinju/LGCP-normConst-simulations.

The rest of the paper is organized as follows: Section 2 contains a description of the likelihood approximation method used by Simpson et al. (2016), in addition to an analytical formula for the integral in question, which is exact under the triangular mesh assumption used by SPDE-INLA. In Section 3, we present four different variants of the deterministic integration method and discuss and illustrate their differences. Section 4 contains a comprehensive simulation study comparing the different integration methods, and discussions of the results we find. Section 5 contains a summary and some concluding remarks. Appendix A contains a table with pre-computation timings for the different integration methods. Supporting information following this paper contains the mathematical derivation leading to the analytical formula for the integral in Section 2.2.

2 | LGCP APPROXIMATION METHODS

In this section we lay out the assumptions and derivations behind the approach of Simpson et al. (2016), and present an analytical formula for the integral in question being exact under the triangular mesh assumption. Before we turn to the two approaches, we provide some necessary background. Both types of methods we shall explore rely on a finite element method (FEM) representation of the latent field. That is, we approximate the original latent field $\mathsf{Z}(\mathbf{s})$ by $\mathsf{Z}^*(\mathbf{s})$ for $\mathbf{s} \in \Omega$, where $\mathsf{Z}^*(\mathbf{s})$ lives on a triangulated mesh with q nodes:

$$\mathsf{Z}^*(\mathbf{s}) = \sum_{\mathsf{j}=1}^{\mathsf{q}} \mathsf{z}_\mathsf{j} \phi_\mathsf{j}(\mathbf{s}). \tag{2}$$

Here $\mathbf{z} = (\mathsf{z}_1, \ldots, \mathsf{z}_\mathsf{q})^\top$ is a multivariate Gaussian variable where each dimension represents the value of the field in one mesh node, and $\{\phi_\mathsf{j}(\mathbf{s})\}_{\mathsf{j}=1}^{\mathsf{q}}$ is a set of deterministic linearly independent basis functions which are piecewise linear between the nodes. More specifically, the basis functions are chosen such that $\phi_\mathsf{j}(\mathbf{s})$ is 1 at mesh node j and 0 at all other nodes. This is illustrated in dimension 1 in Figure A1. Since the ϕ_j's are deterministic, the behavior of $\mathsf{Z}^*(\mathbf{s})$ is completely determined by $\mathbf{z}$. An illustration of how the mesh approximates the original 2D field is provided in Figure A2.

[**FIGURE** 1 about here.]

[**FIGURE** 2 about here.]

For details on how to construct the triangular mesh and how this representation provides a framework for approximate Bayesian inference through the SPDE-INLA framework, we refer to Cameletti, Lindgren, Simpson, and Rue (2013); Lindgren et al. (2011).

2.1 | The general approach of Simpson et al.

Under the approximation in (2), the likelihood in (1) can be written as

$$p(Y|Z^*) = \exp(|\Omega| - \int_\Omega \exp(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}))\, d\mathbf{s}) \prod_{i=1}^{n} \exp(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}_i)), \tag{3}$$

where the component $\exp(|\Omega| - \int_\Omega \exp(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s})))$ is the normalization constant. The associated log-likelihood takes the form

$$\ell(Y) = |\Omega| - \int_\Omega \exp(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}))\, d\mathbf{s} + \sum_{i=1}^{n} \sum_{j=1}^{q} z_j \phi_j(\mathbf{s}_i). \tag{4}$$

While the data dependent final sum in (4) is easy to compute directly, and $|\Omega|$ is a fixed (and thus irrelevant) constant, inference requires computation of the integral

$$\int_\Omega \exp(Z^*(\mathbf{s}))\, d\mathbf{s} = \int_\Omega \exp(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}))\, d\mathbf{s}. \tag{5}$$

Although $Z^*(\mathbf{s})$ takes a simple linear form and is easy to integrate over, the integral over its exponential is not straightforward to compute. It is, however, computationally fast to evaluate the integrand in (5) at specific locations. Thus, Simpson et al. (2016) suggests using a fixed set of integration points $\tilde{\mathbf{s}}_j, j = 1, \ldots, k$, with associated weights $w_j, j = 1, \ldots, k$, and proposes a deterministic integration method of the form

$$\int \exp(Z^*(\mathbf{s}))\, d\mathbf{s} = \int_\Omega \exp(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}))\, d\mathbf{s} \approx \sum_{j=1}^{k} w_j \exp\{Z^*(\tilde{\mathbf{s}}_j)\}. \tag{6}$$

Using this approximation, the full likelihood is identical to a likelihood of $n + k$ independent Poisson pseudo-observations. More specifically, the approximate likelihood takes the form

$$p(Y|Z^*) \approx \exp(|\Omega|) \prod_{j=1}^{n+k} \eta_j^{y_j^*} \exp(-\alpha_j \eta_j), \tag{7}$$

where $\mathbf{y}^* = (\mathbf{0}_{1\times r}, \mathbf{1}_{1\times n})^\top, \log(\boldsymbol{\eta}) = (\mathbf{z}^\top A_1^\top, \mathbf{z}^\top A_2^\top)^\top$ and $\boldsymbol{\alpha} = (w_1, \ldots, w_k, \mathbf{0}_{1\times n})^\top$, where $[A_1]_{rj} = \phi_j(\tilde{\mathbf{s}}_r)$ and $[A_2]_{rj} = \phi_j(\mathbf{s}_r)$. See Simpson et al. (2016, p. 53) for further details.

In order to fully specify the method, one must define a method for selecting integration points and assign weights to these points. As integration points and their weights are direct ingredients in the likelihood formula, the approximation quality depends on the method for assigning the weights. Simpson et al. (2016) describes *one* such method, while other variants are also natural and currently in use. Several such variants will be introduced and discussed in Section 3.

2.2 | The mesh-exact integration method

Referring to the approximate log-likelihood in (4), Simpson et al. (2016) states that "While the continuously specified stochastic partial differential equation models allow us to compute the sum exactly, we must approximate the integral by another sum." This is actually incorrect, as the integral in (5) may also be computed exactly — it is just a bit tedious. We shall here present an analytical formula under the rather weak assumption that the observation domain Ω is piecewise linear. This formula is an exact solution to (5), but an approximation to $\int \exp(Z(\mathbf{s}))\, d\mathbf{s}$, and we will therefore refer to this as the mesh-exact integration method.

Due to the somewhat lengthy mathematical derivations and bookkeeping required, we restrict ourselves to present the main ideas and the final formula here. The full derivation is provided in the supporting information following this paper.

We first define a few simplifying quantities. Let the complete mesh be written as $\bigcup_i^K T_i^{(M)}$, where $T_1^{(M)}, \ldots, T_K^{(M)}$ are the K disjoint triangles constituting the complete mesh. Now, as the observation domain Ω is piecewise linear, it may be written as a finite union of disjoint triangles, each of which is fully contained in exactly *one* mesh triangle $T_i^{(M)}$. That is, we may write

$$\Omega = \bigcup_i^K \bigcup_k^{L_i} T_{ik}, \tag{8}$$

where for each $i = 1, \dots, K$, the $T_{i1}, \dots, T_{iL_i}$ are the L_i disjoint triangles whose union equals the part of the observation domain which falls in mesh triangle $T_i^{(M)}$. The key here is that, instead of integrating over Ω directly, we can integrate over each of the T_{ik}-triangles, where the integrand takes simpler log-linear forms, and then sum all these contributions in the end. Doing that, shows that

$$\int_{\Omega} \exp(Z^*(\mathbf{s}))\, d\mathbf{s} = \sum_{i=1}^{K} \sum_{k=1}^{L_i} \int_{T_{ik}} \exp\left(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}) \right) d\mathbf{s}, \tag{9}$$

where

$$\int_{T_{ik}} \exp\left(\sum_{j=1}^{q} z_j \phi_j(\mathbf{s}) \right) d\mathbf{s} = \begin{cases} |J_{g,ik}| \frac{\exp(\alpha_{ik}^*)}{2}, & \text{if } \beta_{ik}^* = \gamma_{ik}^* = 0, \\ |J_{g,ik}| \frac{\exp(\alpha_{ik}^*)}{(\gamma_{ik}^*)^2} \left[\exp(\gamma_{ik}^*) - 1 - \gamma_{ik}^* \right], & \text{if } \beta_{ik}^* = 0, \gamma_{ik}^* \neq 0, \\ |J_{g,ik}| \frac{\exp(\alpha_{ik}^*)}{(\beta_{ik}^*)^2} \left[\exp(\beta_{ik}^*) - 1 - \beta_{ik}^* \right], & \text{if } \beta_{ik}^* \neq 0, \gamma_{ik}^* = 0, \\ |J_{g,ik}| \frac{\exp(\alpha_{ik}^*)}{(\beta_{ik}^*)^2} \left[1 + \exp(\beta_{ik}^*)(\beta_{ik}^* - 1) \right], & \text{if } \beta_{ik}^* = \gamma_{ik}^* \neq 0, \\ |J_{g,ik}| \frac{\exp(\alpha_{ik}^*)}{\beta_{ik}^* \gamma_{ik}^* (\gamma_{ik}^* - \beta_{ik}^*)} \left[\beta_{ik}^* (\exp(\gamma_{ik}^*) - 1) - \gamma_{ik}^* (\exp(\beta_{ik}^*) - 1) \right], & \text{otherwise.} \end{cases} \tag{10}$$

Here $\alpha_{ik}^*, \beta_{ik}^*$ and γ_{ik}^* are linear combinations of the elements of the latent Gaussian variable $\mathbf{z}$ corresponding to the three corners in triangle $T_i^{(M)}$, while $|J_{g,ik}|$ is the Jacobian determinant of a matrix related to the location of T_{ik}. Thus, the final expression is a sum of functions of linear combinations of the latent Gaussian variable $\mathbf{z}$.

3 | VARIATIONS OF THE DETERMINISTIC INTEGRATION METHOD

As mentioned, there are various ways to choose and assign weights to integration points for the deterministic integration method. When it comes to choosing the integration points, Simpson et al. (2016) suggests using the mesh nodes as integration points. This is natural for two reasons: (a) The triangulated field $Z^*(\mathbf{s})$ is completely determined by the field value in the mesh nodes $\mathbf{z}$, and can be mapped to the mesh nodes by linear combinations. If the integrand had been linear, any integration weight assigned to non-mesh nodes could be mapped exactly to the mesh nodes. Even if the integrand is not linear, it may not be very far from linear within every triangle. (b) Due to the form of (2), the integrand takes its extreme points at the mesh nodes. Using these as integration points ensures that "peaks" and "holes" in $Z^*(\mathbf{s})$ are accounted for in the integration routine.

