Simulating Level-set Percolation on the Discrete Gaussian Free Field with Arbitrary Conductances

Using a Conjugate Gradient Sampler

Bachelor's Thesis

by

Pim Keer

Academic Supervisors:

dr. A. Cipriani,	Technische Universiteit Delft,
dr. J.M. Thijssen,	Technische Universiteit Delft,
dr. A. Chiarini,	Technische Universiteit Eindhoven.

Examination Committee:

dr. A. Cipriani,	Technische Universiteit Delft,
dr. J.M. Thijssen,	Technische Universiteit Delft,
dr. ir. W.G.M. Groenevelt,	Technische Universiteit Delft,
dr. A. Akhmerov,	Technische Universiteit Delft.

ABSTRACT

Level-set percolation on the Discrete Gaussian Free Field (DGFF) turned out to be a hot topic within mathematical physics over the last couple of years. In particular, the DGFF on $\mathbb{Z}^d$, with homogeneously weighted nearest-neighbour interactions, i.e. all conductances equal to 1, has been studied in detail. These models can be simulated with great efficiency. In this research, we abandon the homogeneity requirement and look at three-dimensional DGFFs with arbitrary conductances. Our goal is to find a quick and reliable method to simulate such DGFFs on a finite lattice. Since this is, in essence, a high-dimensional Gaussian sampling problem, we investigated this problem using the Conjugate Gradients (CG) linear solver as a Gaussian sampler. To see how it performed, we compared our implementation of the CG sampler with known methods for DGFFs in the unit conductance case. Finally, as a showcase of our implementation, we studied level-set percolation on a DGFF with a simple checkerboard conductance pattern.

Our main conclusion is that the CG algorithm is very suitable for simulating Discrete Gaussian Free Fields. Since it does not make any assumptions on the conductances, it can be used to generate DGFFs with arbitrary conductances. However, there are still a number of issues with our implementation. The biggest one is concerning the stopping tolerance of the CG sampler. Once the tolerance is set smaller than some lattice size-dependent threshold, the percolative behaviour of the resulting sample changes drastically. We have not been able to explain this. Moreover, we would recommend making the implementation usable for parallel computing. We have been limited to relatively small lattice sizes during this project. Consequently, the use of certain finite-size scaling arguments when analysing level-set percolation might not always have been as justified.

Finally, based on our study of the DGFF on a lattice with checkerboard conductances a and b ($a < b$), we conjectured that, in its percolative behaviour, this DGFF resembles a DGFF defined on a lattice with constant conductance c, where c is a weighted average of a and b. The weight of a is expected to be larger than the weight of b.

Table of Contents

1 INTRODUCTION

The Discrete Gaussian Free Field (DGFF) on the grid $\mathbb{Z}^d$ is a popular random field model within mathematical physics, with applications in, amongst others, magnetism and quantum field theory. Intuitively, the model assigns to each grid site $i \in \mathbb{Z}^d$ a random value, called the spin. This spin value can be in principle any real number, but the model favours a value that is close to the spin value of its neighbours. In its most basic form, the DGFF weighs all neighbouring spins equally in its computation of the spin at i. We say that the conductance is the same for all pairs of neighbouring sites. This "standard" DGFF is also called the harmonic crystal. For $d \geq 3$, it serves as a model for microscopic fluctuations in a homogeneous crystal lattice at non-zero temperatures [1]. An evident generalisation of the DGFF is to leave out the assumption that all conductances are equal. Some sites might have a very high conductance between them, implying that their spins should be almost equal, while other sites might be linked with a very low conductance, such that their spins depend very little on each other. From a physical point of view, arbitrary conductances represent impurities in an otherwise homogeneous crystal.

Often, one is interested in the percolative behaviour of a spin model. For the DGFF, this behaviour is studied by considering level-set percolation. Sites with a spin greater than a certain threshold are defined as occupied, the others as unoccupied. It is known that there exists a finite, nontrivial percolation threshold in the case of constant conductances [2]. When we generalise to arbitrary conductances, many interesting questions come up. To name a few:

- Given a set of arbitrary conductances, is there still a finite, nontrivial percolation threshold?
- If we have a periodically occuring impurity in an otherwise homogeneous lattice, how large should the impurity be in order to have any influence on the percolative behaviour of the lattice?

