

TDK-dolgozat

Száraz Máté András



Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Efficient simulation of normal random fields on metric graphs

Máté András Száraz

Computer Science Engineering BSc (7th semester)

2024-11-13

Supervisors: Prof. Mihály Kovács (Professor), Mihály András Vághy (PhD student)

SUPPORTED BY THE EKÖP-24-1 UNIVERSITY RESEARCH SCHOLARSHIP PROGRAM OF THE
MINISTRY FOR CULTURE AND INNOVATION FROM THE SOURCE OF THE NATIONAL
RESEARCH, DEVELOPMENT AND INNOVATION FUND.

Contents

Introduction	3
Preliminaries	4
2.1 Fundamentals of Stochastic Calculus	5
2.2 Quantum Graphs	9
2.3 Iterative Linear Solvers	13
Finite Element Method for Quantum Graphs	16
3.1 Finite Element Method on One Edge	17
3.2 Extension for Quantum Graphs	21
3.3 Implementation	23
3.4 Results, Efficiency	24
3.5 Neumann - Neumann Domain Decomposition	25
Gaussian Random Fields on Domains	28
4.1 Theoretical Background	29
4.2 Implementation in Python	31
4.3 Examples	32
Fractional Stochastic Partial Differential Equations on Quantum Graphs	32
5.1 SPDE Approach of Gaussian Random Fields	36
5.1.1 Integral Quadrature	36
5.1.2 Implementation	38
Examples	39
Conclusions	41
Acknowledgements	42

Introduction

In recent decades, elliptic partial differential equations (PDEs) defined on networks, often called quantum graphs, were used in a wide range of applications involving quasi-one-dimensional phenomena, such as modeling chemical contaminations in river systems [1], biological neural networks [2], [3], structural analysis of elastic beam frames [4] and estimation and prediction of urban air pollution [5]. Given a graph, we define a suitable PDE on each edge and couple them with appropriate vertex conditions. In the simplest case we require that the solution is continuous at the vertices and it satisfies a Neumann-Kirchhoff condition similar to the well-known current law.

Gaussian random fields and, in particular, Matérn random fields play an important role in the modeling and statistical analysis of spatial data, such as geostatistics [6], engineering mechanics [7] and macroeconomics [8]. The classical approach of simulating an n -dimensional random field involves discretizing an appropriate expected value function and covariance function to obtain an expected value vector and covariance matrix. Then the required realization can be computed using a standard normal vector and the Cholesky decomposition (or the square-root) of the covariance matrix. While there are many suitable covariance kernel functions, arguably, the most important one is the Matérn kernel; in [9] the author concludes the first chapter with the simple practical suggestion: "Use the Matérn kernel." However, the computational complexity and memory requirement of this approach is quite extensive.

In [10] the authors show that random fields with a Matérn-type covariance satisfy a fractional order elliptic PDE with white noise forcing. Since the numerical solution of such a stochastic PDE (SPDE) usually requires less computational power, this approach found many practical applications in the past decade [11]. When generating random fields on more complicated spatial structures, such as manifolds (e.g. sphere) or networks, in principle, we could follow the Cholesky-based approach. However, the natural distance metrics in these cases do not produce positive definite covariance matrices, or only do so for very restrictive parameter configurations [12], [13]. Instead, we *define* Matérn fields

as the solutions of the appropriate fractional SPDE. The well-posedness and numerical approximation of these equations was investigated in [14], [15].

While the theory of spectral numerical methods can be extended to PDEs on networks, in general, the spectra of elliptic differential operators on networks is unknown. Thus, we usually have to resort to the generalization of the well-known finite element method (FEM), introduced in [16]. While the extension of FEM to networks is natural, the stiffness matrix loses its banded structure, and thus the solution of the underlying matrix equation is more computationally heavy. Moreover, the size of the stiffness matrix quickly blows up for large graphs, and thus even the storage of the stiffness matrix can be problematic. In [17] the authors design a Neumann-Neumann type domain decomposition method, show that it is equivalent to the iterative solution of the preconditioned Schur complement of the degrees of freedoms corresponding to the vertex nodes, and prove that the iteration converges to the finite element solution with a rate that is independent of the stepsize. The goal of this work is to generalize the Neumann-Neumann iteration to the case of fractional SPDEs in order to efficiently generate realizations of Matérn fields.

The structure of the paper is as follows. In Chapter 2 I briefly introduce the necessary background of stochastic calculus, quantum graphs and iterative linear solvers. Chapter 3 contains the variational formulation of the PDEs in question, their FEM-based numerical solution, the Neumann-Neumann method, along with some implementation details. In Chapter 4 I demonstrate the classical Cholesky decomposition-based approach of random fields on domains. Then in Chapter 5 I introduce a numerical method for solving the fractional SPDE. Finally, Chapter 6 contains some numerical examples.

Preliminaries

In this chapter, I briefly summarize the necessary background for simulating normal random fields on metric graphs.

2.1. Fundamentals of Stochastic Calculus

In this section, I will present and explain some key concepts and theorems from stochastic calculus that form the basis of this research project, based on [18], [19].

Stochastic calculus introduces numerous mathematical tools for examining different types of experiments and random trials. The **sample space**, denoted by Ω , consists of all possible outcomes of an experiment. An event is any subset of the sample space Ω . If the outcome of the experiment lies in this subset, we say that the event has occurred. To rigorously represent these subsets, we must introduce the concept of σ -fields.

Definition 1. A **σ -field**, also known as a **σ -algebra**, is a collection of subsets of Ω that is closed under countable unions of its elements; that is, a σ -field $\mathcal{F}$ satisfies the following properties:

1. $\emptyset, \Omega \in \mathcal{F}$.
2. If $A \in \mathcal{F}$, then $A^c \in \mathcal{F}$ (i.e., the complement of A is also in $\mathcal{F}$).
3. If $A_1, A_2, A_3, \dots \in \mathcal{F}$, then $\bigcup_{n=1}^{\infty} A_n \in \mathcal{F}$.

The elements of $\mathcal{F}$ are called **measurable sets**. The pair $(\Omega, \mathcal{F})$ is referred to as a **measurable space**.

In order to determine the size of measurable sets, we need to introduce the concept of probability.

Definition 2. A **probability measure** P on a measurable space $(\Omega, \mathcal{F})$ is a non-negative function defined on the collection $\mathcal{F}$, satisfying the following conditions:

1. $P(\Omega) = 1$.
2. For any set $A \in \mathcal{F}$, the probability of its complement is $P(A^c) = 1 - P(A)$.
3. If $A_1, A_2, \dots$ are pairwise disjoint sets in $\mathcal{F}$, then

$$P\left(\bigcup_{n=1}^{\infty} A_n\right) = \sum_{n=1}^{\infty} P(A_n),$$

which is the property of countable additivity, also called σ -additivity.

Note, that the property of σ -additivity is equivalent to finite additivity along with the continuity condition of probability measures. This condition states that if $A_1 \supseteq A_2 \supseteq \dots$ with $A = \bigcap_{n=1}^{\infty} A_n \in \mathcal{F}$, then

$$\lim_{n \rightarrow \infty} P(A_n) = P(A).$$

With the definition of a probability measure established, we can define probability spaces.

Definition 3. A **probability space** is a triplet $(\Omega, \mathcal{F}, P)$, where Ω denotes the sample space, $\mathcal{F}$ represents a σ -algebra of events, and P is a probability measure defined on $\mathcal{F}$.

An important σ -field in the theory of random variables and probability measures is the Borel σ -field, defined as follows.

Definition 4. The **Borel σ -field** (or **Borel σ -algebra**) on the real line $\mathbb{R}$, denoted as $\mathcal{B}$, is the smallest σ -field that contains all the open intervals in $\mathbb{R}$.

The concept of measurable functions is fundamentally associated with the Borel σ -algebra.

Definition 5. A function $X : \Omega \rightarrow \mathbb{R}$ is defined as a **measurable function** with respect to the σ -algebra $\mathcal{F}$ if, for every Borel set $B \subseteq \mathbb{R}$, the preimage $X^{-1}(B) = \{\omega \in \Omega : X(\omega) \in B\}$ belongs to $\mathcal{F}$. This condition ensures that the values of X on Ω align with the structure of the probability space $(\Omega, \mathcal{F}, P)$, making it possible to assign probabilities to the events defined by X .

Random variables are closely related to the concept of the Borel σ -field, and are the basic building blocks of normal random fields. Random variables are measurable functions, which means that they assign a numerical value to each outcome in a sample space in a way that preserves the structure of the probability space. The precise definition can be given as follows.

Definition 6. A *random variable* X defined on the probability space $(\Omega, \mathcal{F}, P)$ is a measurable function that maps from $(\Omega, \mathcal{F})$ to $(\mathbb{R}, \mathcal{B})$, where $\mathcal{B}$ is the Borel σ -algebra on the real line. This implies that for every Borel set $B \in \mathcal{B}$, the set $\{\omega : X(\omega) \in B\}$ is an element of $\mathcal{F}$.

It is important to mention, that instead of verifying this condition for all Borel sets, it suffices to check that for any real number x , the set $\{\omega : X(\omega) \leq x\}$ is contained in $\mathcal{F}$. In simpler terms, we can assign probabilities to sets of the form $\{X \leq x\}$ and $\{a < X \leq b\}$.

A stochastic or random process refers to a set of random variables that are indexed by a particular mathematical set $\mathcal{D} \subset \mathbb{R}$, known as the *index set*. A stochastic process can produce multiple outcomes due to its randomness, and a single outcome from this process is referred to, among other names, as a *sample function* or *realization*. Stochastic processes are used to represent systems that change over time with built-in randomness, such as stock market fluctuations, and weather changes.

Definition 7. Given a probability space $(\Omega, \mathcal{F}, P)$ and a measurable space $(\mathbb{R}, \Sigma)$, a *stochastic process* is defined as a collection of $\mathbb{R}$ -valued random variables, expressed as: $\{X(t) : t \in \mathcal{D}\}$, $\mathcal{D} \subset \mathbb{R}$.

Here

1. t usually represents *time* (continuous or discrete),
2. Ω is the sample space (contains the possible outcomes),
3. $\mathcal{F}$ and Σ are σ -algebras,
4. P is the probability measure.

A random field generalizes the concept of a random process by allowing the index set, denoted by $\mathcal{D}$, to take more abstract forms. For instance, $\mathcal{D}$ may be a subset of $\mathbb{R}^n$ for some $n \in \mathbb{N}$, or it can represent a metric graph, which will be further specified later.

Definition 8. In a probability space $(\Omega, \mathcal{F}, P)$, an $\mathbb{R}$ -valued *random field* is defined as a collection of $\mathbb{R}$ -valued random variables that are indexed by elements from a space $\mathcal{D}$. In particular, a random field X is expressed as $\{X(t) \in \mathbb{R} : t \in \mathcal{D}\}$.

A normal random field is a type of random field in which the relationships among random variables are governed by multivariate, or more abstract normal distributions. To characterize these distributions, it is important to define the expected value operator, and the covariance matrix, which captures the dependency structure among the variables.

Definition 9. The *expected value* of a random variable X is defined as:

$$\mathbb{E}[X] = \int_{\Omega} X \, dP,$$

where Ω is the sample space and P is the probability measure.

1. If X is a discrete random variable with probability function $p_X(x) = P(X = x)$, then the expectation becomes a sum:

$$\mathbb{E}[X] = \sum_x x p_X(x).$$

2. If X is continuous with probability density function $f_X(x) = \frac{dP}{dx}$, the expectation is expressed as an integral:

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x) \, dx.$$

Definition 10. The *covariance matrix*, denoted as K , is a square matrix that describes the covariance between pairs of elements in a vector X composed of random variables $X_1, X_2, \dots, X_n$. The elements of the matrix are defined by

$$K_{i,j} = \text{cov}[X_i, X_j] = \mathbb{E} \left[\left(X_i - \mathbb{E}[X_i] \right) \left(X_j - \mathbb{E}[X_j] \right) \right], \quad i, j \in \{1, 2, \dots, n\},$$

Note that if X_i and X_j are independent, then $K_{i,j} = 0$.

Definition 11. An n -dimensional vector of random variables $X = (X_1, X_2, \dots, X_n)$ is said to have a *multivariate normal distribution* if its probability density function can be given by

$$f_X(x) = \frac{1}{(2\pi)^{n/2} |K|^{1/2}} \cdot \exp \left(-\frac{1}{2} (x - \mu)^\top K^{-1} (x - \mu) \right),$$

where K is the covariance matrix, $|K|$ is the determinant of matrix K , and μ is the mean vector.