We shall consider four different variants of the deterministic integration method, all of which uses the mesh nodes as integration points, such that $m = q$[1]:

1. **Dual mesh**: This approach creates a so-called dual mesh based on the triangles in the mesh, where every mesh node is connected to a single dual mesh polygon. The integration point weights are given by the area of the polygons within Ω.

2. **Dual mesh with extra points**: This approach is identical to the dual mesh approach, but works on a refined mesh with (temporary) extra points along the boundary of Ω.

3. **Barycentric point spreading approach**: This approach spreads a large number of points "evenly" in every triangle, maps their weights to the triangle corners, and use the sum of these to determine the weights of the mesh nodes.

4. **Voronoi tesselation**: This approach is conceptually identical to the dual mesh approach, but uses the Voronoi tessellation of the mesh nodes instead of the triangle based "dual mesh".

Below we describe these four methods in detail and illustrate how they assign weights to integration points. As will become clear, some of these methods give exactly or close to exactly the same weights for integration points where all neighboring mesh triangles are completely covered by the observation domain Ω (hereafter called *internal* integration points), while they behave differently for points where this is not the case (hereafter called *boundary* integration points). If the observation domain Ω coincides exactly with the edges in the triangular mesh, there are no boundary points, such that methods with the same weight for internal points give exactly the same result. While the mesh may be constructed to coincide with Ω in certain situations, there are many situations where this may not be appropriate, particularly when the observation domain is complex. This is the case in most non-complete survey-based application areas; see e.g. Bachl, Lindgren, Borchers, and Illian (2019) for examples in ecology. In the below description, we consider both internal and boundary points.

[1] One of the two variants of the "Dual mesh with extra points" method also uses additional integration points.

3.1 | The dual mesh approach

The dual mesh approach assigns weight to the mesh nodes based on the area of polygons in a dual mesh associated with the original triangulated mesh. Each mesh node is associated with a single polygon in the dual mesh. The polygon is formed by connecting the midpoints of all the edges connected to the mesh node in question and the centroids of the triangles where that mesh node is a corner. The weight is equal to the area of the intersection of this polygon and Ω. For internal integration points, this is equal to the sum of $1/3$ of the areas of the triangles where the integration point is a corner. For boundary points, this is not necessarily the case. Note that mesh nodes which lie outside Ω may also receive a non-zero weight provided the dual mesh polygon of the node intersects with Ω. The method is illustrated in Figure A3, and described briefly in Simpson et al. (2016, Sec. 5) and Krainski et al. (2018, Ch. 2.2.2). In this paper, we have directly used the implementation following Krainski et al. (2018), available at `http://www.r-inla.org/spde-book`.

[**FIGURE** 3 about here.]

3.2 | The dual mesh approach with extra integration points

As mentioned above, the dual mesh approach assigns $1/3$ of the area of the connected triangles to internal integration points. Thus, the dual mesh approach is simple for internal integration points, while it is not for boundary points. An alternative way to handle the latter is to add extra integration points where the observation domain crosses an edge of the mesh or makes a direction change. With these points in place, one can construct sub triangles between these points and the original mesh nodes which lie entirely within the observation domain and do not cross the original mesh edges. Now, the observation domain is a union of disjoint triangles, and their corner points (the integration points) can receive $1/3$ of their area as integration weight, exactly as for the internal integration points. This method was implemented in a previous version of the `R`-package `inlabru` (v 2.1.7). `inlabru` is built on top of the original `INLA` `R`-package, and provides simplified access to Bayesian inference for spatial point processes using LGCP. The implementation of this method had some stability issues and was (probably for this reason) replaced by the method to be explained in Section 3.3.

A drawback of this approach is that it increases the number of integration points and consequently the computational burden of approximating the integral. A variant of the approach, proposed by Yuan et al. (2017, Section 4.2. and Supplement C), is to map the weight for the extra integration points back to the original mesh nodes. This is done using the basis functions $\phi_1, \ldots, \phi_{\mathrm{q}}$ as mapping functions, cf. (2). Thus, a mesh node which is a corner of a triangle containing at least one extra integration point will receive additional weight. This additional procedure ensures that the integration points again corresponds to the q original mesh nodes. For the present paper, we have re-implemented a stable version of the dual mesh approach with extra integration points from `inlabru` with this additional mapping back to the mesh nodes. Figure A4 shows how internal integration points gets the same weight as for the original dual mesh approach, while additional points and triangles are constructed for the boundary point.

[**FIGURE** 4 about here.]

3.3 | The Barycentric point spreading approach

Starting from v 2.1.8 of `inlabru`, a new variant of the deterministic integration method was implemented. That method is still in use as of this writing (v 2.1.12). The idea behind the method is to first spread a large number of points (in the `inlabru` implementation, this has been hard-coded to be 100), in every triangle. Each of these points then receives a weight equal to $(\text{Area of triangle})/(\#\text{points spread in triangle})$. All points within Ω are then mapped to the mesh nodes using the basis functions $(\phi_1, \ldots, \phi_{\mathrm{q}})$, cf. (2). The sum of these mapped weights gives the total weight for the integration point. The points are spread "evenly" in the triangle with reference to a Barycentric coordinate system for every triangle. It is important to note that the points are only used to obtain weights for the original mesh nodes, just like the other methods in the present section. Hence, they are not used to evaluate the integrand. In this paper, we have modified the implementation in `inlabru` v 2.1.12 to allow for different numbers of points being spread in each triangle. Figure A5 shows how the points are spread within the triangles and which points are mapped to the corresponding integration point. For internal integration points, this method may be viewed as an approximation to the "dual mesh" and "dual mesh with extra integration points mapped back to the mesh nodes" approaches, and yields similar integration values.

[**FIGURE** 5 about here.]

3.4 | The Voronoi tessellation approach

The recommended way to construct triangular meshes to be used with the SPDE-INLA approach is to use a so-called (constrained) Delaunay triangulation. Such a triangulation have a direct relationship to the so-called Voronoi tessellation as the circumcenters of the Delaunay triangles are nodes of the Voronoi tessellation. Just like the dual mesh polygons in Section 3.1, the Voronoi tessellation is a collection of connected, non-overlapping polygon tiles, one for every mesh node. The Voronoi tile/polygon of a mesh node consists of the region of locations that are closer

to that mesh node, than to any other mesh nodes. Thus, contrary to the dual mesh polygons, the Voronoi tessellation only depends on the mesh nodes, ignoring which triangles they belong to. If all triangles had exactly the same shape and size, then the dual mesh polygons and the Voronoi tessellation would be identical. Just like the dual mesh approach, the present approach uses the area of the Voronoi tile/polygon as the weight for the associated mesh node. In the present context, this integration strategy was introduced in an old version of a tutorial to SPDE-INLA[2]. The implementation used in this paper builds on code from that tutorial (relying on the `deldir` `R`-package). Figure A6 shows the Voronoi tessellation of the internal and boundary points.

[**FIGURE** 6 about here.]

4 | SIMULATION STUDY

In this section, we describe and perform a comprehensive simulation aiming at investigating which of the aforementioned methods that give the smallest approximation error in different situations and thereby provide the best approximation for the normalization constant in (1). Before going into the specifics of the simulations experiments and their results, we describe how we approximate the true latent field $Z(\mathbf{s})$.

4.1 | Latent field approximation

As mentioned, the SPDE approach is based on a FEM approximation of the (unknown) latent field. In this simulation case, we do however need to approximate sampled realizations of the latent field $Z(\mathbf{s})$ using FEM. For fixed functions, the FEM approximation of $Z(\mathbf{s})$ is the least-squares approximation within the function space spanned by the basis functions $\phi_0, \dots, \phi_q$ (Langtangen & Mardal 2016, Chapter 3.). That corresponds to minimizing the inner product

$$\langle Z - Z^*, Z - Z^* \rangle, \tag{11}$$

where $\langle g, h \rangle = \int g(\mathbf{s})h(\mathbf{s})\,d\mathbf{s}$. Recalling that $Z^*(\mathbf{s}) = \sum_{j=1}^{q} z_j \phi_j(\mathbf{s})$, it can be shown that the minima of (11) is the solution of a linear equation system $B\mathbf{z} = \mathbf{b}$, where B is a $q \times q$ dimensional matrix with elements $B_{i,j} = \langle \phi_i, \phi_j \rangle, i, j = 1, \dots q$, and $\mathbf{b}$ is a vector of length q with elements $b_i = \langle Z, \phi_i \rangle, i = 1, \dots, q$, see e.g. Langtangen and Mardal (2016, Chapter 3.). The B matrix is readily available from the INLA software, while $\mathbf{b}$ can be quickly and accurately computed with numerical integration.

4.2 | Simulation study specification

Each integral approximation method (the mesh-exact approach in Section 2.2) and the different variations of the deterministic integration method in Section 3 has been applied to compute $\int_\Omega \exp(Z^*(\mathbf{s}))\,d\mathbf{s}$. The resulting integral values are then compared to the integral over the true latent field $\int_\Omega \exp(Z(\mathbf{s}))\,d\mathbf{s}$, which we can compute exactly in the simulated setting described below.

For *every* combination of the below parameter specifications for the observation domain Ω, the sampling distribution of $Z(\mathbf{s})$, and the specification of the mesh, a total of $K = 10\,000$ samples are drawn and their integrals are computed. The simulation specifications are as follows:

- The true Gaussian field has mean $\mu = 0$ (see Remark 1 for why this suffices), and takes a Matèrn covariance function of the form

$$\frac{\sigma^2}{2^{\nu-1}\Gamma(\nu)}(\kappa\|\mathbf{s}-\mathbf{t}\|)^\nu K_\nu(\kappa\|\mathbf{s}-\mathbf{t}\|),$$

  where $\mathbf{s}, \mathbf{t} \in \Omega, \nu > 0$ is a smoothing parameter, K_ν is the modified Bessel function of the second kind, $\kappa > 0$ is a scaling parameter and σ^2 is the marginal variance. Instead of working with κ and ν, it is convenient to work with a range parameter $r = \sqrt{8\nu}/\kappa$ corresponding to the distance at which spatial correlation is "small" (slightly higher than 0.1, see Blangiardo and Cameletti (2015, Sec. 6.5)). We set $\nu = 1$, and let r vary among $0.25, 0.5, 0.75, 1, 1.5, 2, 3, 4, 6$, and 8. In addition, σ^2 will vary among $0.1, 0.2, 0.5$, and 1.