We will most certainly not give an answer to any of these questions. However, we might be of help in the process of finding an answer. An important part of studying Discrete Gaussian Free Fields – especially studying level-set percolation on DGFFs – is computer simulation. There are already a number of methods to efficiently generate a DGFF [3][4]. However, they only work for the DGFF with constant conductances. These methods exploit the fact that it can be shown that the DGFF is a Gaussian field, fully described by a mean and covariance matrix [5]. Due to the symmetry in constant conductance DGFFs, it is possible to find an eigendecomposition of the covariance matrix which makes simulation a very efficient task. For arbitrary conductances, such eigendecomposition are no longer possible, so the methods in [3] and [4] fail. It is therefore the goal of this research to find a method that is able to simulate a DGFF, with any given set of conductances. We will consider the three-dimensional case in particular. In our search for a suitable simulation technique, we encountered several methods; from traditional factorisation methods such as the Cholesky decomposition, to Monte Carlo methods like the Gibbs sampler [6]. However, one method in particular quickly caught our attention. This project investigated a Conjugate Gradient-based Gaussian sampler. The Conjugate Gradient (CG) method is predominantly known as a linear solver, first introduced by Hestenes and Stiefel [7] in 1952. However, Parker and Fox showed that it can be turned into a Gaussian sampler with the addition of just one extra vector operation per iteration [8]. We looked at the inner workings of CG, implemented it in Python and then applied it to the Gaussian sampling problem of simulating a DGFF. Also, we tested the method's performance in the constant conductance case by comparing the percolative behaviour from the samples it creates to samples created by the methods described in [3] and [4]. Finally, we used the CG sampler on a nonhomogeneous set of conductances, called the checkerboard pattern.

Let us finish this introduction by giving an idea of what the reader may expect from this dissertation. Chapter 2 gives an introduction to the Gaussian Free Field with unit conductances. In particular, we first construct the model by defining its Hamiltonian and the corresponding Gibbs distribution. Then we show that this definition is equivalent to that of a Gaussian field with the inverse of the graph Laplacian as covariance matrix. After this introduction, we cover the notion of level-set percolation and mention some important results from general percolation theory. At the end of Chapter 2, we introduce the arbitrary conductance DGFF, and also define the checkerboard pattern. Chapter 3 is all about simulating Discrete Gaussian Free Fields. To start off, we cover the methods from [3] and [4] in detail. Thereafter, we present the CG linear solver and the extra step needed to turn it into a Gaussian sampler. Once we know how to sample a DGFF, we talk about the methods needed to analyse its percolative behaviour. In particular, we look at the Hoshen-Kopelman cluster-finding algorithm and a procedure described in [9] to determine the percolation threshold. In Chapter 4, the actual simulations we have run are stated, and their results are given. We immediately discuss a number of these results, and continue this discussion in Chapter 5. In the latter, we try to list all the issues we have encountered with the code, and make a number of recommendations for further research.

2 PREREQUISITES

In this Chapter, the reader can find all the necessary background theory to fully understand this thesis. First of all, the Gaussian Free Field, in particular the discrete form, is introduced. Secondly, the notion of percolation on this Gaussian Free Field is discussed. Along with that, we mention a few important results from percolation theory. We conclude by formulating a Gaussian Free Field with arbitrary conductances. Eventually, this will be the object we would like to simulate.

2.1. THE GAUSSIAN FREE FIELD

THROUGHOUT this dissertation, everything we do will be in the setting of either the standard or a modified version of the Discrete Gaussian Free Field on $\mathbb{Z}^d$. Therefore, it is a good idea to first get acquainted with the standard (Discrete) Gaussian Free Field, abbreviated by (D)GFF. The GFF is a well-known model in statistical physics. In the vaguest sense possible, it models a lattice of particles where each particle carries an abstract value called the spin of the particle. This value is dependent on the spins of neighbouring particles. This explanation deserves a formalisation.

2.1.1. CONSTRUCTION

To construct the GFF, we mainly follow the reasoning of Friedli and Velenik [5] and Borga [10]. First, consider the d-dimensional lattice $\mathbb{Z}^d$. At each lattice point $i = (i_1, i_2, \dots, i_d)^T \in \mathbb{Z}^d$ (i.e. particle), we define a variable $\omega_i \in \mathbb{R}$, which is called the spin at i. We say that the single-spin space Ω_0 of this model is $\mathbb{R}$, as ω_i can be any real number. Combining all these single spins yields a so-called spin configuration of the model, $\omega := (\omega_i)_{i\in\mathbb{Z}^d}$. The set of all possible spin configurations on $\mathbb{Z}^d$ is denoted by $\Omega := \left\{(\omega_i)_{i\in\mathbb{Z}^d} | \forall i \in \mathbb{Z}^d : \omega_i \in \Omega_0\right\}$. This is also written as $\Omega = \Omega_0^{\mathbb{Z}^d}$, so in the case of the GFF, $\Omega = \mathbb{R}^{\mathbb{Z}^d}$. Similarly, we can define a configuration on a subset $S \subset \mathbb{Z}^d$. The set of all spin configurations then becomes: $\Omega_S = \Omega_0^S = \{(\omega_i)_{i\in S} | \forall i \in S : \omega_i \in \Omega_0\}$.