An important special case is the *standard normal distribution* $Z = (Z_1, Z_2, \dots, Z_n)$, the mean vector and covariance matrix of which is 0 and the identity matrix, respectively. The importance of the standard normal distribution is that any normal distribution X can be standardized as follows

$$Z = A^{-1}(X - \mu),$$

where the covariance matrix K of X can be expressed as $K = AA^\top$, for some $n \times n$ matrix A . In the simulations we will use this property backwards, since Z is cheap to sample, and thus if we know μ and A , then we can express X as

$$X = \mu + AZ.$$

Definition 12. A random field $X(t)$ is called a **Gaussian (normal) random field** if, for any finite collection of points $t_1, t_2, \dots, t_n \in \mathcal{D}$, the random vector

$$(X(t_1), X(t_2), \dots, X(t_n))$$

has a multivariate normal distribution.

2.2. Quantum Graphs

In this section, I will explain the concept of quantum graphs based on [17] and [14], as they are inherently tied to Gaussian random fields, as we will see later.

A quantum graph is a metric graph G equipped with an elliptic differential operator on each edge, along with standard vertex conditions. A compact metric graph G is composed of a finite set V of vertices and a finite set E of edges connecting these vertices in pairs. We consider a simple graph without any loops or multiple edges between vertices. Additionally, G is assumed to be directed, meaning each edge has an assigned orientation with specified initial and terminal vertices. Each edge $e \in E$ has an associated length $\ell_e \in (0, \infty)$ and a local coordinate $x \in [0, \ell_e]$. A function u defined on the metric graph G can then be represented as a vector of edge functions, which we denote as follows:

$$u = (u_e)_{e \in E}.$$

To specify the vertex conditions, let E_v denote the set of edges adjacent to a vertex $v \in V$. We consider functions that are continuous on G and satisfy the Neumann-Kirchhoff (also known as the standard) condition:

$$\sum_{e \in E_v} c_e(v) \cdot u'_e(v) = 0, \quad v \in V;$$

where the derivatives are assumed to be taken in directions pointing outward from the vertex. To express the vertex conditions more compactly, we define the vector of function values at each $v \in V$ as

$$U(v) = (u_e(v))_{e \in E_v} \in \mathbb{R}^{|E_v|},$$

and the bi-diagonal matrix

$$I_v = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \\ & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{(|E_v|-1) \times |E_v|}.$$

Then, the condition $I_v U(v) = 0 \in \mathbb{R}^{|\mathbb{E}_v|-1}$ implies that the function values along edges in $\mathbb{E}_v$ are equal at the vertex $v \in \mathbb{V}$. Similarly, we define

$$U'(v) = (u'_e(v))_{e \in \mathbb{E}_v} \in \mathbb{R}^{|\mathbb{E}_v|},$$

the vector of function derivatives at $v \in \mathbb{V}$, along with the row vector

$$C(v)^\top = (c_e(v))_{e \in \mathbb{E}_v} \in \mathbb{R}^{|\mathbb{E}_v|}.$$

Then $C(v)^\top U'(v) = 0$ ensures that u satisfies the Neumann-Kirchhoff conditions at each $v \in \mathbb{V}$. Thus, the quantum graph equations are formally written as

$$\begin{cases} -(c_e \cdot u'_e)'(x) + p_e(x) \cdot u_e(x) = f_e(x) & x \in (0, \ell_e), \quad e \in \mathbb{E} \\ 0 = I_v \cdot U(v) & \text{for } v \in \text{int}(\mathbb{G}), \\ 0 = C(v)^T \cdot U'(v) & \text{for } v \in \mathbb{V}, \end{cases} \quad (2.1)$$

where $\text{int}(\mathbb{G})$ is the set of vertices $v \in \mathbb{V}$ such that $|\mathbb{E}_v| > 1$.

In the following, I will define important function spaces containing functions defined on the graph $\mathbb{G}$. These spaces are essential for defining operators on $\mathbb{G}$.

1. **Hilbert space $L^2(a, b)$:**

- The space of real-valued square-integrable functions, equipped with the norm:

$$\|f\|_{L^2(a,b)}^2 = \int_a^b |f(x)|^2 dx, \quad f \in L^2(a, b).$$

2. **Banach space $L^\infty(a, b)$:**

- The space of real-valued essentially bounded functions, endowed with the norm:

$$\|f\|_{L^\infty(a,b)} = \text{ess sup}_{x \in (a,b)} |f(x)|, \quad f \in L^\infty(a, b).$$

3. **Sobolev space $H^k(a, b)$:**

- The space of real-valued square-integrable functions, whose generalized derivatives up to the k -th order are also square-integrable, with the norm:

$$\|f\|_{H^k(a,b)}^2 = \sum_{j=0}^k \|f^{(j)}\|_{L^2(a,b)}^2, \quad f \in H^k(a, b).$$

4. **Banach space $C[a, b]$:**

- The space of real-valued continuous functions, equipped with the supremum norm.

5. Banach spaces associated with a graph G :

- $L^2(G)$ is defined as:

$$L^2(G) = \bigoplus_{e \in E} L^2(0, \ell_e).$$

- $L^\infty(G)$ is defined as:

$$L^\infty(G) = \bigoplus_{e \in E} L^\infty(0, \ell_e).$$

- $H^k(G)$ is defined as:

$$H^k(G) = \bigoplus_{e \in E} H^k(0, \ell_e).$$

- The norms for these spaces are given by:

$$\begin{aligned} \|u\|_{L^2(G)}^2 &:= \sum_{e \in E} \|u_e\|_{L^2(0, \ell_e)}^2, & u = (u_e)_{e \in E} \in L^2(G), \\ \|u\|_{L^\infty(G)}^2 &:= \max_{e \in E} \|u_e\|_{L^\infty(0, \ell_e)}^2, & u = (u_e)_{e \in E} \in L^\infty(G), \\ \|u\|_{H^k(G)}^2 &:= \sum_{e \in E} \|u_e\|_{H^k(0, \ell_e)}^2, & u = (u_e)_{e \in E} \in H^k(G). \end{aligned}$$

- Both $L^2(G)$ and $H^k(G)$ are Hilbert spaces when equipped with their respective inner products.

6. Space of continuous functions on G :

- The space $C(G)$ is defined as:

$$C(G) := \left\{ u = (u_e)_{e \in E} \mid I_v U(v) = 0, \forall v \in V : u_e \in C[0, \ell_e] \right\}.$$

Building on the function spaces defined above, we can rewrite (2.1) by defining the T operator [16]. Let $T : D(T) \subset L^2(G) \mapsto L^2(G)$ be the second-order elliptic operator that acts on $u \in D(T)$ as

$$(Tu)(x) = -(c_e u'_e)'(x) + p_e(x)u_e(x). \quad (2.2)$$

Here

- $D(T) = C(G) \cap \left\{ u \in H^2(G) \mid \sum_{e \in E_v} c_e(v)u'_e(v) = 0 \text{ for } v \in V \right\}$,
- $c = (c_e)_{e \in E}$ is a positive Lipschitz function,
- $p = (p_e)_{e \in E} \in L^\infty(G)$ with $\text{ess inf}_{x \in G} p(x) \geq p_0 > 0$.

Under these conditions T is a densely defined, self-adjoint, and positive definite operator with a compact inverse [15].

Many real-world phenomena exhibit random fluctuations in both time and space, which motivates the extension of partial differential equations (PDEs) to situations where functions are less smooth. Stochastic partial differential equations (SPDEs) merge the concepts of PDEs with random fields, allowing for the analysis of systems influenced by randomness. In particular, defining SPDEs on metric graphs, resulting in a *stochastic quantum graph*, is a common approach across various applications. By introducing randomness, SPDEs provide a robust framework for modeling complex phenomena, such as the evolution of stock prices.

Let us consider the following *fractional SPDE*:

$$T^\beta u(x) = W(x),$$

where $W(x)$ is Gaussian white noise defined on $\mathbf{G}$, and $\beta \in \mathbb{R}$. The properties of T combined with the fact that $L^2(\mathbf{G})$ is a separable Hilbert space implies that there exists an orthonormal basis $\{e_j\}_{j \in \mathbb{N}}$ formed by the eigenvectors of T . The pairs of eigenvalues and eigenvectors $\{(\lambda_j, e_j)\}_{j \in \mathbb{N}}$ can be organized such that the positive eigenvalues $\{\lambda_j\}_{j \in \mathbb{N}}$ are arranged in a nondecreasing sequence:

$$0 < \lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_j \leq \lambda_{j+1} \leq \dots, \quad \lim_{j \rightarrow \infty} \lambda_j = \infty.$$

We assume that the growth rate of the eigenvalues is described by $\lambda_j \propto j^2$, meaning there are constants $c_\lambda, C_\lambda > 0$ such that

$$c_\lambda j^2 \leq \lambda_j \leq C_\lambda j^2 \quad \forall j \in \mathbb{N}.$$

For $\beta > 0$ and $\phi \in D(T^\beta) := \{\psi \in L^2(\mathbf{G}) : \sum_{j \in \mathbb{N}} \lambda_j^{2\beta} (\psi, e_j)_{L^2(\mathbf{G})}^2 < \infty\}$, the operation of the fractional power operator $T^\beta : D(T^\beta) \rightarrow L^2(\mathbf{G})$ is defined by

$$T^\beta \phi := \sum_{j \in \mathbb{N}} \lambda_j^\beta (\phi, e_j)_{L^2(\mathbf{G})} e_j.$$

The white noise W can formally be represented by the Karhunen–Loève expansion (that is, representing a random process/field as a series of orthogonal components) with respect to the orthonormal eigenbasis $\{e_j\}_{j \in \mathbb{N}} \subset L^2(\mathbf{G})$ of T ,

$$W = \sum_{j \in \mathbb{N}} \xi_j e_j,$$

where $\{\xi_j\}_{j \in \mathbb{N}}$ is a sequence of independent real-valued standard normally distributed random variables.

2.3. Iterative Linear Solvers

A system of linear equations $Ax = b$ is called sparse, if only relatively small number of its matrix elements are nonzero. In this section I will cover the basics of how iterative sparse linear solvers work, based on [20].

For solving the $Ax = b$ equation, two main methods are used:

1. *Direct methods*, like Gaussian elimination, are robust, but computationally expensive with complexities of $\mathcal{O}(n^3)$ for dense and $\mathcal{O}(n^2)$ for sparse matrices. However, $\mathcal{O}(n)$ can be achieved with the Thomas algorithm in special cases like tridiagonal matrices arising from the finite element approximation of one-dimensional PDEs.
2. *Iterative methods* start with an initial guess and refine the solution through repeated steps. They typically involve sparse matrix-vector multiplications, leading to lower complexity of $\mathcal{O}(n)$ per iteration and overall complexity of $\mathcal{O}(kn)$, where k is the number of iterations. If k is much smaller than n , iterative methods can efficiently handle large-scale computations.

The general algorithm for solving sparse linear systems is as follows:

1. Input: initial solution $x^{(0)}$, tolerance ϵ , maximum iteration number K .
2. Initialize the residual:

$$r^{(0)} \leftarrow (b - Ax^{(0)}).$$

3. For $k = 1$ to K do:

- (a) Update the solution with the formula

$$x^{(k)} \leftarrow (G^{(k)}x^{(k-1)} + d^{(k)}),$$

where $G^{(k)}$ and $d^{(k)}$ denote the iteration matrix and vector at step k .

- (b) Update the residual:

$$r^{(k)} \leftarrow (b - Ax^{(k)}).$$

- (c) Check for convergence:

$$\text{if } \|r^{(k)}\| \leq \epsilon \|r^{(0)}\| \text{ then finish.}$$

Different $G^{(k)}$ iteration matrices and $d^{(k)}$ iteration vectors correspond to different iterative methods. For example, one of the simplest methods is the *Richardson iteration*,

where $G^{(k)}$ is the identity matrix, and $d^{(k)} = \omega (b - Ax^{(k-1)})$ with $\omega \in \mathbb{R}$. Therefore, in this method, the solution is updated by adjusting in the direction of the residual, scaled by a relaxation factor ω . The update rule is expressed as

$$x^{(k)} = x^{(k-1)} + \omega (b - Ax^{(k-1)}).$$

The choice of ω is critical, because a well-chosen ω can greatly speed up convergence. Small values of ω can result in slower convergence, while large values can cause divergence or instability. If A has real, positive eigenvalues, the optimal ω , denoted by ω_{opt} can be given by

$$\omega_{\text{opt}} = \frac{2}{\lambda_{\max} + \lambda_{\min}}.$$

Currently, the most widely used iterative methods are Krylov subspace methods, including Conjugate Gradient (CG), Minimal Residual Method (MINRES) and their variations. The convergence rate of these methods is significantly influenced by the condition number of the matrix $\kappa(A)$, defined as the product of the matrix norm and the norm of its inverse: $\kappa(A) = \|A\| \|A^{-1}\|$. The specific definition of the condition number varies based on the chosen norm; for instance, in the case of a symmetric positive definite matrix, the condition number can be expressed as the ratio of its largest eigenvalue to its smallest eigenvalue: $\kappa(A) = \lambda_{\max}/\lambda_{\min}$. Different iterative methods are affected by the matrix condition number to a different extent. For example, let us compare the error estimate values for the conjugate gradient method and the Richardson iteration. Let $\|e_0\| = \|x^{(0)} - x^*\|$, where x^* represents the exact solution, and $x^{(0)}$ is the initial approximation, with e_0 denoting the error.