- Since the methods may perform differently close to the boundary of Ω, we run simulations with four different complexities of Ω as illustrated in Figure A7, hereafter referred to as Ωs with number of sub domains equal to respectively, 1, 2, 8 and 32. All observation domains are bounded by a square box A ranging from 0 to 16 in both the x and y direction.

- We specify meshes with different levels of refinement, all of which extends slightly beyond A. The constructed meshes are shown in Figure A8, and has respectively 126, 183, 359, 1179, 4324, 16429 mesh nodes whose basis functions give nonzero output for some point within A and thus affects the approximation of $Z^*(\mathbf{s})$.

[2]The Voronoi tessellation concept was removed from the most recent versions of the tutorial. A previous version of the tutorial is as of this writing (January 2020) available here `https://folk.ntnu.no/fuglstad/Lund2016/Session6/spde-tutorial.pdf`.

- The true latent field $Z(\mathbf{s})$ is sampled on an extended grid ranging from -2 to 18 in both directions. The resolution of $Z(\mathbf{s})$ is 320×320 pixels. This totals 102 400 pixels, each of which represents an area of $(1/16^2)$. Due to the structure of the observation domains, Ω, the pixels lie either completely inside or completely outside all versions of Ω.

The integral over the true latent field is computed *exactly* by summing the contributions of all pixels within Ω, like a Riemann sum:

$$\text{int}_{\text{true}} = \int_{\Omega} \exp(Z(\mathbf{s}))\, \mathrm{d}\mathbf{s} = \frac{1}{16^2} \sum_{i:\text{pix } i \subset \Omega} \exp(Z_{\text{pix } i}), \tag{12}$$

where $Z_{\text{pix } i}$ denotes the i-th pixel value.

[**FIGURE** 7 about here.]

[**FIGURE** 8 about here.]

The methods we include in the present simulation study are listed in Table A1 along with any specifics and the names we shall use to refer to them.

[**TABLE** 1 about here.]

To simplify the performance comparison for different observation domains, we shall scale/normalize the integrals by the area of Ω, denoted $|\Omega|$. Denoting by $\widehat{\text{int}}_{\text{M}}$ the approximated integral with method M, we shall compare the performance of $\widehat{\text{int}}^*_{\text{M}} = \widehat{\text{int}}_{\text{M}}/|\Omega|$ to that of $\text{int}^*_{\text{true}} = \text{int}_{\text{true}}/|\Omega|$. We will use a few different measures to study the performance differences, all of which relate to the mean squared error (MSE):

$$\text{MSE}(\text{method M}) = \frac{1}{\text{K}} \sum_{\text{k}=1}^{\text{K}} \left(\text{int}^*_{\text{true}}(\text{sim k}) - \widehat{\text{int}}^*_{\text{M}}(\text{sim k})\right)^2, \tag{13}$$

where "(sim k)" denotes the integral values for simulation k, and $\text{K} = 10\,000$ is the total number of samples. It is well known that the MSE may be decomposed into a sum of the squared bias and variance, i.e. $\text{MSE}(\text{method M}) = \text{bias}^2(\text{method M}) + \text{Var}(\text{method M})$, where

$$\text{bias}^2(\text{method M}) = \left(\frac{1}{\text{K}} \sum_{\text{k}=1}^{\text{K}} \left(\text{int}^*_{\text{true}}(\text{sim k}) - \widehat{\text{int}}^*_{\text{M}}(\text{sim k})\right)\right)^2 \quad \text{and} \quad \text{Var}(\text{method M}) = \frac{1}{\text{K}} \sum_{\text{j}=1}^{\text{K}} \left(\left(\frac{1}{\text{K}} \sum_{\text{j}=1}^{\text{K}} \widehat{\text{int}}^*_{\text{M}}(\text{sim j})\right) - \widehat{\text{int}}^*_{\text{M}}(\text{sim k})\right)^2. \tag{14}$$

To better grasp the magnitudes of the different quantities, it is often convenient to bring the squared-scaled quantities above back to the scale of the quantities we measure (the normalized integrals). We shall therefore work with the rooted versions of the quantities in (13) and (14), i.e. the root mean squared error, $\text{RMSE}(\text{method M}) = \sqrt{\text{MSE}(\text{method M})}$, the (absolute) bias and the standard deviation (sd).

In addition to the above quantities, we shall rely on the so-called skill score (Gneiting & Raftery 2007) associated with the RMSE:

$$\text{skill}(\text{RMSE, method M}) = \frac{\text{RMSE}(\text{method M}) - \text{RMSE}(\text{reference})}{\text{RMSE}(\text{optimal}) - \text{RMSE}(\text{reference})} = 1 - \frac{\text{RMSE}(\text{method M})}{\text{RMSE}(\text{reference})}, \tag{15}$$

where RMSE(optimal) is the RMSE of an optimal method, being equal to zero. The skill score shows how well the methods perform relative to a reference method. Here, we will use the dual mesh method as a reference, since this was the method originally proposed by Simpson et al. (2016). The skill score is standardized such that the reference method (dual mesh) takes the value 0. A positive score means the method is better than the dual mesh approach, while a negative score means it is worse. The magnitudes are relative to RMSE(dual mesh), with the optimal value taking the value 1. All these scores will be computed separately for all the integration methods listed in Table A1, and every combination of the specifications for the mesh, Ω, the range r, and σ^2 described above. This gives a total of 960 different combinations for each of the integration methods, each with $\text{K} = 10\,000$ different samples. Note that to reduce the sensitivity to the true field $Z(\mathbf{s})$ actually being sampled when comparing the methods, all the 960 different combinations are performed with the same seeds when drawing the $\text{K} = 10\,000$ latent field samples.

Remark 1. It turns out that

$$\begin{aligned}\forall \mu_0 \in \mathbb{R}: \quad & \text{skill}_{\mu=\mu_0}(\text{RMSE, method M}) = \text{skill}_{\mu=0}(\text{RMSE, method M}), \quad \text{and} \quad \text{RMSE}_{\mu=\mu_0}(\text{method M}) = \exp(\mu_0)\text{RMSE}_{\mu=0}(\text{method M}), \\ & \text{and} \quad \text{bias}_{\mu=\mu_0}(\text{method M}) = \exp(\mu_0)\text{bias}_{\mu=0}(\text{method M}), \quad \text{and} \quad \text{sd}_{\mu=\mu_0}(\text{method M}) = \exp(\mu_0)\text{sd}_{\mu=0}(\text{method M})\end{aligned}$$

where the subscript denotes "simulations" executed with different values μ_0 for the mean of the latent Gaussian field μ. That is, the RMSE, bias and sd are multiplicative in μ_0, while the skill score is independent of μ_0. This is a result of two different factors: (a) $Z^*(\mathbf{s})$ is the minimizer of (11), i.e. it is the least squares solution, such that μ simply shifts the approximation $Z^*(\mathbf{s})$ by $\exp(\mu)$, and (b) all approximation methods for $\int \exp(Z(\mathbf{s}))\, \mathrm{d}\mathbf{s}$ take forms as fixed linear combinations of $\exp(Z^*(\mathbf{s}))$ which are evaluated at a set of fixed locations (the mesh nodes and extra points). Consequently,

$$\text{int}_{\text{true},\mu=\mu_0} = \int \exp(Z_{\mu=\mu_0}(\mathbf{s}))\, \mathrm{d}\mathbf{s} = \int \exp(\mu_0 + Z_{\mu=0}(\mathbf{s}))\, \mathrm{d}\mathbf{s} = \exp(\mu_0) \int \exp(Z_{\mu=0}(\mathbf{s}))\, \mathrm{d}\mathbf{s} = \exp(\mu_0)\text{int}_{\text{true},\mu=0}, \tag{16}$$

with identical expressions for all approximation methods $\widehat{\text{int}}_{\text{M}}$. Inserting this result into respectively (13) and (15), and gathering the common factor $\exp(\mu_0)$, gives the desired result. These results are confirmed by repeated simulations using the same seed for different values of $\mu = \mu_0$, and allows us to display simulation results solely for $\mu = 0$, without loss of generality.

4.3 | Simulation results

The full result tables with all the 5 aforementioned performance measures, the 8 methods described in Table A1 and all 960 simulation combinations ($5 \times 8 \times 960 = 38\,400$ scores) are available in a user-friendly online table through the GitHub repository (github.com/martinju/LGCP-normConst-simulations). The source code for the implementation of the different integration methods and the simulation scripts used to produce the results are also available in that repository. Due to the many different combinations and scores, the present paper will only include plots of a subset of these scores, that are indicating the main conclusions and messages.

We start out by looking at how the MeshExact method performs, both in itself and compared to the deterministic integration methods working directly on the mesh. Since the deterministic integration methods are all very similar compared to MeshExact, we will here just investigate and compare MeshExact with AverageDetMesh, the average of the DualMesh, DualMeshExtraMeshMapped, BarycentricPointSpread, BarycentricPointSpreadManyPoints, and Voronoi methods, see Table A1. Figure A9 shows the RMSE, bias and standard deviation in the specific situation where #sub domains = 1 and $\sigma^2 = 0.1$ for varying range and number of mesh nodes. For this illustration, we used a finer grid for the range to present a smoother plot. The results are very similar for other σ and #sub domains.

[**FIGURE** 9 about here.]

For MeshExact the RMSE plot clearly shows that the longer the range and the more mesh nodes that are used, the smaller is the error. As seen from the bias plot in the bottom left panel, this clear structure mainly stems from differences in the bias, which always is negative for MeshExact, indicating underestimating of the true integral.

The deterministic integration methods (as seen trough AverageDetMesh) have a more complex performance pattern. For the smallest and largest ranges, the RMSE behaves similarly to that of MeshExact (smaller errors for increased range and number of mesh nodes). For medium-sized ranges, however, it is sometimes the case that more mesh nodes or a longer range gives an increased error. This behavior is also a consequence of the bias, being negative for small ranges, and positive for large ranges, but crossing and being closest to zero at different ranges for different number of mesh nodes.

The variability (standard deviation in the bottom right panel) does not vary too much based on the number of mesh nodes, but has an approximately linear increase in the range parameter, for both types of methods. While it is hard to spot from the plot, the standard deviation, as expected, increases slightly when increasing the number of mesh nodes.