The spin model will have a certain configuration. However, not every configuration is equally likely to occur. To find out with which probability we will observe a specific configuration, we use the notion of energy for that configuration. Note that we only define the energy for a configuration in a finite subset $\Lambda \subset \mathbb{Z}^d$. As we will see in a bit, the energy is fully determined by the Hamiltonian $\mathscr{H}_\Lambda : \Omega \to \mathbb{R}$.

Different spin models are distinguished from each other by their single-spin space Ω_0 and their Hamiltonian $\mathscr{H}_\Lambda$. For the Gaussian Free Field, we already mentioned that $\Omega_0 = \mathbb{R}$. It remains to define the Hamiltonian for the GFF, which is done in Definition 2.1.3. Before we do that however, we introduce the notions of nearest-neighbours and edge-sets in Definitions 2.1.1 and 2.1.2.

Definition 2.1.1 (Nearest-neighbours)**.** We say that two lattice points/lattice sites $i, j \in \mathbb{Z}^d$ are nearest-neighbours, denoted by $i \sim j$, whenever $||i - j||_1 = 1$. Here the 1-norm of a lattice point $i = (i_1, i_2, \dots, i_d)^T \in \mathbb{Z}^d$ is defined as $||i||_1 := \sum_{n=1}^d |i_n|$. Thus, two lattice points are nearest-neighbours whenever they are "just next to each other".

Definition 2.1.2 (Edge-set)**.** Given $\Lambda \subset \mathbb{Z}^d$, we define the edge-set of Λ, denoted by $\mathscr{E}_\Lambda$, as the set containing all nearest-neighbour pairs within Λ. That is, $\mathscr{E}_\Lambda = \left\{(i, j) : ||i - j||_1 = 1; i, j \in \Lambda\right\}$. These nearest-neighbour pairs are also called edges in the context of graphs; they link two nearest-neighbour vertices.

Definition 2.1.3 (Hamiltonian of the GFF)**.** For a finite subset $\Lambda \subset \mathbb{Z}^d$, the Hamiltonian of the Gaussian Free Field in Λ, for any $\omega \in \Omega$, is given by:

$$\mathcal{H}_\Lambda(\omega) := \frac{\beta}{4d} \sum_{(i,j)\in\mathcal{E}_\Lambda} (\omega_i - \omega_j)^2 + \frac{m^2}{2} \sum_{i\in\Lambda} \omega_i^2, \tag{2.1}$$

where $\beta \geq 0$ can be recognised from statistical physics as the inverse temperature $\beta = \frac{1}{k_B T}$ and $m \geq 0$ is the mass. We speak of a massive GFF if $m > 0$, and of a massless GFF if $m = 0$.

In this dissertation, we only consider the massless variant of the Gaussian Free Field, but mention the massive case here for completeness. Thus for the remainder of this text, $m = 0$.

With the Hamiltonian now explicitly formulated, we can find out with which probability a specific configuration may occur. The first step in doing that, is realising that the Hamiltonian is defined only on finite subsets $\Lambda \subset \mathbb{Z}^d$. Therefore, we only allow configurations to differ from each other inside this Λ. Outside of this region, every configuration is fixed and equal to the others. We can use a boundary condition $\eta \in \Omega$ to fix all configurations outside of Λ. That is, every configuration $\omega \in \Omega$ must be equal to η in Λ^c. Notice that we call the whole of $\eta \in \Omega$ the boundary condition. However, as the Hamiltonian (2.1) only includes nearest-neighbour interactions, η can only affect what happens inside Λ through the outer boundary of Λ. The latter is denoted by $\partial_o\Lambda$ and defined as $\partial_o\Lambda := \{i \in \mathbb{Z}^d : i \notin \Lambda \wedge i \sim j, \text{for some } j \in \Lambda\}$.