1. *Richardson iteration*

$$\|e_k\| \approx \left(\frac{\kappa(A) - 1}{\kappa(A) + 1} \right)^k \|e_0\|$$

2. *Conjugate gradient method*

$$\|e_k\| \approx 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|e_0\|$$

Therefore, it is not hard to show that the conjugate gradient method is generally more efficient than the Richardson iteration.

In large-scale ill-conditioned problems, where $\kappa(A) \gg 1$, convergence tends to be slow. This slow convergence often requires the implementation of preconditioning techniques. A preconditioning technique involves constructing a matrix $M \in \mathbb{R}^{n \times n}$ such that $M^{-1} \approx$

A^{-1} . If the approximation is accurate enough, the following preconditioned system can have a much better condition number:

$$M^{-1}Ax = M^{-1}b.$$

The challenge lies in constructing an effective preconditioner matrix. On one hand, it is essential that the $M^{-1} \approx A^{-1}$ approximation is accurate; on the other hand, constructing M^{-1} should not be computationally expensive. However, these two requirements can often be contradictory. For instance, when $M = A$, the approximation is ideal, but computing M^{-1} is really expensive. To balance these concerns, it is often necessary to construct appropriate preconditioners based on the matrix properties and application characteristics:

1. **Jacobi Preconditioning:** Also known as diagonal preconditioning, this is one of the simplest forms, where:

$$M = D,$$

D being the diagonal matrix containing the diagonal entries of A . This preconditioner is effective when A is diagonally dominant, meaning that for each row of the matrix, the absolute value of the diagonal entry is greater than or equal to the sum of the absolute values of all other (non-diagonal) entries in that row:

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i.$$

2. **LU Factorization:** This method factors A into a product of a lower triangular matrix L and an upper triangular matrix U :

$$A = LU.$$

Direct LU factorization can serve as a preconditioner but can be impractical for very large, sparse systems due to the fact that L and U can have an increased fill-in even if A is sparse. This increase in storage and computational requirements motivates the use of the incomplete LU factorization.

3. **Incomplete LU (ILU) Factorization:** This method computes an approximate factorization, where $\tilde{L}$ and $\tilde{U}$ are sparse lower and upper triangular matrices:

$$M = \tilde{L}\tilde{U} \approx A.$$

The ILU factorization is "incomplete" because it limits fill-in by only retaining certain entries in $\tilde{L}$ and $\tilde{U}$, making it a sparser approximation of A . This is done

according to predefined rules, such as retaining entries within certain bands or positions, which reduces memory usage and computational costs compared to full LU factorization. Consequently, ILU is more efficient in sparse matrices that are diagonally dominant, banded, or have low fill-in characteristics.

4. **Incomplete Cholesky Factorization:** For symmetric positive definite matrices:

$$M = \tilde{L}\tilde{L}^T \approx A,$$

where $\tilde{L}$ is a sparse lower triangular matrix. The most efficient sparse positive definite matrix structures for Cholesky decomposition are those that exhibit banded patterns, diagonal dominance, block structures, and a concentration of non-zero elements near the diagonal.

The Jacobi preconditioner is constructed quickly, requiring only the diagonal elements of the matrix A , resulting in a time complexity of $O(n)$ for an $n \times n$ matrix. In contrast, computing the factorized matrices for the other introduced preconditioning methods is generally slower, with a worst-case scenario of $\mathcal{O}(n^3)$ complexity. However, these methods often give better estimates for M .

Finite Element Method for Quantum Graphs

In this chapter, I will introduce the finite element method (FEM), based on [16], [21], [22]. The finite element method is a widely used technique for the numerical solution of differential equations. The essence of the FEM is that the domain, on which the differential equation is defined, is partitioned into smaller regions (e.g. intervals in the case of one-dimensional problems), and the solution to the equation on these elements is approximated using a low-order polynomial function. The major advantages of the method:

1. It is robust w.r.t. the shape of the domain, and different types of boundary conditions can be easily handled with it.
2. A more accurate estimate can be easily achieved by increasing the degree of the approximating polynomial, or refining the domain discretization; which is essentially done with the help of a posteriori error estimation and adaptive mesh refinement.

3.1. Finite Element Method on One Edge

First, let us consider an elliptic equation on the interval $[a, b]$; that is, we consider the equation

$$-(c \cdot u')(x) + p(x) \cdot u(x) = f(x), \quad x \in [a, b], \quad (3.3)$$

where $u \in H^2(a, b)$, $f \in L^2(a, b)$, c is a positive Lipschitz function, and p is essentially bounded with $\text{ess inf}_{x \in [a, b]} p(x) \geq p_0 > 0$. Without the loss of generality, let us assume that there is a Dirichlet boundary condition at point a and a Neumann boundary condition at point b :

$$\begin{cases} u(a) = u_a, \\ c(b) \cdot u'(b) = q_b. \end{cases}$$

We need to determine the weak form of the differential equation. To achieve this, we first have to multiply both sides of (3.3) by the test function $v \in H^1(a, b)$, which has the property that it takes the value 0 at the locations of the Dirichlet boundary conditions. After multiplying by v , we integrate both sides of the equation resulting in

$$-\int_a^b (c \cdot u')'(x) \cdot v(x) \, dx + \int_a^b p(x) \cdot v(x) \cdot u(x) \, dx = \int_a^b f(x) \cdot v(x) \, dx.$$

On the left side of the equation, the first term can be expressed using integration by parts

$$-\left[c(x) \cdot u'(x) \cdot v(x)\right]_a^b + \int_a^b c(x) \cdot u'(x) \cdot v'(x) \, dx + \int_a^b p(x) \cdot v(x) \cdot u(x) \, dx = \int_a^b f(x) \cdot v(x) \, dx.$$

Using the Neumann boundary condition we obtain

$$\int_a^b c(x) \cdot u'(x) \cdot v'(x) \, dx + \int_a^b p(x) \cdot v(x) \cdot u(x) \, dx = \int_a^b f(x) \cdot v(x) \, dx + q_b \cdot v(b). \quad (3.4)$$

Equation (3.4) is the weak or variational formulation of equation (3.3).

The main idea of the FEM is to divide the domain into smaller parts to approximate the solution of the given differential equation. In one dimension, if the problem is defined on the interval $[a, b]$, we divide this interval into N subintervals using $N + 1$ division points: $a = x_1 < x_2 < \dots < x_{N+1} = b$. Let the length of the j -th subinterval be denoted by $h_j = x_{j+1} - x_j$, where $j = 1, 2, \dots, N$. Using the division points $x_1, \dots, x_{N+1}$, we can define the basis functions of the finite element method as follows:

$$\psi_j(x) = \begin{cases} \frac{x - x_{j-1}}{h_j} & \text{if } x_{j-1} \leq x < x_j, \\ \frac{x_{j+1} - x}{h_{j+1}} & \text{if } x_j \leq x < x_{j+1}, \\ 0 & \text{otherwise.} \end{cases} \quad j = 2, \dots, N, \quad (3.5)$$

Note, that $\psi_1(x)$ and $\psi_{N+1}(x)$ can be defined similarly with omitting the left and right subintervals, respectively. However, since we have a Dirichlet boundary condition at $x = a$, $\psi_1(x) \equiv 0$ (Figure 3.1).

Let us assume that $U(x)$ is a continuous function, which is linear on the intervals $[x_i, x_{i+1}]$ for $i = 1, 2, \dots, N$. Then, $U(x)$ can be expressed as the linear combination of the basis functions $\{\psi_j(x)\}_{j=1,2,\dots,N+1}$:

$$U(x) = \sum_{j=1}^{N+1} U_j \psi_j(x). \quad (3.6)$$

We seek a solution $u \in H^1(a, b)$ of (3.4), such that it satisfies the Dirichlet boundary conditions. It is important to mention, that the weak formulation should hold for all suitable v functions. Consequently, the weak formulation remains an infinite-dimensional

problem, which we cannot solve directly. However, continuous piecewise linear functions can be written as linear combinations of the FEM basis functions, so it is sufficient for the equation to be satisfied only for the basis functions, rather than for all v . Therefore, by substituting the functions $\{\psi_i\}_{i=1,2,\dots,N+1}$ into v and approximating the function $u(x)$ with the continuous and piecewise linear function $U(x)$, we obtain a finite-dimensional problem that we can solve:

$$\int_a^b c(x) \cdot U'(x) \cdot \psi_i'(x) + p(x) \cdot \psi_i(x) \cdot U(x) dx = \int_a^b f(x) \cdot \psi_i(x) dx + q_b \cdot \psi_i(b).$$

By utilizing the linearity between the summation and the integral, we obtain

$$\sum_{j=1}^{N+1} U_j \left(\int_a^b c(x) \cdot \psi_j'(x) \cdot \psi_i'(x) + p(x) \cdot \psi_i(x) \cdot \psi_j(x) dx \right) = \int_a^b f(x) \cdot \psi_i(x) dx + q_b \cdot \psi_i(b).$$

Let us introduce the following notations:

$$a_{ij} = \int_a^b c(x) \cdot \psi_j'(x) \cdot \psi_i'(x) + p(x) \cdot \psi_i(x) \cdot \psi_j(x) dx \quad \text{and} \quad b_i = \int_a^b f(x) \cdot \psi_i(x) dx. \quad (3.7)$$

Using these definitions, we obtain a system of linear equations $A \cdot U = b$, which is given by

$$\sum_{j=1}^{N+1} a_{ij} \cdot U_j = b_i + q_b \cdot \psi_i(b).$$

In the next steps of the solution, we will ignore the effects of the boundary conditions, since it is sufficient to consider them only in the final steps of the derivation; instead, we will focus on calculating the integrals defined in (3.7). Let us examine the term a_{ij} and break the integral into the sum of integrals defined over the subintervals:

$$a_{ij} = \sum_{k=1}^N \left(\int_{x_k}^{x_{k+1}} c(x) \cdot \psi_j'(x) \cdot \psi_i'(x) + p(x) \cdot \psi_i(x) \cdot \psi_j(x) dx \right).$$

Since $\psi_j(x)$ has the property that it is non-zero only on the interval (x_{j-1}, x_{j+1}) , the products $\psi_i(x) \cdot \psi_j(x)$ and $\psi_i'(x) \cdot \psi_j'(x)$ are only non-zero for $i = j-1, j, j+1$. Hence, if we arrange the a_{ij} values into a matrix, we would obtain a tridiagonal matrix as a result.

Calculating its elements:

$$a_{ii} = \int_{x_{i-1}}^{x_{i+1}} c(x) \cdot \psi_i'(x) \cdot \psi_i'(x) + p(x) \cdot \psi_i(x) \cdot \psi_i(x) dx \quad i = 1, 2, \dots, N+1$$

$$a_{i(i+1)} = a_{(i+1)i} = \int_{x_i}^{x_{i+1}} c(x) \cdot \psi_i'(x) \cdot \psi_{i+1}'(x) + p(x) \cdot \psi_i(x) \cdot \psi_{i+1}(x) dx \quad i = 1, \dots, N$$

However, the values a_{11} and $a_{(N+1)(N+1)}$ need to be treated separately, since in this case, the corresponding basis functions are non-zero only on the intervals $[x_1, x_2)$ and $(x_N, x_{N+1}]$, respectively.

$$a_{11} = \int_{x_1}^{x_2} c(x) \cdot \psi_1'(x) \cdot \psi_1'(x) + p(x) \cdot \psi_1(x) \cdot \psi_1(x) dx$$

$$a_{(N+1)(N+1)} = \int_{x_N}^{x_{N+1}} c(x) \cdot \psi'_{N+1}(x) \cdot \psi'_{N+1}(x) + p(x) \cdot \psi_{N+1}(x) \cdot \psi_{N+1}(x) dx$$

By approximating the above integrals using numerical integration (the trapezoidal rule), we obtain

$$a_{ii} = \begin{cases} \frac{c(x_1) + c(x_2)}{2h_1} + \frac{p(x_1)}{2} \cdot h_1 & i = 1 \\ \frac{c(x_i) + c(x_{i-1})}{2h_{i-1}} + \frac{p(x_i)}{2} \cdot h_{i-1} + \frac{c(x_{i+1}) + c(x_i)}{2h_i} + \frac{p(x_i)}{2} \cdot h_i & i = 2, \dots, N \\ \frac{c(x_N) + c(x_{N+1})}{2h_N} + \frac{p(x_{N+1})}{2} \cdot h_N & i = N + 1 \end{cases}$$

The next step is to compute the values of b_i :

$$b_i = \sum_{k=1}^N \left(\int_{x_k}^{x_{k+1}} f(x) \cdot \psi_i(x) dx \right) \Rightarrow b_i = \begin{cases} \frac{h_1 \cdot f(x_1)}{2} & i = 1 \\ \frac{f(x_i)}{2} \cdot (h_{i-1} + h_i) & i = 2, \dots, N \\ \frac{h_N \cdot f(x_{N+1})}{2} & i = N + 1 \end{cases}$$

The final step of the derivation is to consider the effect of the boundary conditions. The Neumann boundary condition only alters the previous results in that we need to add the value q_b to the right side of the $(N + 1)$ -th equation in $A \cdot u = b$. For the other equations, $q_b \cdot \Phi_i(b)$ is zero since $\Phi_i(b) = 0$ for $i \neq N + 1$. Due to the Dirichlet condition at point a , the following holds: $U_1 = u_a$. To represent this in the system of linear equations, we must make the following adjustments: $a_{11} = 1$, $a_{12} = 0$, and $b_1 = u_a$.