Comparing the two types of approaches for a fixed range and mesh, MeshExact is best for the longest ranges, while the reverse is true for small ranges. The location of the flipping point between the two scenarios depends on the number of mesh nodes (the more mesh nodes, the smaller need the range be for MeshExact to be the best).

Since MeshExact performs uniformly better when increasing the number of mesh nodes, while the deterministic integration methods do not, it is relevant to investigate how the latter performs compared to MeshExact when the latter uses the largest number of mesh nodes. Figure A10 illustrates this in terms of the differences in the RMSE between the MeshExact method with the largest number of mesh nodes, and the AverageDetMesh with different number of mesh nodes.

[**FIGURE** 10 about here.]

For medium to large ranges, the MeshExact approach has the smallest error. For very short ranges it is preferable to use a deterministic integration method with many mesh nodes instead. For short (but not very short) ranges, it is better to reduce the number of mesh nodes slightly.

Let us now look at the performance differences for the deterministic integration methods with the mesh nodes as the only integration points[3]. To avoid information overload, a selection of the parameter combinations are included in the plots below. Figure A12 contains the skill score of the RMSE with DualMesh as reference method as a function of the range when the variance is fixed at $\sigma^2 = 0.5$, while the different panels show plots for different combinations of the four Ω-specifications, and the six different mesh specifications. Figure A11 contains the same type of plots as Figure A12, but with the densest Ω (# sub domains = 1), variances at $\sigma^2 = 0.1, 0.2$ and 1 ($\sigma^2 = 0.5$ is already shown in Figure A12), combined with all the six different mesh specifications.

[**FIGURE** 11 about here.]

[**FIGURE** 12 about here.]

Before we take a closer look at the specific results, it is worth mentioning that while many of the methods appear to be performing very similarly, permutation tests of differences in their (R)MSE actually show that more than 93% of the differences between the methods with all parameter

[3] For a fair comparison, the DualMeshExtra approach (but not DualMeshExtraMeshMapped) is excluded from this comparison, as that method uses more integration points than the other methods and therefore has a clear advantage.

combinations presented in Figure A11 and A12 are significantly different (p-value < 0.01). A full table of p-values for the (R)MSE difference permutation tests for all combinations of methods and parameter combinations in the simulation study are available in a user-friendly table through the aforementioned GitHub repository. Even though the methods perform significantly different for most parameter combinations, which method that performs the best is strongly dependent on the specific parameter specification. This makes it a bit difficult to fully generalize on specialties for different methods.

As seen by comparing the columns in Figure A11, increasing the variance reduces the performance difference between the methods in terms of the skill score. This is a result of all methods getting reduced RMSE, leading to more similar skill scores. Similarly, increasing the number of mesh nodes typically has a similar effect for the two dual mesh approaches and the two Barycentric point spreading methods, at least for smaller number of subdomains (the sparsity of Ω). When the range is small, all methods (possibly except for Voronoi) perform quite similar. In most situations increasing the range gives larger performance differences between the methods. As expected, the performances are most stable when # subdomains = 1. However, it is hard to generalize further on the behavior of the sparsity of Ω based on these results.

Overall, Voronoi stands out negatively in small segments for the range parameter with every parameter combination. On the other hand, the Voronoi method performs well for medium to large ranges for the two densest meshes. The BarycentricPointSpread method is the method performing best for most parameter combinations, and in some cases, it clearly stands out as much better than the others. However, in a few cases (e.g. for the largest number of mesh nodes and subdomains with $\sigma^2 = 0.5$, see Figure A12, it has poor performance. Behavior like this makes the method a bit unstable, just like the Voronoi method. BarycentricPointSpreadManyPoints is a refined version of BarycentricPointSpread, that in theory ought to be a more precise approximation to DualMeshExtraMeshMapped – a property that is also verified in our simulation. BarycentricPointSpreadManyPoints is seldom the best method, but it does not show instability problems like BarycentricPointSpread and Voronoi, it is often better than DualMeshExtraMeshMapped, and almost always better than DualMesh. In other words, BarycentricPointSpreadManyPoints is a stable and safe choice that always performs decently. We, therefore, recommend this method among the deterministic integration methods working directly on the mesh nodes.

When comparing approximation methods to be used in computationally expensive algorithms, not only the accuracy of the methods matter but also the computation time. First, comparing the expressions for the deterministic integration method in (6), with the formula for the mesh-exact integration method in (9) and (10), we see that the former has a much simpler form. The former is generally much faster to compute both since the summand is more complex and the size of the sum is larger. The increased complexity is natural as (6) is an approximation to the mesh-exact integration method. Furthermore, the variants of the deterministic integration method using solely the mesh nodes differ only in how they assign the weights. These weights can be pre-computed for a given mesh. Computing the weights is therefore just a fixed one-time cost from a full inference perspective. Table A2 in Appendix A shows the computation time for these integration weights. While our recommendation BarycentricPointSpreadManyPoints often is the slowest, we deem all the one-time pre-computation costs small enough not to cause practical problems (a few seconds up to a few minutes in the most extreme case). For completeness, the table also includes the corresponding pre-computation time for the mesh-exact integration method (i.e. the components involved in the α^*, β^*, and γ^* quantities), which also has comparable speed. Note, however, that none of the implementations are optimized for speed, so the numbers should merely be seen as rough indicators.

5 | SUMMARY AND CONCLUDING REMARKS

We have investigated different methods for approximating the normalization constant in the Cox process likelihood consisting of the integral $\int_\Omega \exp(Z(\mathbf{s}))\,\mathrm{d}\mathbf{s}$. According to Figures A9 and A10, our main finding is that the mesh-exact integration method gives more precise approximations to the normalization constant when the mesh is dense and/or the range of the latent field is long. On the other hand, when the mesh is sparse and the range is small, it is often better to use a deterministic integration method.

As a deterministic integration method is preferable in some situations, we also compared the different variants of that approach against each other. According to Figures A11 and A12, for simple observation domains, small ranges and large variances of the latent field, the performance differences are small such that it is not crucial which of the variants are being used. Nevertheless, our general recommendation among the deterministic integration methods is the BarycentricPointSpreadManyPoints method. This is a simple extension to the BarycentricPointSpread method implemented in `inlabru` (v2.1.12). The extension is, as of this writing (Jan 2020), available through a fork of the `inlabru` `R`-package repository at github.com/martinju/inlabru, and will hopefully make it into the official version of the package in due time.

While the performed simulation experiments are directly useful for LGCP models, the approaches outlined in the present paper are also relevant for other models where an integral over a spatial field needs to be computed, and a mesh-based approximation is sensible. This includes a series of other Cox processes or Markov point processes, see e.g. Møller and Waagepetersen (2003, Ch. 5-6).

In the present paper, we have concentrated on the pure Cox process where no other information than an observed point pattern, is available. In many practical situations, additional covariate information is available, that one may wish to include in the inference. Typically, such covariate effects

are included in the intensity function $\lambda(\mathbf{s})$ on a log-linear form, i.e. $\log(\lambda(\mathbf{s})) = \mathsf{Z}_{\text{cov}}(\mathbf{s}) = \mathsf{Z}(\mathbf{s}) + \mathbf{x}(\mathbf{s})^\top \boldsymbol{\beta}$, for $\mathbf{x}(\mathbf{s})$ a vector of covariates observed at location $\mathbf{s}$, and $\boldsymbol{\beta}$ a coefficient vector to be estimated. In this case, the integral to approximate is $\int_\Omega \lambda(\mathbf{s})\,\mathrm{d}\mathbf{s} = \int_\Omega \exp(\mathsf{Z}(\mathbf{s}) + \mathbf{x}(\mathbf{s})^\top \boldsymbol{\beta})\,\mathrm{d}\mathbf{s} = \int_\Omega \exp(\mathsf{Z}(\mathbf{s}))\exp(\mathbf{x}(\mathbf{s})^\top \boldsymbol{\beta})\,\mathrm{d}\mathbf{s}$. Thus, characteristics of the covariate vector such as the observation resolution and variability within the observation domain, may affect the performance of the different integration methods. Since the random field is assumed to live on the mesh, it is natural and simplifying to restrict $\mathbf{x}(\mathbf{s})$ to live on the same mesh when performing the integration. In this case, the covariate approximation may be carried out similarly to the latent field approximation in Section 4.1. One may then proceed just like before, and choose integration method based on the assumed/estimated characteristics of $\mathsf{Z}_{\text{cov}}(\mathbf{s})$, rather than $\mathsf{Z}(\mathbf{s})$. If the resolution of the covariates is much finer than the mesh and this fine scale is deemed important to reach precise approximations to the integral, there are at least two alternative approaches. One alternative is to simply extend the mesh with additional mesh nodes, approximate $\mathbf{x}(\mathbf{s})$ to live on the extended mesh, and then perform integration as above using the extended mesh. Another alternative is to use approximate Riemann integration by evaluating the integrand on a high resolution grid of the observation domain Ω, similar to (12). Further analysis and work are required to conclude on the performance of such an approach. However, approximate Riemann integration can approximate the mesh-exact method arbitrarily well (by increasing the resolution of the grid) when $\mathbf{x}(\mathbf{s})$ lives on the mesh, we can expect the performance to be similar to the mesh-exact approach, unless the fine scale behavior of $\mathbf{x}(\mathbf{s})$ dominates $\mathsf{Z}_{\text{cov}}(\mathbf{s})$ completely.

A practical limitation of the INLA software implementation is that it cannot handle likelihoods consisting of multiple linear combinations of the mesh nodes (Yuan et al. 2017, Remark Sec. 3.2). This is unproblematic for the deterministic integration methods, whose integrand approximation depends on a single linear combination. The formula for the mesh-exact integration method, on the other hand, depends on three linear combinations. Thus, it is currently not possible to fit a model with the mesh-exact integration method using the INLA software. Luckily, there exists a computationally efficient alternative to INLA which does not have such a restriction: Template Model Builder (TMB) (Kristensen, Nielsen, Berg, Skaug, & Bell 2016) is an inference method and R-package that performs high dimensional inference by combining Laplace approximations, automatic differentiation, and non-linear optimization. Implementing the full LGCP likelihood in TMB with the mesh-exact approximation for the normalization constant has a high potential for providing more accurate inference for LGCP models. While being out of scope for the present paper, this task is on the top of the list of further work.

ACKNOWLEDGMENTS

The author thanks Thordis Thorarinsdottir for several helpful discussions and comments that greatly improved the manuscript, and Finn Lindgren and Haakon Bakka for input on the mesh approximation method.