With that said, we can now define a probability measure $\mu_\Lambda^\eta(A)$ on $(\Omega, \mathcal{F})$, given the region $\Lambda \subset \mathbb{Z}^d$ and a boundary condition $\eta \in \Omega$. Here A is some arbitrary event in the event space $\mathcal{F}$. We will elaborate on the structure of this event space after we define the measure in equation (2.2).

$$\mu_\Lambda^\eta(A) = \int \frac{e^{-\mathcal{H}_\Lambda(\omega_\Lambda\eta_{\Lambda^c})}}{\mathcal{Z}_\Lambda^\eta} \mathbb{1}_A(\omega_\Lambda\eta_{\Lambda^c}) \prod_{i\in\Lambda} \mathrm{d}\omega_i, \tag{2.2}$$

where $d\omega_i$ is the Lebesgue measure corresponding to location $i \in \mathbb{Z}^d$ and $\mathcal{H}_\Lambda(\cdot)$ the Hamiltonian defined earlier. The notation ω_Λ represents the restriction of ω to Λ, so $\omega_\Lambda := (\omega_i)_{i\in\Lambda}$. With this in mind, $\omega_\Lambda\eta_{\Lambda^c}$ stands for the spin configuration in Ω that coincides with ω in the region Λ, and with the boundary condition η in the region Λ^c. Important to keep in mind is that η is fixed, so everything that is random happens only inside of Λ. The reader familiar with statistical mechanics may recognise a Gibbs (or Boltzmann) distribution here. The quantity $\mathcal{Z}_\Lambda^\eta$ is the so-called partition function, defined by equation (2.3). One can see $\mathcal{Z}_\Lambda^\eta$ as the normalising constant for the probability distribution, making sure that $\mu_\Lambda^\eta(\Omega) = 1$.

$$\mathcal{Z}_\Lambda^\eta := \int e^{-\mathcal{H}_\Lambda(\omega_\Lambda\eta_{\Lambda^c})} \prod_{i\in\Lambda} \mathrm{d}\omega_i. \tag{2.3}$$

We may set $\beta = 1$. Indeed, the change of variables $\omega_i' = \sqrt{\beta}\omega_i$ for all $i \in \Lambda$ implies $\mathcal{Z}_\Lambda^\eta = \beta^{\frac{|\Lambda|}{2}} \mathcal{Z}_\Lambda^{\eta'}$, with $\eta' := \sqrt{\beta}\eta$. In a similar fashion, $\mu_\Lambda^\eta(A) = \mu_\Lambda^{\eta'}\big(\sqrt{\beta}A\big)$ for all $A \in \mathscr{F}$. Setting $\beta = 1$ can therefore be done without any problem; it only changes the distribution by a scaling factor.

As promised, let us now take a closer, measure-theoretic look at $\mathscr{F}$. This event space is the natural choice for the collection of events on Ω, but to find out why, we first need to introduce the notion of a cylinder. Consider a configuration $\omega \in \Omega$ and our finite region $\Lambda \subset \mathbb{Z}^d$ from earlier. The restriction ω_Λ can be linked to the original configuration via a projection map $\Pi_\Lambda : \Omega \to \Omega_\Lambda$ such that $\Pi_\Lambda(\omega) = \omega_\Lambda$. With this mapping, given A in the product σ-algebra $\mathscr{B}_\Lambda := \bigotimes_{i\in\Lambda} \mathscr{B}_0$, we can easily write the event that A happens in Λ as $\Pi_\Lambda^{-1}(A) = \{\omega \in \Omega : \omega_\Lambda \in A\}$. Observe that $\mathscr{B}_\Lambda$ is nothing else then the smallest σ-algebra generated by the sets of the form $\times_{i\in\Lambda} A_i$, where $A_i \in \mathscr{B}_0$. Keep in mind that $\mathscr{B}_0$ is the Borel σ-algebra on Ω_0. For the GFF, $\Omega_0 = \mathbb{R}$ and so $\mathscr{B}_0 = \mathscr{B}(\mathbb{R})$.

We can now define the collection of events $\mathscr{C}(\Lambda) := \left\{\Pi_\Lambda^{-1}(A) : A \in \mathscr{B}_\Lambda\right\}$. This is the collection of all events that only depend on what is happening inside of Λ, i.e. only on the spins ω_i with $i \in \Lambda$. Any event in $\mathscr{C}(\Lambda)$ is called a cylinder (with base Λ). Consequently, the collection of all events that depend on a finite amount of ω_i's is given by $\mathscr{C} := \bigcup_{\Lambda \subset \mathbb{Z}^d \text{ finite}} \mathscr{C}(\Lambda)$. It is the σ-algebra generated by this collection that we call $\mathscr{F}$, i.e. $\mathscr{F} = \sigma(\mathscr{C})$. In this σ-algebra, we can find all events that depend on the spins inside $\mathbb{Z}^d$. Notice that $\mathscr{F}$ is generated by collections of events which depend only on a finite amount of spins, but contains also events which depend on the whole of $\mathbb{Z}^d$. This is exactly what we would expect from our event space. It does not contain all events, but it does contain all the events that are relevant to this project.