After solving the linear system for U , we can approximate $u(x)$ by computing $U(x)$, given by (3.6).

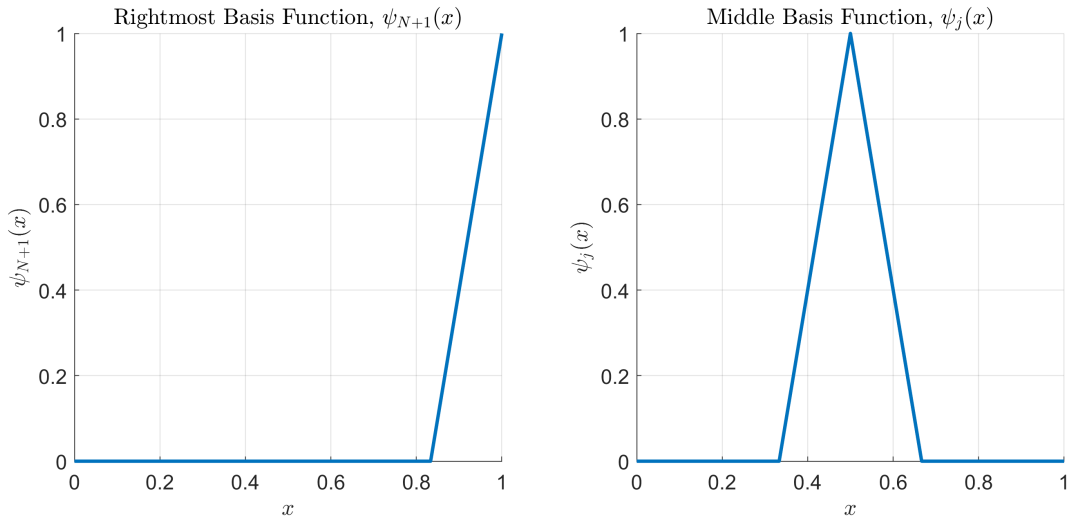


Figure 3.1: FEM basis functions defined on the $[a,b]$ interval

3.2. Extension for Quantum Graphs

In this section, I will extend the solution previously discussed for a single edge to the entire quantum graph. Our goal is to solve the system of equations (2.1) defined in Section 2.2.

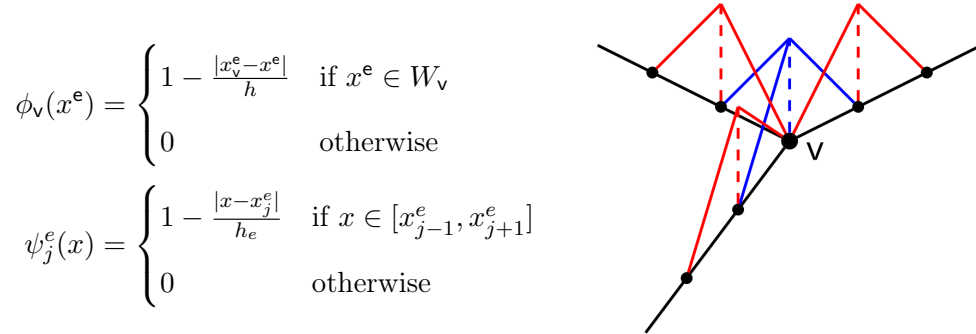
The weak form of (2.1) can be defined by the sum of the weak forms on the edges:

$$a(u, v) = \sum_{e \in E} \left(\int_e c_e(x) \cdot u'_e(x) \cdot v'_e(x) dx + \int_e p_e(x) \cdot u_e(x) \cdot v_e(x) dx \right), \quad (3.8)$$

where $D(a) = (H^1(G) \cap C(G)) \times (H^1(G) \cap C(G))$. Let $f(v) = \langle f, v \rangle_{L^2(G)}$ (the scalar product of f , and v in the $L^2(G)$ function space). Then, the weak or variational form is expressed as

$$a(u, v) = f(v), \quad v \in D(a). \quad (3.9)$$

Equation (3.9) is also well-posed and the unique solution of (3.9) coincides with that of (2.1) [15]. To ensure continuity, the basis functions meeting at vertex v will be treated as a single function. The location x_v^e is either 0 or ℓ_e depending on the orientation of the edge.



On the figure, the blue basis function corresponds to ϕ_v , while the red ones correspond to ψ_j^e . The domain W_v is the union of the nearest subintervals of edges connected to the vertex $v \in V$, given by

$$W_v = \left(\bigcup_{e \in E: v_a^e = v} [v, x_1^e] \right) \cup \left(\bigcup_{e \in E: v_b^e = v} [x_{n_e-1}^e, v] \right). \quad (3.10)$$

We search for the solution $a(w_h, v) = f(v)$ in the following form:

$$w_h(x) = \sum_{e \in E} \sum_{j=1}^{n_e-1} \alpha_j^e \cdot \psi_j^e(x) + \sum_{v \in V} \beta_v \cdot \phi_v(x).$$

Finally, I would like to present the structure of the linear system that we need to solve. Similarly as before, we substitute the basis functions into the weak formulation (3.9) and observe that most basis functions do not interact. Thus, the system takes the form:

$$A \cdot \mathbf{u} = \mathbf{f}$$

$$\mathbf{u} = \begin{bmatrix} u_E \\ u_V \end{bmatrix}, \quad u_E = \begin{bmatrix} u^{e_1} \\ u^{e_2} \\ \vdots \\ u^{e_{|E|}} \end{bmatrix}, \quad u^e = \begin{bmatrix} u_1^e \\ u_2^e \\ \vdots \\ u_N^e \end{bmatrix}, \quad u_V = \begin{bmatrix} u_{v_1} \\ u_{v_2} \\ \vdots \\ u_{v_{|V|}} \end{bmatrix} \quad (3.11)$$

$$\mathbf{f} = \begin{bmatrix} f_E \\ f_V \end{bmatrix}, \quad f_E = \begin{bmatrix} f^{e_1} \\ f^{e_2} \\ \vdots \\ f^{e_{|E|}} \end{bmatrix}, \quad f_V = \begin{bmatrix} f_{v_1} \\ f_{v_2} \\ \vdots \\ f_N^e \end{bmatrix}, \quad f^e = \begin{bmatrix} f_1^e \\ f_2^e \\ \vdots \\ f_{v_{|V|}}^e \end{bmatrix} \quad (3.12)$$

$$A = \begin{bmatrix} A_E & A_{EV} \\ A_{VE} & A_V \end{bmatrix}. \quad (3.13)$$

Here

1. The vectors f_k^e and f_v are given by

$$f_k^e = \int_e f \psi_k^e dx, \quad f_v = \int_{W_v} f \phi_v dx. \quad (3.14)$$

2. The matrix $A_E = \text{diag}(A_e)_{e \in E}$ is block diagonal, and the entries of the tridiagonal matrix A_e are given by

$$[A_e]_{jk} = \int_e c_e \psi_j^{e'} \psi_k^{e'} dx + \int_e p_e \psi_j^e \psi_k^e dx, \quad j, k = 1, 2, \dots, N-1. \quad (3.15)$$

3. The entries of the blocks of $A_{EV}^\top = A_{VE} = (A_e)_{e \in E}$ are given by

$$[A_e]_{vk} = \int_{W_v} c_e \phi_v' \psi_k^{e'} dx + \int_{W_v} p_e \phi_v \psi_k^e dx, \quad k = 1, 2, \dots, N-1, \quad v \in V. \quad (3.16)$$

4. The entries of the diagonal matrix $A_V = \text{diag}(A_v)_{v \in V}$ are given by

$$A_v = \int_{W_v} c_e \phi_v' \phi_v' dx + \int_{W_v} p_e \phi_v \phi_v dx. \quad (3.17)$$

Explanation:

1. The block-diagonal matrix A_E and the vector f_E characterize the interactions between the basis functions within the edges.
2. The matrices A_{EV} and A_{VE} describe the connections between the edges and the vertices.
3. The diagonal matrix A_V and the vector f_V represent the interactions between the vertices.

3.3. Implementation

I implemented the mathematical method described above in C++ to solve the system defined in Section 2.2. Efficiency was key in this project, which is why I chose a C++ environment, allowing me to take advantage of the language's flexibility. For fast execution of mathematical operations, I chose the **Eigen** library, specifically I have worked with sparse data structures - this is a reasonable choice, since the matrix A defined in (3.13) contains a small number of nonzero elements for most of the graphs. But before going into the details, I want to describe the general (abstract) algorithm of a FEM solver for graphs:

1. Using the relationships discussed in Section 3.1 and Section 3.2, we calculate one element of the matrix A or the vector $\mathbf{f}$.
2. With an appropriate index function, we determine the location of the calculated element in the matrix A or in the vector $\mathbf{f}$.
3. The calculated element is additively inserted into the determined location, and if there are more elements to be calculated, the process is repeated from step 1.
4. Using the filled matrix A and vector $\mathbf{f}$, we compute the solution vector $\mathbf{u}$ defined in (3.11) with a linear equation solver, which approximates the actual solution.

My implementation returns an **Eigen::VectorXd** vector corresponding to the $\mathbf{u}$ vector, and requires the following input parameters:

1. **std::function<double(double)>** f , p , c - the functions $c(x)$, $p(x)$, and $f(x)$, which define the specific differential equation to be solved.
2. An **std::vector<std::tuple<int,int>>**. The vector contains **std::tuple** elements, each representing an edge. The first element of the tuple is the source vertex from which the edge points to the target vertex, which is the second element of the tuple. I identified the vertices with integer values.

First, I implemented the basic recommendation of the **Eigen** library for computing the sparse matrix A and the vector $\mathbf{f}$. This approach collects the calculated matrix elements into an **Eigen** triplet list, where each triplet represents the $(x, y, value)$ tuple. Here, x and y specify the position of **value** in the sparse matrix A . At the end of the process, this list is used to create the sparse matrix A , with the vector $\mathbf{f}$ being constructed through

direct insertion. From this point forward, we will refer to this approach as the *general triplet method*. When prior knowledge about the structure or values of the matrix A is unavailable, this is one of the most efficient approaches for sparse matrix construction. However, if some useful knowledge about the structure of the matrix A is available, we can make this process faster.

More specifically, when the number of elements in each column is known, then the column-major sparse matrix A can be more efficiently constructed. In case of the stiffness matrix A , the first $|A| - |V|$ columns contain exactly 3 non-zero elements each, due to the tridiagonal nature of the blocks corresponding to the edges; and the last $|V|$ columns each contain $|E_v| + 1$ non-zero elements. To see this: the matrix A_{EV} has $|E_v|$ non-zero elements in its column corresponding to vertex v , while the matrix A_{VV} contributes one non-zero element per column, as it is diagonal. To understand why the A_{EV} matrix contains $|E_v|$ non-zero elements per column, we must consider the role of this matrix. It describes the interactions between vertices and edges. Each vertex v is connected by $|E_v|$ edges, indicating that there will be $|E_v|$ non-zero elements in the corresponding column.

This observation is particularly useful because if the number of non-zero elements in the columns of a sparse matrix is known in advance and insertions occur column-wise, the cost of the insertion operations is $\mathcal{O}(1)$. This fact is clearly outlined in the documentation of the `Eigen` library. From this point forward, we will refer to this approach as the *column-wise insertion method*. It is important to note, that in a sense, this approach is a special case of the general triplet method, where insertions are performed in an optimal order.

Once the sparse matrix A and the vector $\mathbf{f}$ representing the linear system are constructed, the system is solved using the `Eigen SimplicialLDLT` solver. This solver takes advantage of the special properties of the matrix A . It was noticeable that since `SimplicialLDLT` uses sparse matrices, the solution time increased intensely as the matrix density grew. This explains why solving a complete graph is much more computationally expensive than solving a sparser graph with the same number of vertices but fewer edges, as the matrix A becomes denser with the graph.