APPENDIX

A TABLE WITH PRE-COMPUTATION TIMINGS OF THE INTEGRATION METHODS

[**TABLE** 2 about here.]

SUPPORTING INFORMATION

Additional information for this article is available. This contains the mathematical derivation of the analytical formula for integral of the over the exponential latent field, being exact under the triangular mesh assumption.

DATA AVAILABILITY STATEMENT

All source code and simulation scripts in R, in addition to user-friendly online tables with the results for the full simulation study in Section 4 and associated permutation tests are available through the GitHub repository github.com/martinju/LGCP-normConst-simulations.

References

Babu, G. J., & Feigelson, E. D. (1996). Spatial point processes in astronomy. *Journal of statistical planning and inference*, *50*(3), 311–326.

Bachl, F. E., Lindgren, F., Borchers, D. L., & Illian, J. B. (2019). inlabru: an r package for bayesian spatial modelling from ecological survey data. *Methods in Ecology and Evolution*, *10*(6), 760–766.

Blangiardo, M., & Cameletti, M. (2015). *Spatial and spatio-temporal bayesian models with r-inla*. John Wiley & Sons.

Cameletti, M., Lindgren, F., Simpson, D., & Rue, H. (2013). Spatio-temporal modeling of particulate matter concentration through the spde approach. *AStA Advances in Statistical Analysis*, *97*(2), 109–131.

Cox, D. R. (1955). Some statistical methods connected with series of events. *Journal of the Royal Statistical Society. Series B (Methodological)*, 129–164.

Eberhard, D. A., Zechar, J. D., & Wiemer, S. (2012). A prospective earthquake forecast experiment in the western pacific. *Geophysical Journal International*, *190*(3), 1579–1592.

Gneiting, T., & Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. *Journal of the American Statistical Association*, *102*(477), 359–378.

Guttorp, P., & Thorarinsdottir, T. L. (2012). Bayesian inference for non-markovian point processes. In *Advances and challenges in space-time modelling of natural events* (pp. 79–102). Springer.

Jullum, M., Thorarinsdottir, T., & Bachl, F. E. (2020). Estimating seal pup production in the greenland sea by using bayesian hierarchical modelling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *69*(2), 327-352. Retrieved from `https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/rssc.12397` doi: 10.1111/rssc.12397

Krainski, E. T., Gómez-Rubio, V., Bakka, H., Lenzi, A., Castro-Camilo, D., Simpson, D., … Rue, H. (2018). *Advanced spatial modeling with stochastic partial differential equations using R and INLA*. Chapman and Hall/CRC.

Kristensen, K., Nielsen, A., Berg, C. W., Skaug, H., & Bell, B. M. (2016). TMB: Automatic differentiation and Laplace approximation. *Journal of Statistical Software*, *70*(5), 1–21. doi: 10.18637/jss.v070.i05

Langtangen, H. P., & Mardal, K.-A. (2016). Introduction to numerical methods for variational problems. `https://folk.uio.no/kent-and/hpl-fem-book/doc/pub/book/pdf/fem-book-4print.pdf`.

Lindgren, F., Rue, H., & Lindström, J. (2011). 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.

Mohler, G. O., Short, M. B., Brantingham, P. J., Schoenberg, F. P., & Tita, G. E. (2011). Self-exciting point process modeling of crime. *Journal of the American Statistical Association*, *106*(493), 100–108.

Møller, J., & Waagepetersen, R. P. (2003). *Statistical inference and simulation for spatial point processes*. CRC Press.

Rue, H., Martino, S., & Chopin, N. (2009). Approximate Bayesian inference for latent Gaussian models using integrated nested Laplace approximations (with discussion). *Journal of the Royal Statistical Society, Series B*, *71(2)*, 319-392.

Simpson, D., Illian, J. B., Lindgren, F., Sørbye, S. H., & Rue, H. (2016). Going off grid: Computationally efficient inference for log-gaussian cox processes. *Biometrika*, *103*(1), 49–70.

Stoica, R., Tempel, E., Liivamägi, L., Castellan, G., & Saar, E. (2014). Spatial patterns analysis in cosmology based on marked point processes. *European Astronomical Society Publications Series*, *66*, 197–226.

Waller, L. A., Särkkä, A., Olsbo, V., Myllymäki, M., Panoutsopoulou, I. G., Kennedy, W. R., & Wendelschafer-Crabb, G. (2011). Second-order spatial analysis of epidermal nerve fibers. *Statistics in Medicine*, *30*(23), 2827–2841.

Yuan, Y., Bachl, F. E., Lindgren, F., Borchers, D. L., Illian, J. B., Buckland, S. T., … others (2017). Point process models for spatio-temporal distance sampling data from a large-scale survey of blue whales. *The Annals of Applied Statistics*, *11*(4), 2270–2297.

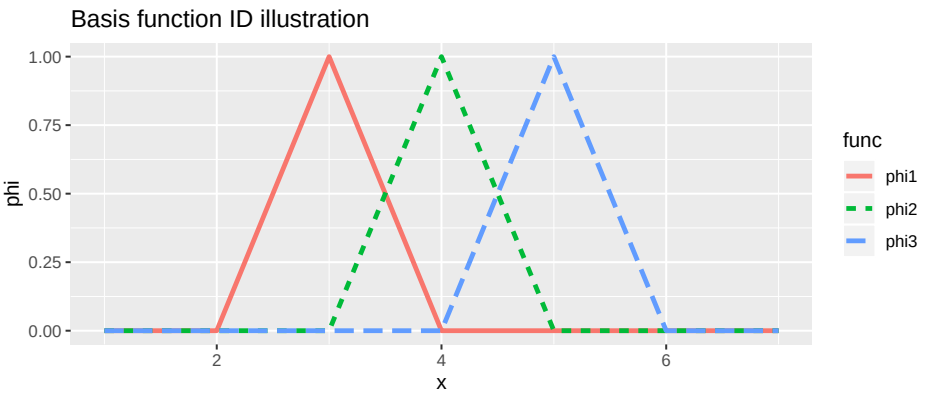

FIGURE A1 Illustration of three 1-dimensional basis functions with nodes at every integer value.

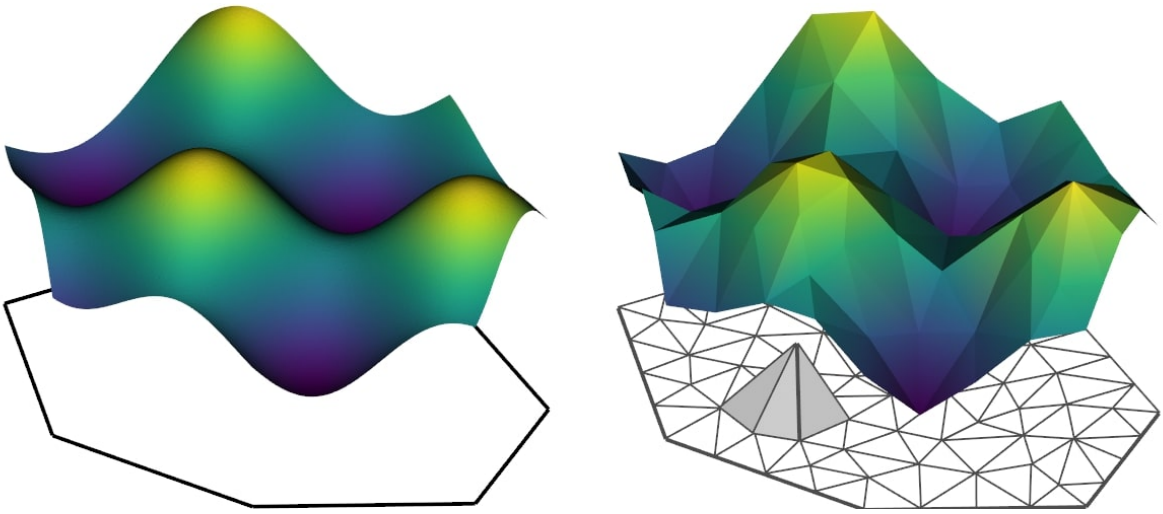

FIGURE A2 Illustration of how a FEM based triangular mesh (to the right) approximates an original smooth field (to the left).

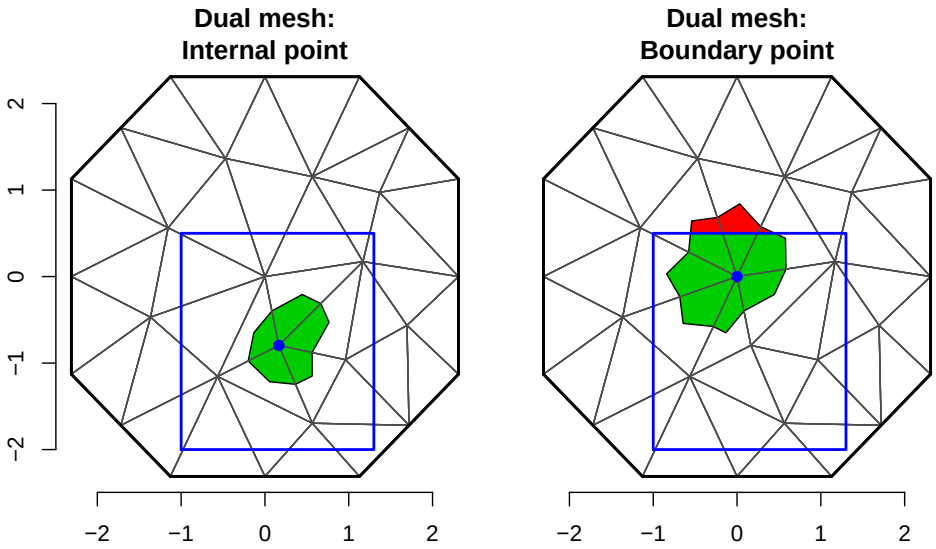

FIGURE A3 Illustration of how the dual mesh approach assigns weights to internal and boundary integration points (shown by blue dots). The black triangles show the mesh and the blue squares show the extent of the observation domain Ω. The area of the green part of the polygon is the weight assigned to the corresponding integration point. The red part of the polygon is part of the dual mesh, but not included in the weight computation, as it is outside Ω.