All of the above is a lot of measure theory, which is certainly important for the rigour of this text. However, it is also important to keep track of what is going on an intuitive level. Let us therefore return to Definition 2.1.3. This one may seem to come out of the blue at first. Why does one define the Hamiltonian for the GFF in such a way? It turns out that this formulation reflects the manner in which we want our Gaussian Free Field to behave really well.

To see this, let us first recall the probability measure given by equation (2.2). In particular, note that the probability of some event, say $A = \{\omega'\}$ with ω' some arbitrary configuration, scales with $e^{-\mathscr{H}_\Lambda(\omega')}$. We disregard Λ for a moment as we look only at the effect of one nearest-neighbour pair on $\mathscr{H}_\Lambda$. In other words, the larger the Hamiltonian becomes for a certain spin configuration, the more negative the exponent becomes and so the less likely it will be for this configuration to occur. This is analog to physics, where in general the states of a system with the lowest energy are the most stable and so the most likely to occur.

We can now investigate both sums in equation (2.1) separately. Observe that the first sum goes over all the nearest-neighbour pairs in Λ. This implies that we only consider interactions between spins at nearest-neighbour points. To learn more about what kind of interactions we are considering, we must look at what is actually inside this sum. One can see the squared difference of the spins at the two neighbouring points. This difference is small for a spin configuration where the spins are close to each other, i.e. $\omega_i \approx \omega_j$. In that case, a small contribution to the Hamiltonian will be made, enlarging the probability of this configuration occurring. One could say that the Gaussian Free Field favours configurations where neighbouring spins are similar in value. In fact, the GFF strives having each spin equal to the average of its neighbouring spin. This is because, for any lattice site i, the average minimises the sum $\sum_{j:j\sim i}(\omega_i - \omega_j)^2$ in the Hamiltonian (2.1):

$$\frac{\partial}{\partial \omega_i}\left(\sum_{j:j\sim i}(\omega_i - \omega_j)^2\right) = 0 \Rightarrow \omega_i = \frac{1}{2d}\sum_{j:j\sim i}\omega_j$$

This is why the GFF is also called the harmonic crystal; it wants to be a harmonic function on $\mathbb{Z}^d$. This interpretation is the key to why we can use the GFF as a model for microscopic fluctuations on a lattice at nonzero temperature. In the zero temperature limit $\beta \to \infty$, the Hamiltonian (2.1) will blow up to infinity for almost all spin configurations. The only configurations for which this blowing up does not happen are those where $(\omega_i - \omega_j)^2 \to 0$ for each $i \sim j$. These are exactly the configurations with constant spin everywhere. As a result, only these configurations are likely to occur. We may see them as 'zero-energy', or ground state, configurations. Once the temperature starts rising however, random fluctuations will happen with nonzero probability and will perturb the constant spin configurations. The higher the temperature, the more severe the random fluctuations and the farther away the DGFF gets distorted from its desired harmonic configuration.

Notice that only the wish for harmonicity of the Gaussian Free Field on an infinite lattice does not say anything about the magnitude of the spin itself. It could be that two neighbouring spins are both very large, but comparable in magnitude. This would be allowed if the Hamiltonian only consisted of the first sum, i.e. in the massless case. However, for the massive case, we see two sums in equation (2.1). This second sum becomes large for large magnitudes of ω_i. Consequently, the probability for a configuration with large ω_i's decreases. The massive GFF thus favours spins that lie close to zero.

2.1.2. The Gaussian Free Field is a Gaussian Field

Until now, we have defined the Gaussian Free Field purely by a certain Hamiltonian and probability measure. At first sight, there is not much Gaussian about the GFF. However, we will show in this section that the Gaussian Free Field is in fact a Gaussian field. To show that we define the random variables $\varphi_i : \Omega \to \mathbb{R}$ by $\varphi_i(\omega) := \omega_i$, for all $i \in \mathbb{Z}^d$. It turns out that under μ_Λ^η, $\varphi_\Lambda := (\varphi_i)_{i\in\Lambda}$ is a Gaussian vector, for any finite $\Lambda \subset \mathbb{Z}^d$. In this section, we will see how to come to this observation. Let us first recall what a Gaussian vector is in Definition 2.1.4.

Definition 2.1.4 (Gaussian vector)**.** Let $\varphi_\Lambda = (\varphi_i)_{i\in\Lambda} \in \Omega_\Lambda$ be an arbitrary random vector. This random vector is called Gaussian if the inner product $c_\Lambda \cdot \varphi_\Lambda := \sum_{i\in\Lambda} c_i\varphi_i$ is a Gaussian random variable, for all constant vectors $c_\Lambda = (c_i)_{i\in\Lambda}$.