3.4. Results, Efficiency

To test the hypothesis that the column-wise insertion is indeed faster, I implemented both method and measured their performance on graphs of various sizes. The measurements showed that constructing the A matrix and the $\mathbf{f}$ vector using the column-wise insertion

method was approximately 1.5 times faster compared to the general triplet method. Below is a comparison table demonstrating this fact. The table contains two test cases: I measured the execution times on a $|V| \times |V|$ square grid graph and on a complete graph with $|V|$ vertices.

Notations: $N(\sqrt{|V|})$ square grid graph, $T(|V|)$ complete graph

Graph	Measurements	Intervals	Triplet	Column-wise	Ratio
$N(1000)$	25	100	15.4986 s	10.0263 s	1.55
$N(1000)$	50	50	7.4007 s	5.0794 s	1.46
$N(250)$	75	100	0.9351 s	0.6585 s	1.42
$T(500)$	50	50	0.4891 s	0.3235 s	1.51
$T(100)$	1000	100	0.03877 s	0.0262 s	1.48

As the number of subintervals or vertices increases, both processing time and memory requirements become significant. While the column-wise insertion method is more memory-efficient than the general triplet method, it still faces limitations with large-scale graphs. For example, consider an extreme case in which solving a differential equation on a large graph would require multiple terabytes of RAM. To illustrate this, imagine modeling a biological neural network with approximately 10^8 neurons and 10^{14} synaptic connections. This problem requires roughly 10^3 elements per edge for precision. This results in a stiffness matrix A of dimension $10^{17} \times 10^{17}$, containing around 3×10^{17} nonzero elements. Consequently, the memory requirement becomes approximately 3 exabytes (EB). To manage memory more efficiently in such cases, the graph can be decomposed and solved incrementally. In the following section, I introduce a solution based on this principle: the Neumann–Neumann domain decomposition approach.

3.5. Neumann - Neumann Domain Decomposition

The core concept of this approach involves partitioning the graph G into disjoint subgraphs $G_i = (V_i, E_i)$ for $i = 1, 2, \dots$, such that the subgraphs do not overlap in terms of edges. Each G_i is a metric graph on its own, and a subgraph can consist of a single edge. We define the set of vertices that lie on the boundary of more than one subgraph as Γ , referring to it as the interface. The corresponding values of the function at these vertices are denoted by u_Γ . The Neumann-Neumann method is based on tracking the values at the interface and iteratively updating them by examining the deviations from the

Neumann-Kirchhoff condition. Formally, the algorithm begins with an initial estimate $u_\Gamma^0 = 0$. Consider the linear equation $A\mathbf{u} = \mathbf{f}$ derived from the finite element approximation of an elliptic problem on the quantum graph $G = (V, E)$, where A is a symmetric, positive definite matrix. To simplify the discussion, we assume that G is divided into two non-overlapping subgraphs $G_i = (V_i, E_i)$ for $i = 1, 2$. This implies that the edge set satisfies $E = E_1 \cup E_2$ with $E_1 \cap E_2 = \emptyset$, and the interface is given by $\Gamma = V_1 \cap V_2$. Then the equation system takes the form

$$A = \begin{bmatrix} A_{II}^{(1)} & 0 & A_{I\Gamma}^{(1)} \\ 0 & A_{II}^{(2)} & A_{I\Gamma}^{(2)} \\ A_{\Gamma I}^{(1)} & A_{\Gamma I}^{(2)} & A_{\Gamma\Gamma} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} u_I^{(1)} \\ u_I^{(2)} \\ u_\Gamma \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} f_I^{(1)} \\ f_I^{(2)} \\ f_\Gamma \end{bmatrix},$$

where the $A_{II}^{(i)}$, $A_{I\Gamma}^{(i)}$, $A_{\Gamma I}^{(i)}$, $A_{\Gamma\Gamma}$, $f_I^{(i)}$, and f_Γ for $i = 1, 2$ can be obtained from the systems defined on the subgraphs:

$$\begin{cases} f^{(i)} = \begin{bmatrix} f_I^{(i)} \\ f_\Gamma^{(i)} \end{bmatrix}, \quad A^{(i)} = \begin{bmatrix} A_{II}^{(i)} & A_{I\Gamma}^{(i)} \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} \end{bmatrix} \\ A_{\Gamma\Gamma} = A_{\Gamma\Gamma}^{(1)} + A_{\Gamma\Gamma}^{(2)} \\ f_\Gamma = f_\Gamma^{(1)} + f_\Gamma^{(2)} \end{cases}.$$

Consequently, we would like to find the solution for the following coupled problem:

$$\begin{cases} A_{II}^{(i)} u_I^{(i)} + A_{I\Gamma}^{(i)} u_\Gamma^{(i)} = f_I^{(i)}, \quad i = 1, 2 \\ u_\Gamma^{(1)} = u_\Gamma^{(2)} =: u_\Gamma \\ A_{\Gamma I}^{(1)} u_I^{(1)} + A_{\Gamma\Gamma}^{(1)} u_\Gamma^{(1)} - f_\Gamma^{(1)} = -A_{\Gamma I}^{(2)} u_I^{(2)} - A_{\Gamma\Gamma}^{(2)} u_\Gamma^{(2)} + f_\Gamma^{(2)} =: \lambda_\Gamma \end{cases}. \quad (3.18)$$

If the boundary values u_Γ are known, we can compute the approximate solution within the domains by solving two independent Dirichlet problems. Since we compute the solution within the domain only in the last step of the algorithm, we need to eliminate the unknowns $u_I^{(i)}$ with a block factorization:

$$A = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ A_{\Gamma I}^{(1)} A_{II}^{(1)} & A_{\Gamma I}^{(2)} A_{II}^{(2)} & I \end{bmatrix} \cdot \begin{bmatrix} A_{II}^{(1)} & 0 & A_{I\Gamma}^{(1)} \\ 0 & A_{II}^{(2)} & A_{I\Gamma}^{(2)} \\ 0 & 0 & S \end{bmatrix},$$

where I is the identity matrix, and $S = A_{\Gamma\Gamma} - A_{\Gamma I}^{(1)} A_{II}^{(1)-1} A_{II}^{(1)} - A_{\Gamma I}^{(2)} A_{II}^{(2)-1} A_{II}^{(2)}$ is the Schur complement of $A_{\Gamma\Gamma}$. The corresponding linear system is given by:

$$\begin{bmatrix} A_{II}^{(1)} & 0 & A_{II}^{(1)} \\ 0 & A_{II}^{(2)} & A_{II}^{(2)} \\ 0 & 0 & S \end{bmatrix} \begin{bmatrix} u_I^{(1)} \\ u_I^{(2)} \\ u_\Gamma \end{bmatrix} = \begin{bmatrix} f_I^{(1)} \\ f_I^{(2)} \\ g_\Gamma \end{bmatrix},$$

where $g_\Gamma = f_\Gamma - A_{\Gamma I}^{(1)} \cdot A_{II}^{(1)-1} \cdot f_I^{(1)} - A_{\Gamma I}^{(2)} \cdot A_{II}^{(2)-1} \cdot f_I^{(2)}$. This can be further reduced to the Schur complement system:

$$S \cdot u_\Gamma = g_\Gamma. \quad (3.19)$$

If we introduce the following notations for $i = 1, 2$

$$S^i := A_{\Gamma\Gamma}^{(i)} - A_{\Gamma I}^{(i)} A_{II}^{(i)-1} A_{II}^{(i)} \quad \text{and} \quad g_\Gamma^{(i)} := f_\Gamma^{(i)} - A_{\Gamma I}^{(i)} \cdot A_{II}^{(i)-1} \cdot f_I^{(i)},$$

then we have that

$$S = S^{(1)} + S^{(2)} \quad \text{and} \quad g_\Gamma = g_\Gamma^{(1)} + g_\Gamma^{(2)}.$$

We define the discrete version of the Neumann-Neumann iteration method as follows. Starting with an initial guess u_Γ^0 , the iteration proceeds by first solving the Dirichlet problems for each subdomain $i = 1, 2$:

$$(D_i) \quad A_{II}^{(i)} u_I^{(i), n+\frac{1}{2}} + A_{II}^{(i)} u_\Gamma^n = f_I^{(i)}.$$

Next, using the approximate flux residual r_Γ (as given by the third row of equation (3.18)), we solve the Neumann problems for each subdomain:

$$(N_i) \quad \begin{bmatrix} A_{II}^{(i)} & A_{II}^{(i)} \\ A_{\Gamma I}^{(i)} & A_{\Gamma\Gamma}^{(i)} \end{bmatrix} \begin{bmatrix} w_I^{(i), n+1} \\ w_\Gamma^{(i), n+1} \end{bmatrix} = \begin{bmatrix} 0 \\ r_\Gamma \end{bmatrix}.$$

Finally, the interface values are updated according to:

$$u_\Gamma^{n+1} = u_\Gamma^n - \theta \left(w_\Gamma^{(1), n+1} + w_\Gamma^{(2), n+1} \right).$$

By eliminating the interior variables from both the Dirichlet and Neumann problems, we obtain the relation:

$$u_\Gamma^{n+1} - u_\Gamma^n = \theta \left(S^{(1)-1} + S^{(2)-1} \right) (g_\Gamma - S u_\Gamma^n),$$

which shows that the Neumann-Neumann method is a preconditioned Richardson iteration for the system in (3.19), using $S^{(1)-1} + S^{(2)-1}$ as the preconditioner. To improve

convergence, a diagonal scaling based on the vertex degrees on the interface Γ is often employed. This leads to a preconditioner of the form:

$$D_\Gamma \left(S^{(1)^{-1}} + S^{(2)^{-1}} \right) D_\Gamma,$$

where the diagonal entries of D_Γ are given by d_v^{-1} for each vertex $v \in \Gamma$. I emphasize that the Richardson iteration is presented here primarily for historical reasons, and to avoid the complexity of updating u_Γ in the context of a more advanced iteration method. However, in practice, it is generally recommended to use either a preconditioned conjugate gradient method or a minimal residual method. Additionally, the matrices $S^{(i)}$, and particularly their inverses, should typically not be explicitly computed. Instead, we are only concerned with their effect when applied to a vector. Specifically, rather than performing matrix multiplication with $S^{(i)}$, we solve a Dirichlet problem. Similarly, instead of multiplying with $S^{(i)^{-1}}$, we solve a Neumann problem.

Gaussian Random Fields on Domains

This chapter introduces the theoretical basis for generating random fields, a concept used to model spatially correlated data. We start by defining a realization of a random field and show how correlations between points are described by covariance matrices constructed with kernel functions. These kernel functions help model the smoothness and scale of variation in the field. We then discuss some common kernel functions and conclude with methods for efficiently generating random field realizations, including Cholesky decomposition and GPU accelerated methods. This chapter is based on [9].

4.1. Theoretical Background

As stated in Definition 8, a random field is a collection of random variables indexed by the set $\mathcal{D}$. One realization of this collection is generated by randomly selecting a value for each random variable from its domain while also considering the covariance among the random variables. The covariance matrix K can be computed using kernel functions, which map pairs of elements from $\mathcal{D} \times \mathcal{D}$ to real values $\mathbb{R}$:

$$K = \begin{bmatrix} k(x_1, x_1) & k(x_1, x_2) & \cdots & k(x_1, x_n) \\ k(x_2, x_1) & k(x_2, x_2) & \cdots & k(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x_n, x_1) & k(x_n, x_2) & \cdots & k(x_n, x_n) \end{bmatrix},$$

where $\{x_1, x_2, \dots, x_n\} \subseteq \mathcal{D}$; and we assume, that k is a kernel function such that $k(x_i, x_j) = k(x_j, x_i)$ and that K is positive definite.

In the following, I will introduce a few well-known kernel functions, first briefly ex-

plain the notations used. Let $x, y \in \mathcal{D}$, and let $\|x - y\|$ denote the Euclidean distance:

$$\|x - y\| = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}.$$

Furthermore, σ represents the signal variance, and l is the length scale parameter. The length scale l determines how quickly the covariance (or correlation) between points decreases.

1. **Exponential kernel:** The exponential kernel is defined the following way:

$$k(x, y) = \sigma^2 \cdot \exp\left(\frac{-\|x - y\|}{l}\right).$$

In the formula, we observe a negative sign in front of the Euclidean distance within the exponential function, indicating that this kernel returns small values for points that are far apart in the input space. This is a common property of kernel functions.