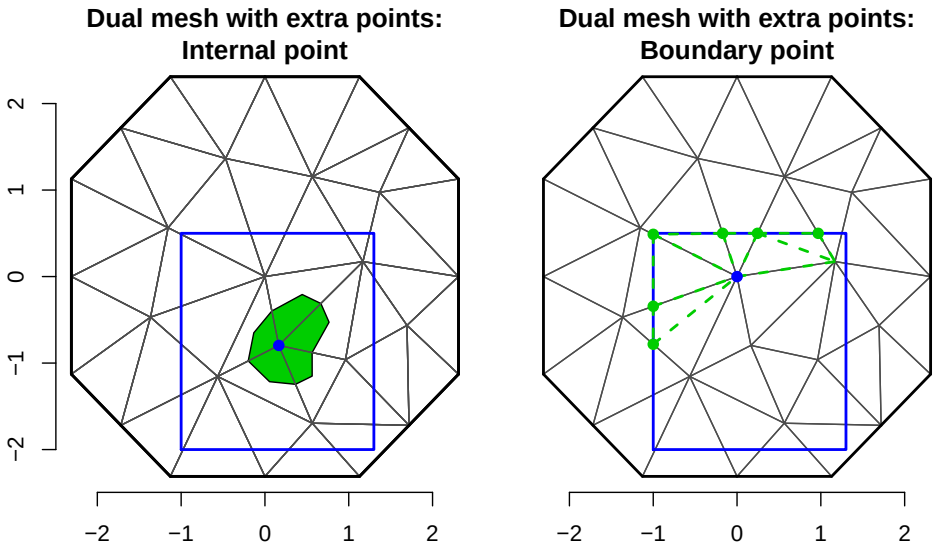

FIGURE A4 Illustration of how the dual mesh approach with extra integration points assigns weights to internal and boundary integration points (shown by blue dots). The black triangles show the mesh and the blue squares show the extent of the observation domain Ω. For internal integration points, the area of the green part of the polygon is the weight assigned to the corresponding integration point. For boundary integration points, green points show additional integration points and green dashed lines indicate new sub-triangles for which 1/3 of the area is assigned to the integration point.

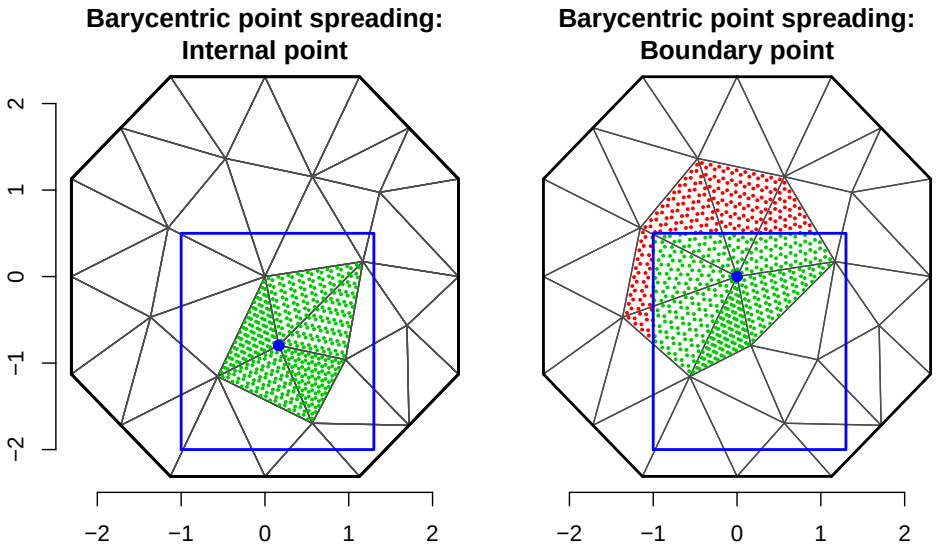

FIGURE A5 Illustration of how the Barycentric point spreading approach assigns weights to internal and boundary integration points (shown by blue dots). The black triangles show the mesh and the blue squares show the extent of the observation domain Ω. Green points are mapped to the integration points, while red points, being outside Ω, are not.

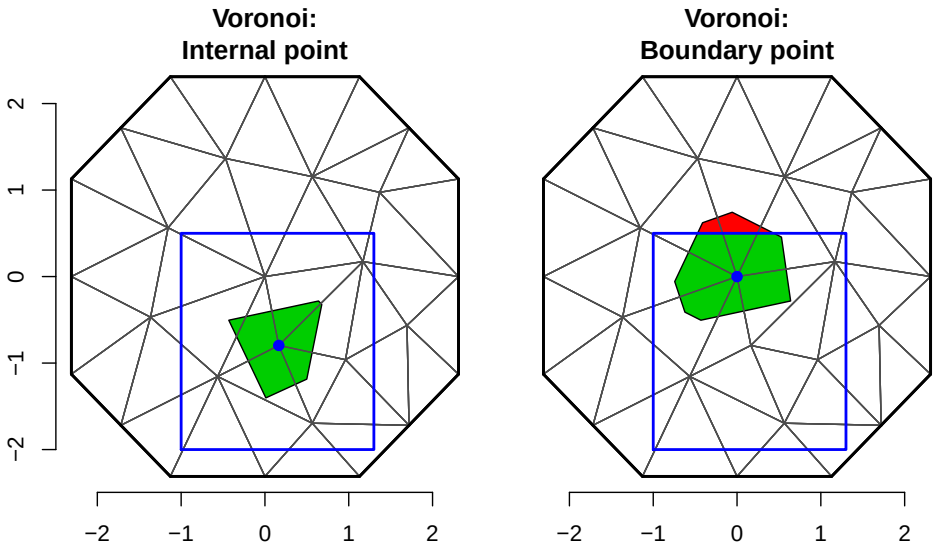

FIGURE A6 Illustration of how the Voronoi tessellation approach assigns weights to internal and boundary integration points (shown by blue dots). The black triangles show the mesh and the blue squares show the extent of the observation domain Ω. The area of the green part of the polygon is the weight assigned to the corresponding integration point. The red part of the polygon is part of the Voronoi tessellation, but not included in the weight computation, as it is outside Ω.

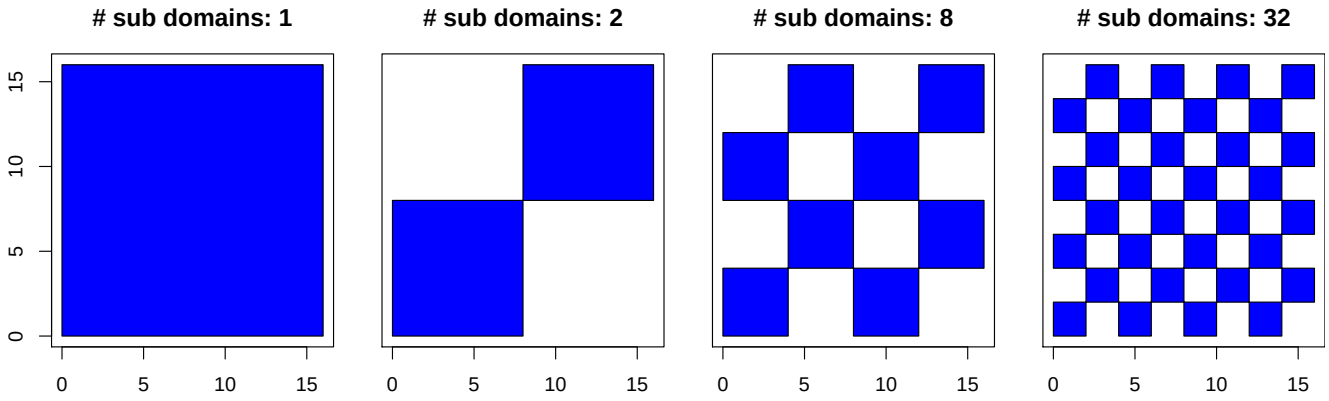

FIGURE A7 Illustration of the different observation domains Ω (union of blue polygons) used in the simulation study.

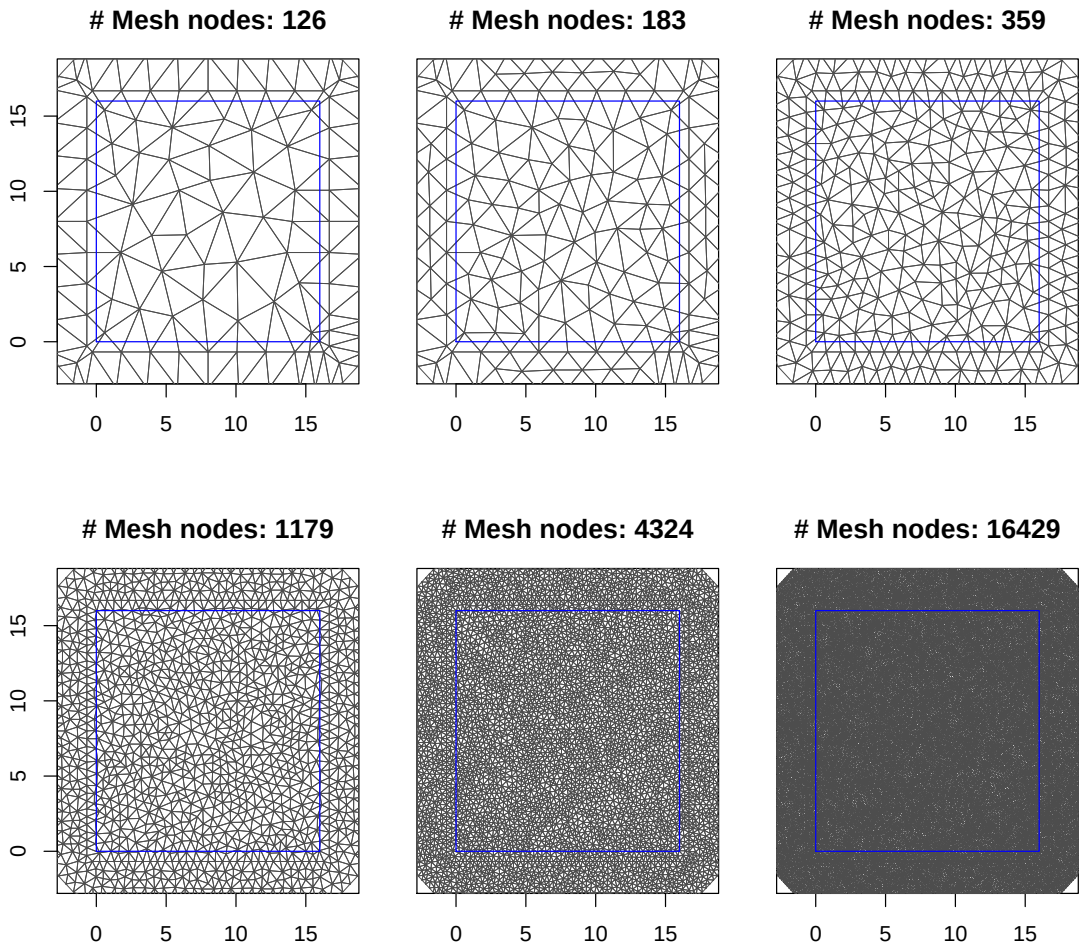

FIGURE A8 Illustration of the different meshes used in the simulation study. The number of mesh nodes above the plots refers to the number of mesh nodes which affects approximations in the largest observation domain (shown by the blue square).