If φ_Λ is Gaussian, its distribution is fully determined by its mean $m_\Lambda = (m_i)_{i\in\Lambda} = \left(\mathbb{E}_\Lambda[\varphi_i]\right)_{i\in\Lambda}$ and its covariance matrix Σ_Λ, where the matrix elements are given by $\Sigma_\Lambda(i,j) = \mathrm{Cov}_\Lambda(\varphi_i,\varphi_j)$ for $i,j \in \Lambda$. By Theorem A.1.2, Σ_Λ is symmetric and positive semidefinite. The Λ-subscript in $\mathbb{E}_\Lambda$, Cov_Λ and Var_Λ indicates that we are taking expectations and (co)variances with respect to the distribution μ_Λ of the random vector φ_Λ. We denote φ_Λ being Gaussian by $\varphi_\Lambda \simeq \mathscr{N}(m_\Lambda, \Sigma_\Lambda)$. If Σ_Λ is positive definite, it is regular, i.e. $\det\Sigma_\Lambda \neq 0$. In that case, we can define the density $f_{\varphi_\Lambda}(x_\Lambda) : \Omega_\Lambda \to [0,\infty)$, given by (2.4), using Theorem A.1.3 .

$$f_{\varphi_\Lambda}(x_\Lambda) = \frac{1}{(2\pi)^{\frac{|\Lambda|}{2}}\sqrt{|\det\Sigma_\Lambda|}} e^{-\frac{1}{2}(x_\Lambda - m_\Lambda)\cdot\Sigma_\Lambda^{-1}(x_\Lambda - m_\Lambda)}, \quad \text{for } x_\Lambda \in \Omega_\Lambda. \tag{2.4}$$

Very important for us however is a more general, converse statement.

Theorem 2.1.5. Consider a random vector $\varphi_\Lambda \in \Omega_\Lambda$. If its measure μ_Λ is absolutely continuous with respect to the Lebesgue measure dx_Λ on Ω_Λ, and if the density function with respect to dx_Λ of φ_Λ is given by (2.4), then $\varphi_\Lambda \simeq \mathscr{N}(m_\Lambda, \Sigma_\Lambda)$.

A measure μ is absolutely continuous with respect to another measure ν whenever $\nu(A) = 0$ implies that $\mu(A) = 0$, for any $A \in \mathscr{F}$. This relationship between measures is denoted by $\mu \ll \nu$ [11]. Furthermore, let us state one intermediate result we will need to prove Theorem 2.1.5.

Lemma 2.1.6 (Radon-Nikodym)**.** Let μ be probability measure on $(\Omega, \mathscr{F})$ and let ν be a measure on Ω. Furthermore, assume that μ is absolutely continuous with respect to ν. Then there exists an integrable function $f : \Omega \to [0,\infty)$ such that $\mu(A) = \int_A f\,d\nu$ for all $A \in \mathscr{F}$. That is, μ has a density function with respect to ν.

Proof. This is a very well-known result from measure theory. For a proof, see for example the proof of Theorem 4.3 in Stein and Shakarchi [12]. □

Proof of Theorem 2.1.5. Suppose $\varphi_\Lambda \in \Omega_\Lambda$ is random vector with measure μ_Λ, with $\mu_\Lambda \ll dx_\Lambda$. By Lemma 2.1.6, it makes sense to define a density function for φ_Λ. By assumption, this density function is given by (2.4). This is the density of a Gaussian random vector, which implies that φ_Λ must be a Gaussian random vector, i.e. $\varphi_\Lambda \simeq \mathscr{N}(m_\Lambda, \Sigma_\Lambda)$. □

Now that we know a bit more about Gaussian vectors, we can finally define a Gaussian field.

Definition 2.1.7 (Gaussian field)**.** A Gaussian field is an infinitely large collection of random variables $\varphi = (\varphi_i)_{i\in\mathbb{Z}^d}$ such that for each finite $\Lambda \subset \mathbb{Z}^d$, the random vector $\varphi_\Lambda = (\varphi_i)_{i\in\Lambda}$ is a Gaussian vector.

Remember that we are trying to show that the GFF is such a Gaussian field. In the remainder of this section, we will consider a Gaussian Free Field φ_Λ in an arbitrary, connected finite region $\Lambda \subset \mathbb{Z}^d$. This GFF obeys the measure μ_Λ^η defined by equation (2.2). Moreover, on Ω_Λ, μ_Λ^η is absolutely continuous with respect to the Lebesgue measure, as μ_Λ^η is defined in terms of the Lebesgue measures $d\omega_i$ with $i \in \Lambda$. This means that if we can write μ_Λ^η as a density given by (2.4), we have shown that φ_Λ is a Gaussian vector, using Theorem 2.1.5. As this holds for any finite $\Lambda \subset \mathbb{Z}^d$, Definition 2.1.7 tells us that our GFF is indeed a Gaussian Field.