2. **RBF kernel:** The RBF (radial basis function) kernel is given by:

$$k(x, y) = \sigma^2 \cdot \exp\left(\frac{-\|x - y\|^2}{2 \cdot l^2}\right).$$

3. **Matérn kernel:** The Matérn kernel is defined by:

$$k(x, y) = \frac{2^{1-\nu} \cdot \sigma^2}{\Gamma(\nu)} (\kappa \cdot \|x - y\|)^\nu K_\nu(\kappa \cdot \|x - y\|),$$

where ν is a positive parameter of the covariance, Γ is the gamma function, K_ν is the modified Bessel function of the second kind, and $\kappa = \sqrt{2\nu}/l$. The modified Bessel function of the second kind can be obtained for a complex parameter ν and a real variable x (with $x > 0$) as part of the solution to the following modified Bessel differential equation:

$$x^2 \frac{d^2 y}{dx^2} + x \frac{dy}{dx} - (x^2 + \nu^2)y = 0.$$

For $\nu \in \mathbb{R}$, the general solution to this equation is given by a linear combination of two linearly independent functions:

$$y(x) = C_1 J_\nu(x) + C_2 K_\nu(x),$$

where $I_\nu(x)$ is the modified Bessel function of the first kind, defined by

$$J_\nu(x) = \sum_{k=0}^{\infty} \frac{(-1)^k \left(\frac{x}{2}\right)^{\nu+2k}}{k! \Gamma(1 + \nu + k)!}$$

and $K_\nu(x)$ is the modified Bessel function of the second kind, given by

$$Y_\nu(x) = \frac{J_\nu(x) \cos(\nu\pi) - J_{-\nu}(x)}{\sin(\nu\pi)}.$$

C_1 and C_2 are constants determined by boundary conditions.

As we will see in a later chapter, this kernel function is fundamental, as it is closely connected to Gaussian random fields on graphs.

The kernel functions mentioned above are all *stationary*, meaning they depend only on the distance between points in the domain. Additionally, they are *isotropic*, meaning that they treat all directions in the input space equally.

In the following, I will explain the algorithm of generating normal random fields on domains $\mathcal{D} \subset \mathbb{R}^m, m \in \mathbb{N}$.

1. Define the domain $\mathcal{D} \subset \mathbb{R}^m$. If $|\mathcal{D}| = \infty$, then sample it by selecting a set of points $\{x_1, x_2, \dots, x_n\} \subset \mathcal{D}$ where the random field will be evaluated.
2. Choose an appropriate kernel function $k : \mathcal{D} \times \mathcal{D} \rightarrow \mathbb{R}$ based on the desired properties of the random field (for example smoothness, range). Using this kernel function, compute the covariance matrix K .
3. Motivated by Definition 11, perform the Cholesky decomposition on the covariance matrix K such that $K = LL^\top$, where L is a lower triangular matrix. The Cholesky decomposition is guaranteed to exist, since the covariance function creates a positive definite covariance matrix K .
4. Sample a vector $z = [z_1, z_2, \dots, z_n]^\top$, where z is a standard Gaussian vector.
5. Multiply z by L to obtain correlated samples: $y = Lz$. The vector $y = [y_1, y_2, \dots, y_n]^\top$ represents a realization of the normal random field evaluated at the grid points $\{x_1, x_2, \dots, x_n\}$ on the domain $\mathcal{D}$.

4.2. Implementation in Python

For simplicity, I will explain the process of generating a realization of a normal random field by selecting $\mathcal{D} = ([0, 1] \times [0, 1]) \subset \mathbb{R}^2$. We will sample from this space by creating a meshgrid with a step size $h = 1/n$. Each point in this grid represents a random variable. Assume that this grid contains n^2 points. The steps of the algorithm:

1. *Create the grid:* Compute an equidistant grid consisting of n^2 (x, y) points. In my implementation, I use `numpy.linspace` for this task.

2. *Compute the covariance matrix:* Iterate through the grid (with two for loops) created in the previous step and compute the elements of the covariance matrix using the Matérn kernel function.
3. *Compute the Cholesky decomposition of the covariance matrix:* we want to obtain the L matrix - in Python, we can easily achieve this using the `numpy.linalg.cholesky` function.
4. *Generate a random vector:* Create the random vector z , which contains n^2 random values sampled from the standard normal distribution. I use the `numpy.random` library for this.
5. *Compute $L \cdot z$:* The result of this expression is the Gaussian random field itself in vector form.
6. *Plotting:* Visualize the result using `Matplotlib`.

A serious issue with the algorithm introduced above is that it can become quite slow when the n^2 factor is large. This is primarily due to the Cholesky decomposition's complexity, which is $\mathcal{O}(s^3)$ in general, where s is the size of the square matrix K ; that is, in our case, $\mathcal{O}(n^{12})$. Since this complexity can cause random field generation to be really slow, I implemented the algorithm on GPU to speed up the process. While this method is quite fast, even for larger grids, it requires a significant amount of GPU memory. In my tests, I found that for $n = 100$, the algorithm used over 4 GB of GPU memory. The `PyTorch` library provides the necessary functions to compute Gaussian random fields, so my implementation mainly builds on this library.

4.3. Examples

In the following plots I highlight the effect of the various parameters on random fields generated with the Matérn kernel. In particular:

- The parameter n determines the grid precision (Figures 4.2 and 4.3).
- The parameter σ represents the signal variance; a larger value will result in more drastic changes in the normal random field (Figure 4.5).
- The parameter l denotes the length scale: increasing l will reduce local variations, while decreasing it will increase the amount of local changes (Figure 4.4).
- The parameter ν indicates the smoothness of the field (Figures 4.6 and 4.7).

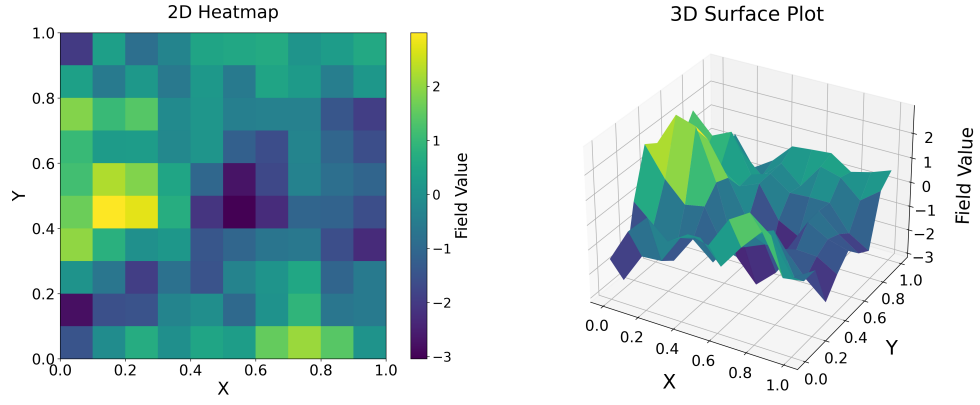


Figure 4.2: Parameters: $n = 10$, $\sigma = 1.0$, $l = 0.1$, $\nu = 5.0$

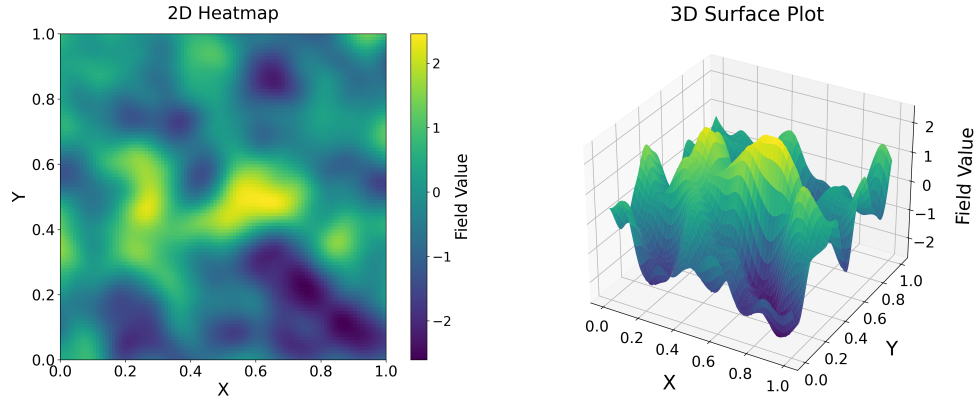


Figure 4.3: Parameters: $n = 100$, $\sigma = 1.0$, $l = 0.1$, $\nu = 5.0$

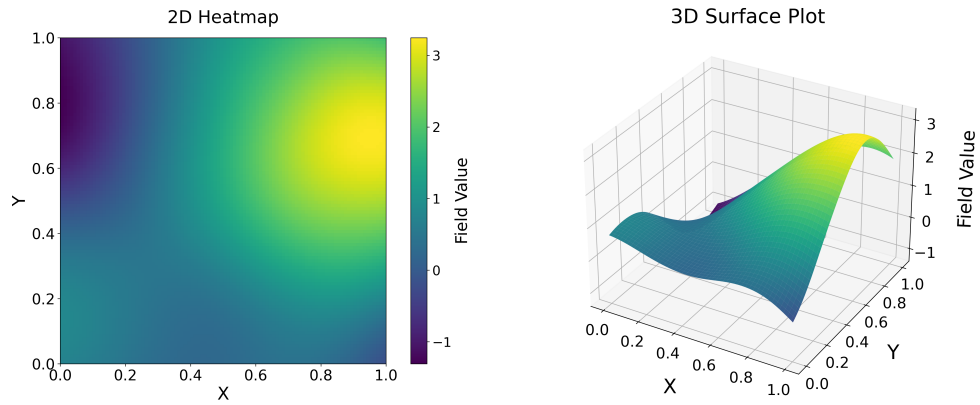


Figure 4.4: Parameters: $n = 100$, $\sigma = 1.0$, $l = 0.5$, $\nu = 5.0$

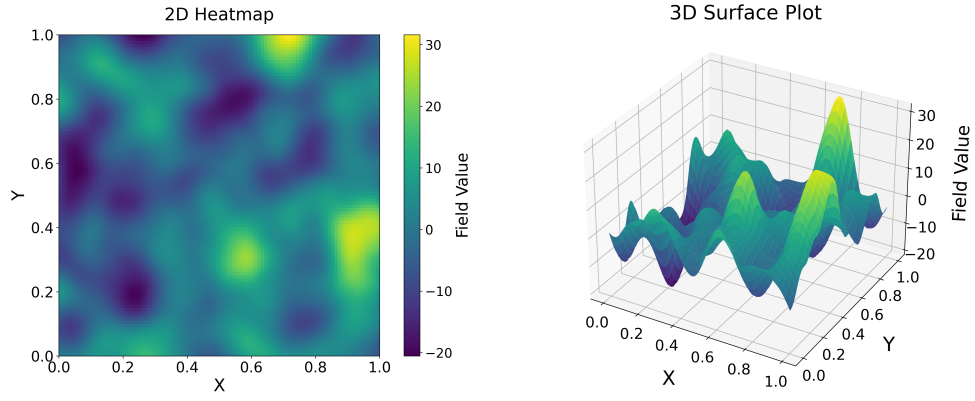


Figure 4.5: Parameters: $n = 100$, $\sigma = 10.0$, $l = 0.1$, $\nu = 5.0$

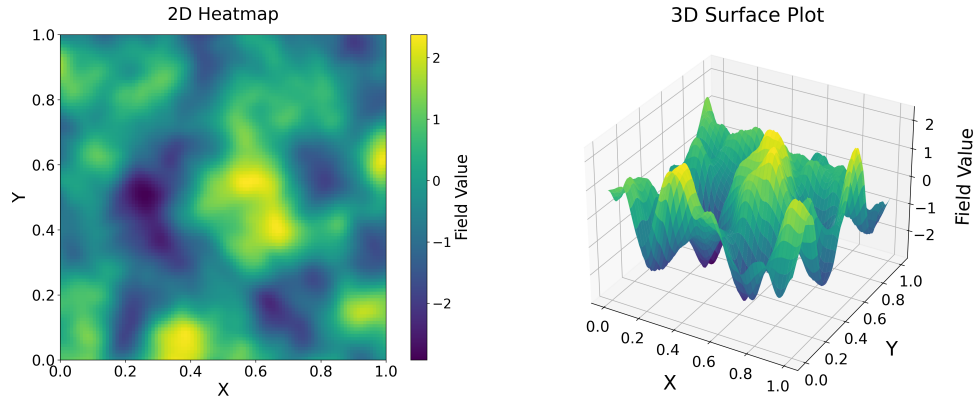


Figure 4.6: Parameters: $n = 100$, $\sigma = 1.0$, $l = 0.1$, $\nu = 3.0$

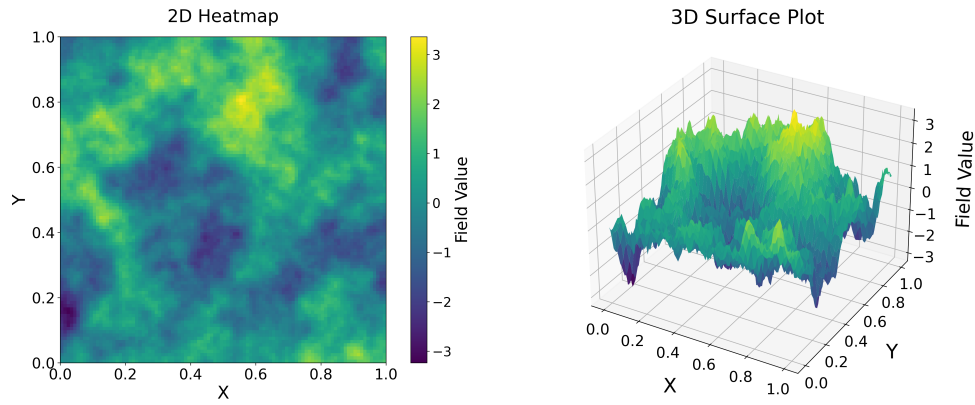


Figure 4.7: Parameters: $n = 100$, $\sigma = 1.0$, $l = 0.1$, $\nu = 1.0$

Fractional Stochastic Partial Differential Equations on Quantum Graphs

In several applications, there is an increasing demand for statistical models defined over network structures, such as connected rivers or road networks. Traditional approaches based on Euclidean distances often perform poorly in these settings, where connectivity follows non-Euclidean pathways. A common workaround is to use the geodesic or shortest-path metric, yet the electrical resistance distance metric has also gained attention for defining covariance structures on networks. In graph theory, the resistance distance between two vertices of a simple, connected graph G is defined as the electrical resistance between the corresponding vertices in an electrical network where each edge of the graph is replaced by a 1-ohm resistor.