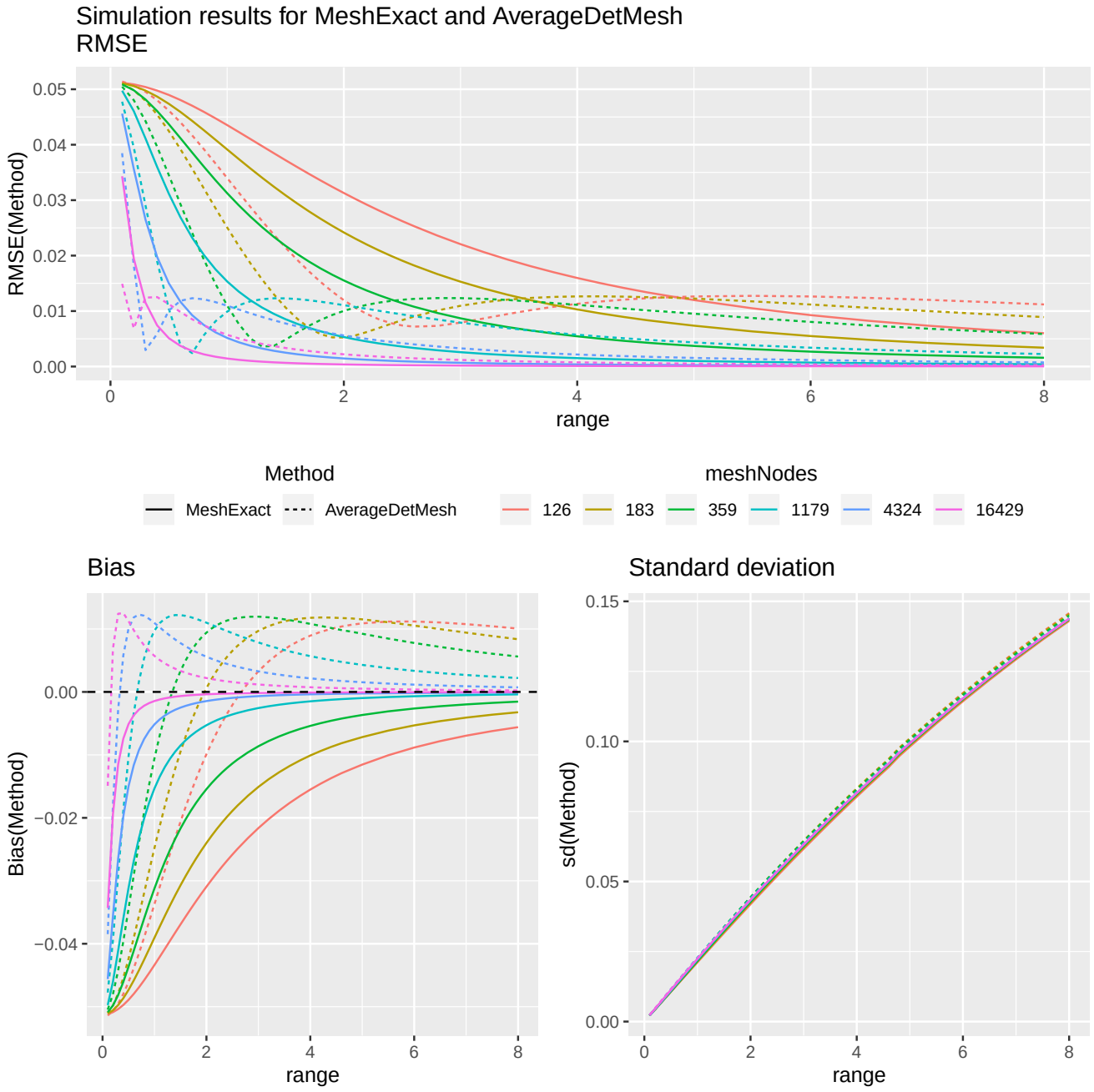

FIGURE A9 Simulation results for MeshExact (solid lines) and AverageDetMesh (dashed lines) for the densest Ω (# sub domains = 1), $\sigma^2 = 0.1$, the six different meshes (shown in different colors), and a finer grid of ranges $(0.1, 0.2, \ldots, 7.9, 8.0)$. The top panel shows the RMSE, the bottom left panel shows the bias and the bottom right panel shows the standard deviation. Lower RMSE and sd, as well as bias closer to zero, indicate better performance.

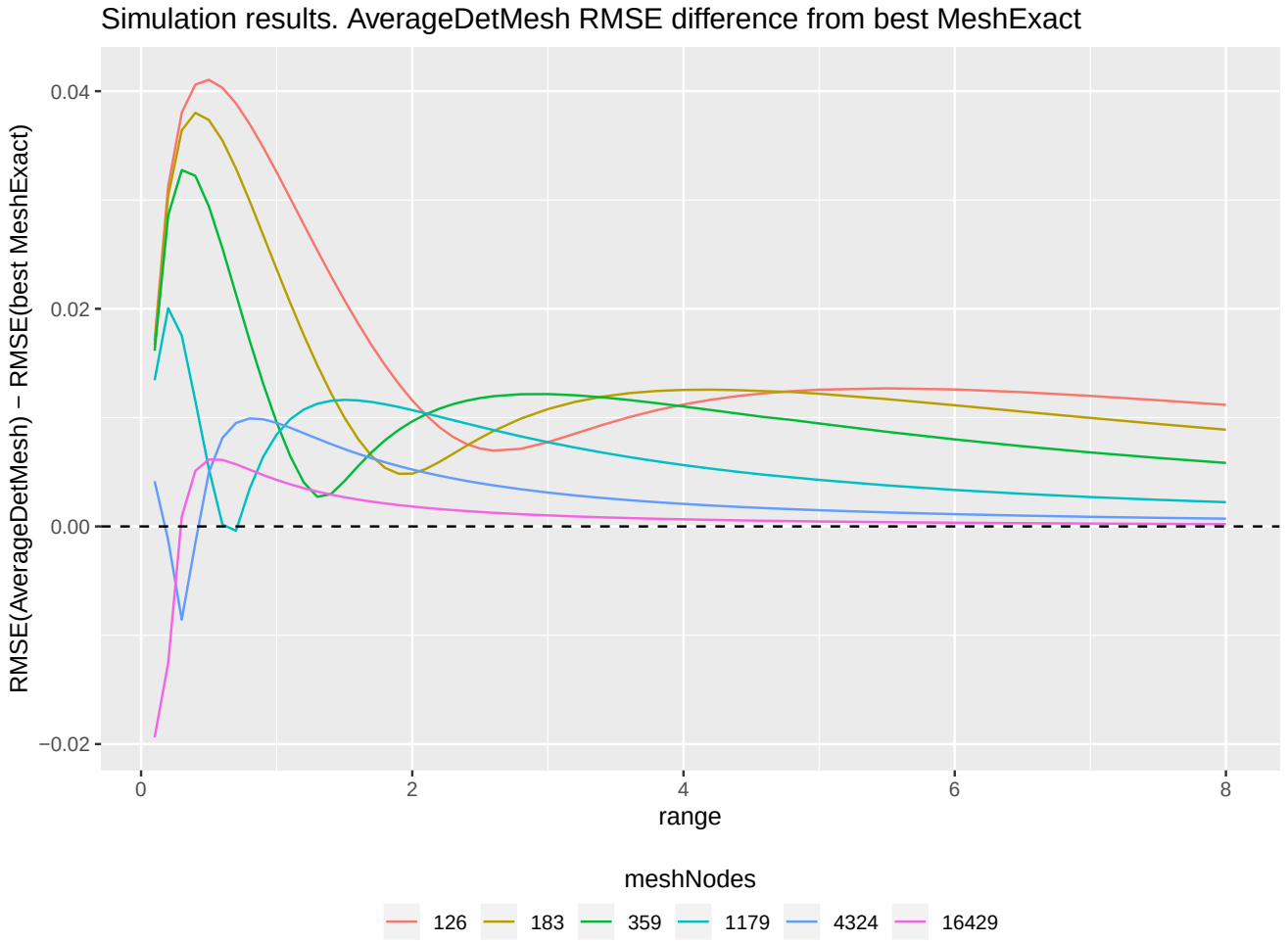

FIGURE A10 The difference in RMSE for AverageDetMesh with different number of mesh nodes compared to the (best performing) MeshExact approach with the largest number of mesh nodes (16 429) for the setup shown in Figure A9. When the lines are above the black dashed horizontal line at 0, MeshExact performs best, and vice versa below the 0-line.