We will rewrite the Hamiltonian from (2.1) into a desired form using the discrete Laplacian.

Definition 2.1.8 (Discrete Laplacian)**.** Let $x = (x_i)_{i\in\mathbb{Z}^d} \in \mathbb{R}^{\mathbb{Z}^d}$ be a collection of real numbers. For each pair $(i,j) \in \mathscr{E}_{\mathbb{Z}^d}$, define the discrete gradient $(\nabla x)_{ij} := x_j - x_i$. Then, for all $i \in \mathbb{Z}^d$, the discrete Laplacian is given by $(\Delta x)_i := \sum_{j:j\sim i} (\nabla x)_{ij}$.

It is easy to see that we can write $(\Delta x)_i = \sum_{j\in\mathbb{Z}^d} \Delta_{ij} x_j$ for all $i \in \mathbb{Z}^d$, where we define the discrete Laplacian matrix $(\Delta_{ij})_{i,j\in\mathbb{Z}^d}$ by:

$$\Delta_{ij} = \begin{cases} -2d, & i = j, \\ 1, & \|i-j\|_1 = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

The main idea here is to let this Laplacian matrix play the role of the inverse covariance matrix Σ_Λ^{-1} in (2.4). This holds for general x; so we may switch to φ later on. As this is all happening within Λ, we restrict ourselves to work with $\Delta_\Lambda := (\Delta_{ij})_{i,j\in\Lambda}$. Before proceeding, let us state a discrete version of Green's first identity.

Theorem 2.1.9 (Discrete Green's identity)**.** For all $x = (x_i)_{i\in\mathbb{Z}^d}, y = (y_i)_{i\in\mathbb{Z}^d} \in \mathbb{R}^{\mathbb{Z}^d}$, we have

$$\sum_{(i,j)\in\mathscr{E}_\Lambda} (\nabla x)_{ij} (\nabla y)_{ij} = -\sum_{i\in\Lambda} y_i (\Delta x)_i + \sum_{\substack{i\in\Lambda, j\in\Lambda^c \\ i\sim j}} y_j (\nabla x)_{ij}. \tag{2.6}$$

Proof. Recall that $\mathscr{E}_\Lambda$ is the edge-set of Λ, as defined in Definition 2.1.2. Note that,

$$\begin{aligned}
\sum_{(i,j)\in\mathscr{E}_\Lambda} (\nabla x)_{ij}(\nabla y)_{ij} &= \sum_{(i,j)\in\mathscr{E}_\Lambda} y_j(x_j - x_i) - \sum_{(i,j)\in\mathscr{E}_\Lambda} y_i(x_j - x_i) \\
&\overset{(1)}{=} \sum_{(i,j)\in\mathscr{E}_\Lambda} y_i(x_i - x_j) - \sum_{(i,j)\in\mathscr{E}_\Lambda} y_i(x_j - x_i) \\
&= -2 \sum_{(i,j)\in\mathscr{E}_\Lambda} y_i(x_j - x_i) \\
&\overset{(2)}{=} -\sum_{i\in\Lambda} y_i \sum_{j\in\Lambda: j\sim i} (x_j - x_i) \\
&\overset{(3)}{=} -\sum_{i\in\Lambda} y_i \sum_{j\sim i}(x_j - x_i) + \sum_{i\in\Lambda} y_i \sum_{j\in\Lambda^c} (x_j - x_i) \\
&= -\sum_{i\in\Lambda} y_i(\Delta x)_i + \sum_{i\in\Lambda} y_i \sum_{j\in\Lambda^c} (\nabla x)_{ij}
\end{aligned}$$

At (1), we have interchanged the i and the j for the first sum. This does not change the value of the sum and allows us to add the two sums up, as is done at the next equality. Observe that at the equality (2), we have rewritten the sum such that we in fact count all neighbour pairs twice. Indeed, we sum over all $i \in \Lambda$, but in the second sum we go over all $j \in \Lambda$ with $j \sim i$. That is why the 2 is left out. We recognise that the inner sum is in fact the discrete Laplacian for the point i. However, for the values of $i \in \Lambda$ which have one or more neighbours outside of Λ, we cannot replace $\sum_{j\in\Lambda:j\sim i}(x_j - x_i)$ by $(\Delta x)_i$. We therefore add and substract $\sum_{i\in\Lambda} y_i \sum_{j\in\Lambda^c}(x_j - x_i)$, as is done in equality (3).