Recent studies, for instance, have shown that Gaussian fields can be constructed on certain metric graphs using a Matérn-type covariance function, given by:

$$r(s, t) = \frac{\Gamma(\nu)}{\tau^2 \Gamma(\nu + 1/2) \sqrt{4\pi\kappa^{2\nu}}} (\kappa d(s, t))^\nu K_\nu(\kappa d(s, t)),$$

where $d(s, t)$ denotes the resistance distance; τ and κ are parameters that determine the variance and correlation range, respectively; $0 < \nu \leq \frac{1}{2}$ controls path regularity; K_ν represents the modified Bessel function of the second kind; and Γ is the gamma function. However, due to the constraint $\nu \leq \frac{1}{2}$, this approach cannot produce differentiable Gaussian fields on metric graphs. Furthermore, generating Matérn fields over Euclidean spaces requires computing the Cholesky decomposition of large covariance matrices, which is computationally expensive, and scales poorly with the size of the network.

In this chapter, I introduce a method for generating Gaussian random fields on compact metric graphs by solving a fractional stochastic partial differential equation. This approach avoids the limitations of traditional kernel-based models, which struggle to

ensure positive definite covariance matrices on graphs, making Cholesky decomposition and efficient field generation challenging. This approach also provides an alternative for generating normal random fields on simpler domains. This chapter is based on [13], [15].

5.1. SPDE Approach of Gaussian Random Fields

A Gaussian Matérn field defined on the graph $\mathbf{G}$ with parameters σ , ν , and $\kappa > 0$ can be viewed as the statistically stationary solution u to the following SPDE:

$$(\kappa^2 - \Delta)^\beta u(x) = W(x) \quad x \in \mathbf{G},$$

where Δ denotes the Laplacian with domain $D(\Delta)$ consisting of $H^2(\mathbf{G})$ functions that satisfy the vertex conditions; and the positive parameters σ , ν , and κ allow for the control of the most important characteristics of the random field u : its variance, smoothness, and correlation range. In the context of Gaussian Matérn fields, the *correlation range* represents the distance at which the field values remain significantly correlated. Essentially, it determines how rapidly the correlations decrease as the distance in the spatial domain increases. The parameter β can be calculated as $\beta = (2\nu + 1)/4$, and W denotes Gaussian white noise. By introducing the operator L as $L = \kappa^2 - \Delta$, the above equation can be rewritten as

$$L^\beta u(x) = W(x). \quad (5.20)$$

We can formally solve this equation by multiplying both sides with the inverse of L^β . The $L^{-\beta}$ operator can be given as

$$L^{-\beta} = \frac{\sin(\pi\beta)}{\pi} \int_0^\infty \lambda^{-\beta} (\lambda I + L)^{-1} d\lambda = \frac{2\sin(\pi\beta)}{\pi} \int_{-\infty}^\infty \left(e^{2\beta y} I + e^{2y} L \right)^{-1} dy. \quad (5.21)$$

Similarly to Chapter 2.3, we can solve the above differential equation by applying the finite element method.

Observe that (5.20) can be rewritten in the form of (2.2) if we set $c = 1$ and $p = \kappa^2$. Hence, the L operator is the same as the operator T in (2.2). This implies that, after applying the finite element method to both equations, we obtain stiffness matrices of the same form for each equation.

5.1.1 Integral Quadrature

In this subsection, we will approximate the solution of (5.20) using numerical methods. First, we select a uniform grid given by $\{y_l = l \cdot k : l \in \mathbb{Z}, -K^- \leq l \leq K^+\}$ where $k > 0$, and K^- and K^+ are positive integers. In the discrete solution the operator L

is approximated with the L_h stiffness matrix. Then we approximate the integral (5.21) with the quadrature

$$Q_{h,k}^\beta := \frac{2k \sin(\pi\beta)}{\pi} \sum_{l=-K^-}^{K^+} e^{2\beta y_l} (I + e^{2y_l} L_h)^{-1},$$

where

- L_h is the stiffness matrix defined the same way as (3.13).
- An appropriate choice for the limits K^- and K^+ are

$$K^- := \left\lceil \frac{\pi^2}{4\beta k^2} \right\rceil, \quad K^+ := \left\lceil \frac{\pi^2}{4(1-\beta)k^2} \right\rceil.$$

- I is the identity matrix.

Since this quadrature converges exponentially, a good choice for k is $k \leq -\pi^2/(\beta \ln(h))$ [15]. After computing $Q_{h,k}^\beta$, the solution $u_{h,k}^Q$ of the stochastic partial differential equation can be approximated by calculating the following:

$$u_{h,k}^Q = Q_{h,k}^\beta \cdot W_h,$$

where $W_h = R \cdot \xi$. Here ξ is a random vector sampled from the standard normal distribution, and R is the Cholesky decomposition of the so-called mass matrix M . The structure and nonzero pattern of the mass matrix is the same as the structure of the matrix A defined in (3.13):

$$M = \begin{bmatrix} M_E & M_{EV} \\ M_{VE} & M_V \end{bmatrix}$$

Consequently, the index function for the non-zero elements is exactly the same in the case of the M and A matrices. However, the calculation of the elements of the matrices is a bit different. Instead of integrating the $\psi'_i \cdot \psi'_j$ product shown in section 3.1, we need to integrate the $\psi_i \cdot \psi_j$ product. The calculations for one subinterval:

$$\begin{cases} m_{ii} = \int_{x_i}^{x_{i+1}} \psi_i^2 dx \\ m_{i(i+1)} = m_{(i+1)i} = \int_{x_i}^{x_{i+1}} \psi_i \cdot \psi_{i+1} dx \end{cases}$$

Computing the integrals using Simpson's quadrature rule (which in this case yields the exact value of the integral with no error):

$$\begin{cases} m_{ii} = \frac{h}{6} \psi_i^2(x_i) + \frac{2h}{3} \psi_i^2(x_{\text{mid}}) + \frac{h}{6} \psi_i^2(x_{i+1}) \\ m_{i(i+1)} = m_{(i+1)i} = \frac{h}{6} \psi_i(x_i) \psi_{i+1}(x_i) + \frac{2h}{3} \psi_i(x_{\text{mid}}) \psi_{i+1}(x_{\text{mid}}) + \frac{h}{6} \psi_i(x_{i+1}) \psi_{i+1}(x_{i+1}), \end{cases}$$

where $x_{\text{mid}} = (x_{i+1} + x_i)/2$. Substituting into the above expressions:

$$\begin{cases} m_{ii} = \frac{h}{6} \cdot 1 + \frac{2h}{3} \cdot \frac{1}{4} + \frac{h}{6} \cdot 0 = \frac{h}{3} \\ m_{i(i+1)} = m_{(i+1)i} = \frac{h}{6} \cdot 1 \cdot 0 + \frac{2h}{3} \cdot \frac{1}{2} \cdot \frac{1}{2} + \frac{h}{6} \cdot 0 \cdot 1 = \frac{h}{6}. \end{cases}$$

In conclusion, the solution can be expressed as:

$$u_{h,k}^Q = Q_{h,k}^\beta \cdot W_h = \left(\frac{2k \sin(\pi\beta)}{\pi} \sum_{l=-K^-}^{K^+} e^{2\beta y_l} (I + e^{2y_l} L_h)^{-1} \right) \cdot (R \cdot \xi). \quad (5.22)$$

5.1.2 Implementation

In this section, I will detail the implementation of the mathematical method previously discussed in this chapter using C++. Essentially, I had to implement the calculation of the formula (5.22).

Since computing the expression $(I + e^{2y_l} L_h)^{-1}$ is computationally expensive, and we have to do it $(K^+ + K^- + 1)$ times, instead, we calculate the effect of the inverses on the white noise individually and then sum the results. This means, that we solve the equivalent problem

$$\begin{cases} \left(\frac{\pi}{2k \sin(\pi\beta)} \cdot e^{-2\beta y_l} \cdot (I + e^{2y_l} L_h) \right) u_{h,k}^Q(l) = R \cdot \xi, & l = -K^-, (-K^- + 1), \dots, K^+ \\ u_{h,k}^Q = \sum_{l=-K^-}^{K^+} u_{h,k}^Q(l) \end{cases}$$

instead of (5.22). This way, we do not need to invert matrices, we just have to solve the linear system

$$\left(\frac{\pi}{2k \sin(\pi\beta)} \cdot e^{-2\beta y_l} \cdot (I + e^{2y_l} L_h) \right) u_{h,k}^Q(l) = R \cdot \xi, \quad l = -K^-, (-K^- + 1), \dots, K^+$$

$(K^+ + K^- + 1)$ times, and sum up the $u_{h,k}^Q(l)$ results, which is much cheaper computationally. With the theoretical background established, I will proceed to break down the implementation steps in detail to demonstrate how the program efficiently computes $u_{h,k}^Q$:

1. Create an `Eigen::VectorXd` vector filled with zeros. This vector, denoted as `s`, will represent the $u_{h,k}^Q$ vector in the program.
2. Compute $R \cdot \xi$ - this computation is performed only once to generate a random field, as subsequent steps of the algorithm will use the same result:

- (a) The matrix R is derived from the Cholesky decomposition of the mass matrix M . The mass matrix is constructed using the column-wise insertion method discussed in Section 3.3. For the Cholesky decomposition, I used the `Eigen::SimplicialLLT` solver.
 - (b) The vector ξ , representing Gaussian white noise, is generated as follows: the process starts with initializing a random number generator based on the Mersenne Twister algorithm, implemented in C++ as `std::mt19937`. This generator is seeded with a value from `std::random_device`. A normal distribution with a mean of 0 and a standard deviation of 1 is employed to create the white noise, utilizing `std::normal_distribution<>()`.
3. Compute the `Eigen::VectorXd` vector `s_iterate` representing $u_{h,k}^Q(l)$ by solving the corresponding equation system. To obtain this linear system, we need some further calculations:
- (a) First, compute the coefficient $\frac{\pi}{2k \sin(\pi\beta)} \cdot e^{-2\beta y_l}$. Note that the factor $\frac{\pi}{2k \sin(\pi\beta)}$ only needs to be computed once, as it does not depend on l .
 - (b) In the expression $(I + e^{2y_l} L_h)$, compute the matrix L_h using the column-wise insertion method. For the entire expression, multiply the L_h matrix by e^{2y_l} and add the result to the identity matrix I to obtain the final matrix.
4. Add the vector `s_iterate` to the main solution vector `s`, accumulating the results.
5. If there are remaining $u_{h,k}^Q(l)$ values to compute, return to step 2 to repeat the process until all values have been calculated.

As discussed in Section 3.4, there are situations where we need to solve differential equations on large graphs that exceed the computer's memory capacity. To generate normal random fields on these larger graphs, we must decompose the graphs into smaller sections and solve the problem iteratively. To achieve this, I adapted a Neumann-Neumann domain decomposition implementation in C++ for generating normal random fields. This implementation decomposes the graph into its edges, so essentially the subgraphs are the edges themselves. After computing the $R \cdot \xi$ vector, we need to distribute its values among the edges, and we need to use the decomposed version of the L_h matrix instead of the original one. Unfortunately, since we need the entire mass matrix M to calculate the $R \cdot \xi$ product, we encounter some memory efficiency issues. Addressing this limitation will be part of our future work, where we aim to develop an approach to determine the values of the $R \cdot \xi$ vector without explicitly computing the entire mass matrix.