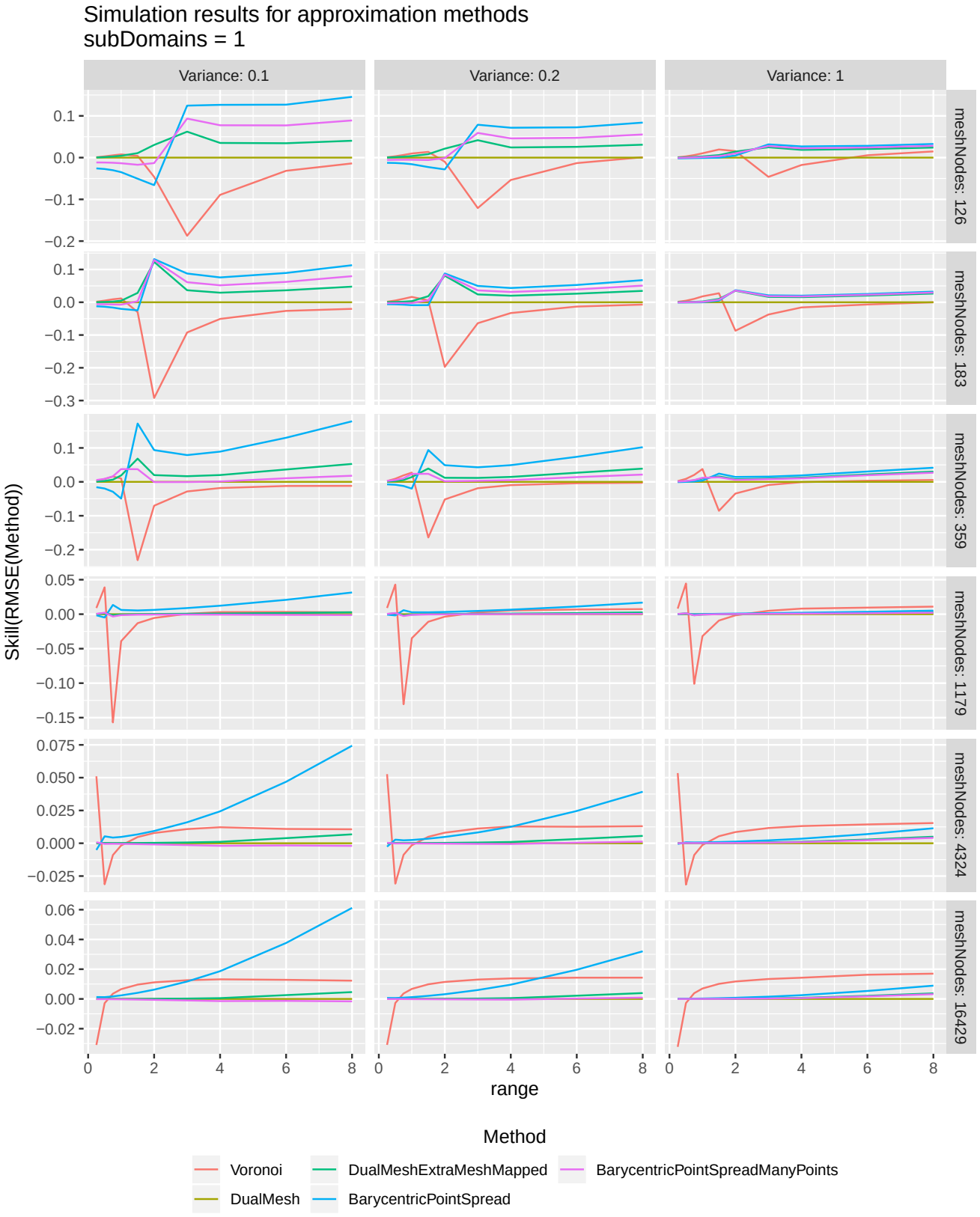

FIGURE A11 Simulation results for the deterministic integration methods which uses the mesh nodes as integration points. The plots show the skill score of the RMSE with DualMesh as reference method for the five approaches (in different colors), as a function of the range. The number of Ω sub domains is fixed at 1, while plots are shown for variance σ^2 = 0.1, 0.2, and 1, in combination with the six different mesh specifications. Higher skill score indicates better performance compared to DualMesh.

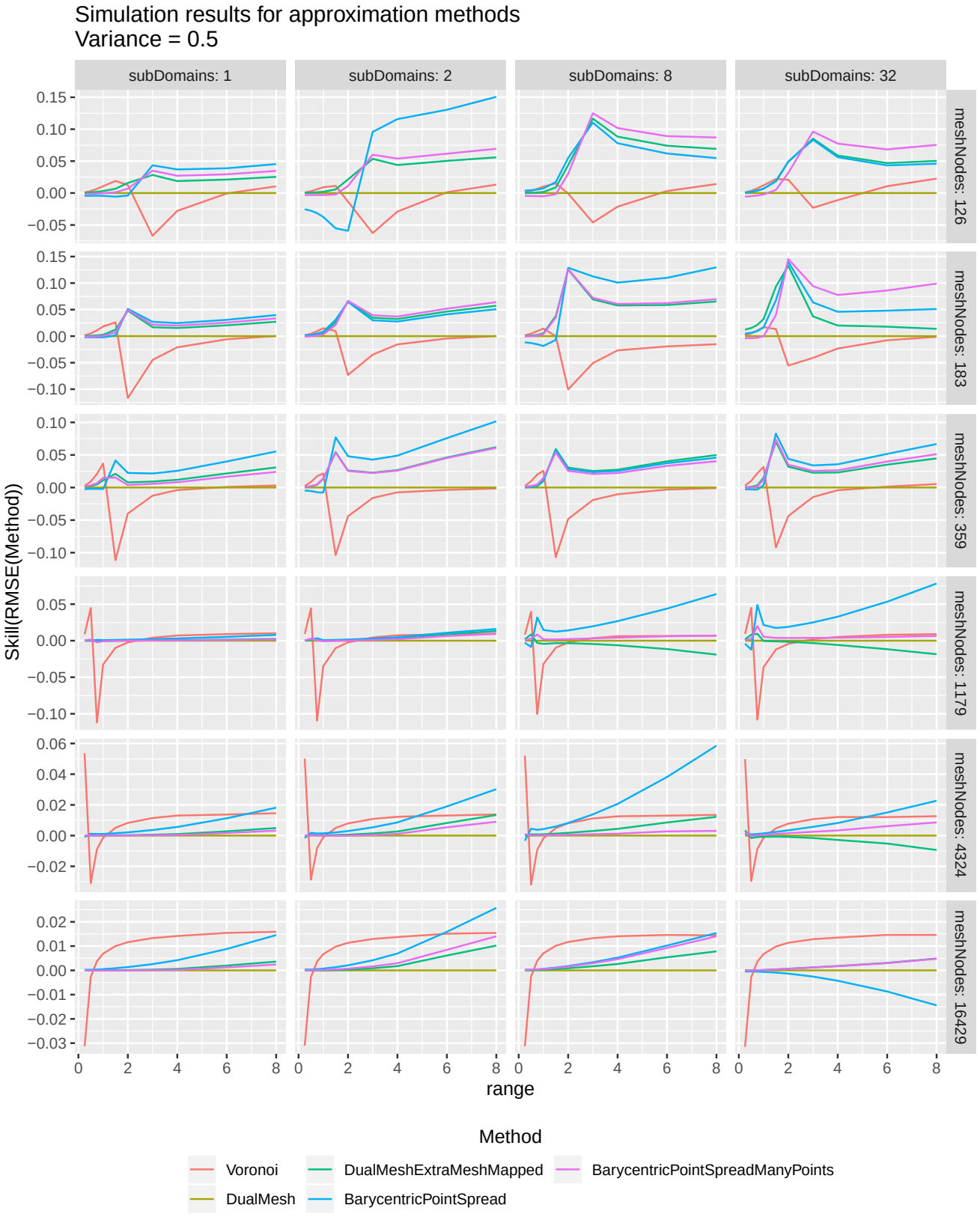

FIGURE A12 Simulation results for the deterministic integration methods which use the mesh nodes as integration points. The plots show the skill score of the RMSE with DualMesh as reference method for the five approaches (in different colors), as a function of the range. The variance is kept fixed at $\sigma^2 = 0.5$ while the different panels show plots for different combinations of all four Ωs (# sub domains = 1, 2, 8 and 32), and the six different mesh specifications. Higher skill score indicates better performance compared to DualMesh.

TABLE A1 List of methods included in the simulation study

- **MeshExact**: The mesh-exact integration method (Section 2.2).
- **DualMesh**: The dual mesh approach (Section 3.1).
- **DualMeshExtra**: The dual mesh approach with extra integration points using the extra integration points directly (Section 3.2).
- **DualMeshExtraMeshMapped**: The dual mesh approach with extra integration points, where the points are mapped back to the mesh nodes (Section 3.2).
- **BarycentricPointSpread**: The Barycentric point spreading approach with 100 points per triangle as hard-coded in `inlabru` (v 2.1.12) (Section 3.3).
- **BarycentricPointSpreadManyPoints**: The Barycentric point spreading approach with 1000 points per triangle (Section 3.3).
- **Voronoi**: The Voronoi tessellation approach (Section 3.4).
- **AverageDetMesh**: The average integration value among all methods using the deterministic integration method based solely on the mesh nodes (DualMesh, DualMeshExtraMeshMapped, BarycentricPointSpread, BarycentricPointSpreadManyPoints and Voronoi).

meshNodes	subDomains	DualMesh	DualMesh-ExtraMesh-Mapped	Barycentric-PointSpread	Barycentric-PointSpread-ManyPoints	Voronoi	MeshExact
126	1	0.36	1.12	0.72	0.85	0.90	0.77
183	1	0.40	1.33	0.85	2.05	0.25	0.75
359	1	0.67	2.31	0.90	1.43	0.45	1.54
1179	1	2.11	5.12	1.24	3.55	1.54	3.68
4324	1	7.23	17.23	2.54	12.63	7.84	11.61
16429	1	31.69	97.04	9.59	58.02	34.22	40.39
126	2	0.28	0.96	1.43	1.64	0.17	0.54
183	2	0.34	1.15	1.50	1.81	0.23	0.85
359	2	0.58	1.88	1.60	2.22	0.43	1.23
1179	2	1.89	4.16	2.16	4.62	1.50	3.40
4324	2	6.88	13.97	4.41	16.28	6.64	10.62
16429	2	29.53	58.18	14.01	58.49	34.25	36.08
126	8	0.32	1.53	5.68	6.41	0.21	1.13
183	8	0.42	2.07	6.38	7.23	0.29	1.46
359	8	0.71	3.21	6.67	8.66	0.56	2.49
1179	8	2.22	6.94	8.23	16.95	1.83	6.00
4324	8	7.72	20.55	16.14	55.62	7.79	15.85
16429	8	32.24	79.51	53.10	192.71	36.72	48.46
126	32	0.61	6.44	21.57	24.17	0.53	5.97
183	32	0.88	8.64	24.51	28.08	0.77	8.04
359	32	1.55	13.80	24.82	32.19	1.44	12.36
1179	32	4.80	27.61	31.80	62.44	4.61	26.33
4324	32	16.55	63.67	63.74	201.65	16.16	57.89
16429	32	63.61	178.73	208.66	706.24	66.44	139.70

TABLE A2 The table shows the number of CPU seconds used to compute all mesh-related quantities (simply the integration weights for the deterministic integration methods) for different methods on a desktop computer with a AMD Ryzen Threadripper 1950X, 3.4 GHz CPU, on a Linux Ubuntu 18.04 system, running R version 3.6.1.