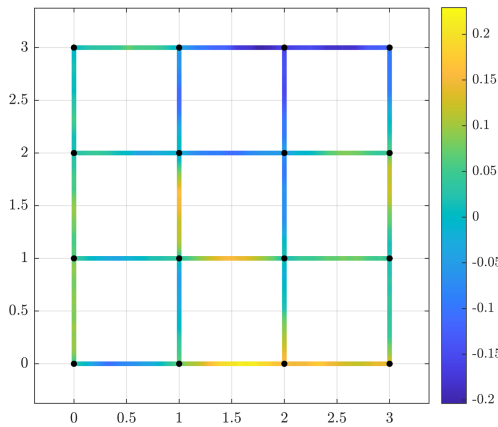
Examples

In this chapter, I present several results obtained using the implementation described in Chapter 5. The adjacency matrices of the input graphs were generated with the Python **NetworkX** library. These matrices were then processed using the C++ implementation described in this paper, with results visualized in MATLAB for analysis.

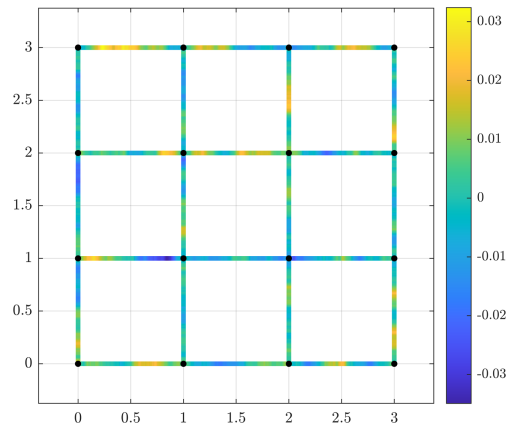
When generating normal random fields on graphs, three key parameters directly influence the properties of the normal random field:

1. The parameter N determines the number of subintervals on each edge of the graph, acting as a discretization parameter.
2. The parameter β serves as a smoothness parameter. It is related to the parameter ν of the Matérn kernel and can be calculated as $\beta = \frac{2\nu + 1}{4}$. The range of β is restricted to the interval $\beta \in \left(\frac{1}{4}, 1\right)$.
3. The parameter κ controls the balance between the second derivative of the Gaussian random field and the field itself. Higher values of κ lead to noisier outputs, while lower values produce smoother results.

4×4 Square Graph

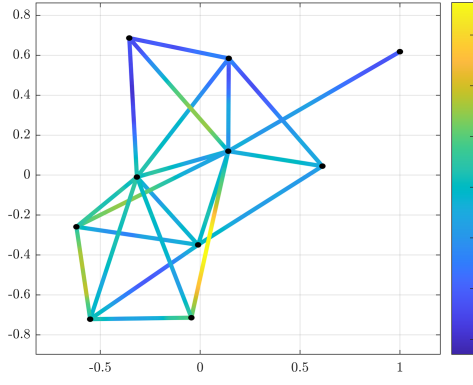


$$N_1 = 1000, \beta_1 = 0.8, \kappa_1 = 1.0$$

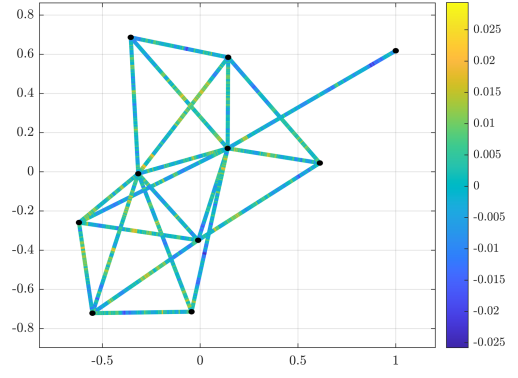


$$N_2 = 1000, \beta_2 = 0.8, \kappa_2 = 10.0$$

Barabasi Albert (Scale Free) Graph

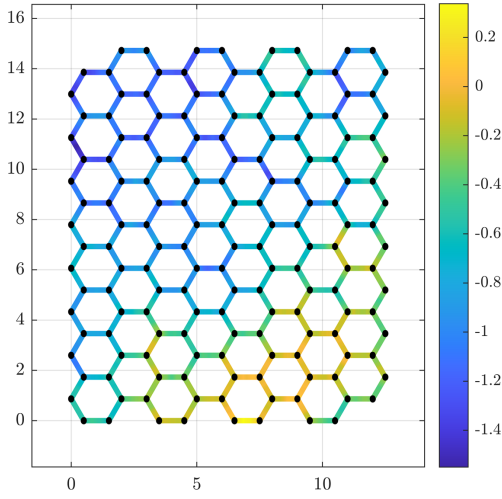


$$N_1 = 1000, \beta_1 = 0.8, \kappa_1 = 1.0$$

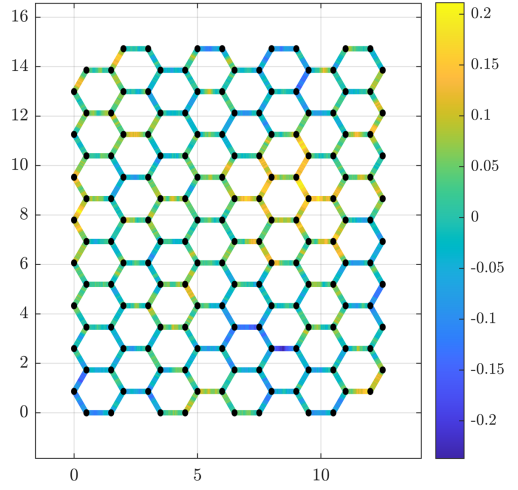


$$N_2 = 1000, \beta_2 = 0.3, \kappa_2 = 10.0$$

Hexagonal Lattice Graph

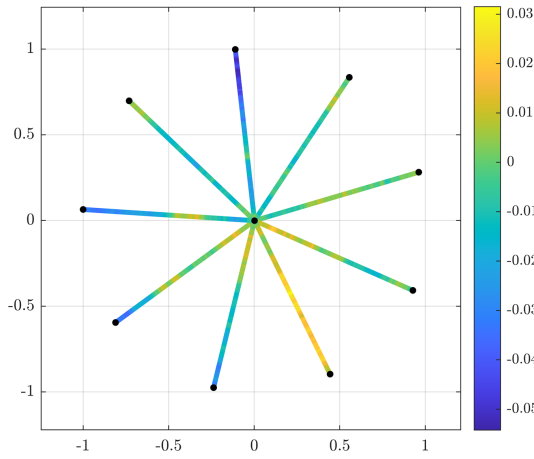


$$N_1 = 100, \beta_1 = 0.8, \kappa_1 = 0.1$$

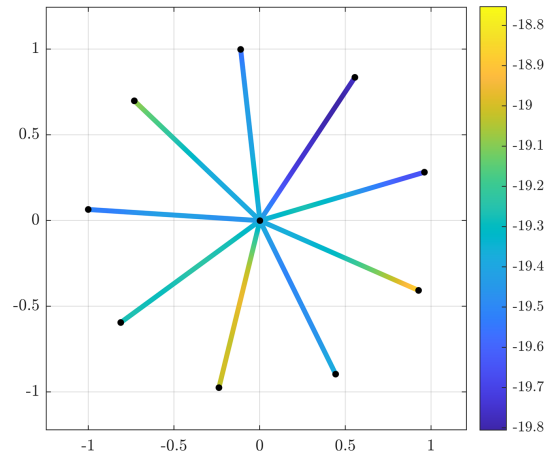


$$N_2 = 100, \beta_2 = 0.5, \kappa_2 = 0.1$$

Star Graph



$$N_1 = 1000, \beta_1 = 0.5, \kappa_1 = 0.1$$



$$N_2 = 1000, \beta_2 = 0.99, \kappa_2 = 0.1$$

Conclusions

In this work the efficient simulation of Gaussian random fields on networks was investigated using a fractional PDE with white noise forcing. The integral representation of the fractional operator was discretized with an integral quadrature, which was solved with FEM. To overcome the memory limitations of FEM, I used a Neumann-Neumann-type domain decomposition method. Furthermore, I measured and optimized the creation of the underlying sparse matrices by implementing a column-wise ordering. The only limitation of the approach is that the white noise forcing need to be computed for the whole network. In the future I wish to study white noise synthesis to overcome this restriction.

Acknowledgements

I would like to thank my supervisors, Prof. Mihály Kovács and Mihály András Vághy, for their support, guidance, and encouragement throughout this work. The weekly consultations and helpful advice made this research project both productive and enjoyable. I am very grateful for their knowledge and commitment, which made this work possible.

I would also like to express my gratitude to my parents and family for their constant support and encouragement throughout this research project.

Bibliography

- [1] J. M. V. Hoef, E. Peterson, and D. Theobald, “Spatial statistical models that use flow and stream distance,” *Environmental and Ecological Statistics*, vol. 13, no. 4, pp. 449–464, 2006.
- [2] S. Bonaccorsi, C. Marinelli, and G. Ziglio, “Stochastic FitzHugh-Nagumo equations on networks with impulsive noise,” *Electronic Journal of Probability*, vol. 13, 2008.
- [3] B. Al Beattie, P. Feketa, K. Ochs, and H. Kohlstedt, “Criticality in FitzHugh-Nagumo oscillator ensembles: Design, robustness, and spatial invariance,” *Communications Physics*, vol. 7, no. 1, 2024.
- [4] G. Berkolaiko and M. Ettehad, “Three-dimensional elastic beam frames: Rigid joint conditions in variational and differential formulation,” *Studies in Applied Mathematics*, vol. 148, no. 4, pp. 1586–1623, 2022.
- [5] M. Dolbeault, O. Mula, and A. Somacal, *State estimation of urban air pollution with statistical, physical, and super-learning graph models*, arXiv:2402.02812, 2024.
- [6] J. M. K. Aheto and G. A. Dagne, “Geostatistical analysis, web-based mapping, and environmental determinants of under-5 stunting: evidence from the 2014 Ghana Demographic and Health Survey,” *The Lancet Planetary Health*, vol. 5, no. 6, e347–e355, 2021.
- [7] H. Zhang, J. Guilleminot, and L. J. Gomez, “Stochastic modeling of geometrical uncertainties on complex domains, with application to additive manufacturing and brain interface geometries,” *Computer Methods in Applied Mechanics and Engineering*, vol. 385, p. 114 014, 2021.
- [8] L. Maynou, M. Monfort, B. Morley, and J. Ordóñez, “Club convergence in european housing prices: The role of macroeconomic and housing market fundamentals,” *Economic Modelling*, vol. 103, p. 105 595, 2021.
- [9] M. L. Stein, *Interpolation of Spatial Data*. Springer New York, 1999.

- [10] F. Lindgren, H. Rue, and J. Lindström, “An Explicit Link between Gaussian Fields and Gaussian Markov Random Fields: The Stochastic Partial Differential Equation Approach,” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 73, no. 4, pp. 423–498, 2011.
- [11] F. Lindgren, D. Bolin, and H. Rue, “The SPDE approach for Gaussian and non-Gaussian fields: 10 years and still running,” *Spatial Statistics*, vol. 50, p. 100599, 2022.
- [12] J. Guinness and M. Fuentes, “Isotropic covariance functions on spheres: Some properties and modeling considerations,” *Journal of Multivariate Analysis*, vol. 143, pp. 143–152, 2016.
- [13] D. Bolin, A. B. Simas, and J. Wallin, “Gaussian Whittle–Matérn fields on metric graphs,” *Bernoulli*, vol. 30, no. 2, 2024.
- [14] D. Bolin, K. Kirchner, and M. Kovács, “Numerical Solution of Fractional Elliptic Stochastic PDEs with Spatial White Noise,” *IMA Journal of Numerical Analysis*, vol. 40, no. 2, pp. 1051–1073, Dec. 2018.
- [15] D. Bolin, M. Kovács, V. Kumar, and A. Simas, “Regularity and numerical approximation of fractional elliptic differential equations on compact metric graphs,” *Mathematics of Computation*, vol. 93, no. 349, pp. 2439–2472, 2023.
- [16] M. Arioli and M. Benzi, “A finite element method for quantum graphs,” *IMA Journal of Numerical Analysis*, vol. 38, no. 3, pp. 1119–1163, 2018, Advance Access publication on June 21, 2017.
- [17] M. Kovács and M. A. Vághy, “Neumann-Neumann Type Domain Decomposition of Elliptic Problems on Metric Graphs,” *arXiv:2402.05707*, 2024.
- [18] F. C. Klebaner, *Introduction to Stochastic Calculus with Applications*. London, UK: Imperial College Press, 2005.
- [19] F. D. and C. Kraaikamp H.P. Lopuhaa and L. Meester, *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Springer, 2005.
- [20] H. Wendland, *Numerical Linear Algebra: An Introduction*. Cambridge, UK: Cambridge University Press, 2017.
- [21] S. Brenner, *The Mathematical Theory of Finite Element Methods*. New York: Springer, 2002.

- [22] S. Larsson and V. Thomée, *Partial Differential Equations with Numerical Methods*.
Berlin: Springer, 2008.