The statement we want to show is now quite easily found:

$$\begin{aligned}
\sum_{(i,j)\in\mathscr{E}_\Lambda} (\nabla x)_{ij}(\nabla y)_{ij} &= \sum_{(i,j)\in\mathscr{E}_\Lambda} (\nabla x)_{ij}(\nabla y)_{ij} + \sum_{\substack{i\in\Lambda, j\in\Lambda^c \\ i\sim j}} (\nabla x)_{ij}(\nabla y)_{ij} \\
&\overset{(*)}{=} -\sum_{i\in\Lambda} y_i(\Delta x)_i + \sum_{i\in\Lambda} y_i \sum_{j\in\Lambda^c} (\nabla x)_{ij} + \sum_{\substack{i\in\Lambda, j\in\Lambda^c \\ i\sim j}} (y_j - y_i)(\nabla x)_{ij} \\
&= -\sum_{i\in\Lambda} y_i(\Delta x)_i + \sum_{\substack{i\in\Lambda, j\in\Lambda^c \\ i\sim j}} y_j(\nabla x)_{ij},
\end{aligned}$$

where at $(*)$ we used our earlier observation. □

We can start rewriting the Hamiltonian $\mathscr{H}_\Lambda$ by applying Theorem 2.1.9 with $y = x$

on the term $\sum_{(i,j)\in\mathscr{E}_\Lambda}(x_i-x_j)^2$, in the following way:

$$\begin{aligned}\sum_{(i,j)\in\mathscr{E}_\Lambda}(x_i-x_j)^2 &= \sum_{(i,j)\in\mathscr{E}_\Lambda}(\nabla x)_{ij}(\nabla x)_{ij}\\ &= -\sum_{i\in\Lambda}x_i(\Delta x)_i + \sum_{\substack{i\in\Lambda,j\in\Lambda^c\\ i\sim j}}x_j(\nabla x)_{ij}\\ &\overset{(1)}{=} -\sum_{i\in\Lambda}x_i(\Delta_\Lambda x)_i - \sum_{\substack{i\in\Lambda,j\in\Lambda^c\\ i\sim j}}x_ix_j + \sum_{\substack{i\in\Lambda,j\in\Lambda^c\\ i\sim j}}x_j^2 - x_ix_j\\ &\overset{(2)}{=} -x\cdot\Delta_\Lambda x - 2\sum_{\substack{i\in\Lambda,j\in\Lambda^c\\ i\sim j}}x_ix_j + B_\Lambda. \end{aligned}\tag{2.7}$$

At (1), we used that for any $i\in\Lambda$, $(\Delta x)_i=(\Delta_\Lambda x)_i+\sum_{j\in\Lambda^c}x_j$ to split up the first sum. Also at (1), we used that $x_j(\nabla x)_{ij}=x_j^2-x_ix_j$ for all $i\in\Lambda$, $j\in\Lambda^c$ with $i\sim j$. At (2), we introduced the term $B_\Lambda=\sum_{i\in\Lambda,j\in\Lambda^c:i\sim j}x_j^2$. This term only depends on the j's outside of Λ. Since the region Λ^c was fixed by the boundary condition η, we have $x_j=\eta_j$ for all $j\in\Lambda^c$ and so we replace the sum by B_Λ, which we call the boundary term.

In the expression given at (2), we can already recognise the $x\cdot\Delta_\Lambda x$ term as the form we would like to have, but not quite yet. Up to boundary terms, we want to rewrite this expression such that it contains a term of the form $-(x-u)\cdot\Delta_\Lambda(x-u)$. Here $u=(u_i)_{i\in\mathbb{Z}^d}$ acts as the mean of x. What it is exactly will be determined later on. Observe that:

$$\begin{aligned}-(x-u)\cdot\Delta_\Lambda(x-u) &= x\cdot\Delta_\Lambda x - 2x\cdot\Delta_\Lambda u + u\cdot\Delta_\Lambda u\\ &\overset{(3)}{=} x\cdot\Delta_\Lambda x - 2\sum_{i\in\Lambda}x_i(\Delta u)_i + 2\sum_{\substack{i\in\Lambda,j\in\Lambda^c\\ i\sim j}}x_iu_j + B_\Lambda,\end{aligned}\tag{2.8}$$

where at (3) we again use that for any $i\in\Lambda$, $(\Delta u)_i=(\Delta_\Lambda u)_i+\sum_{j\in\Lambda^c}u_j$. We also included everything that was only a function of u inside the boundary term B_Λ.

With (2.7) and (2.8), we now have two expressions for $x\cdot\Delta_\Lambda x$. Combining these two, we find that,

$$\sum_{(i,j)\in\mathscr{E}_\Lambda}(x_i-x_j)^2 = -(x-u)\cdot\Delta_\Lambda(x-u) - 2\sum_{i\in\Lambda}x_i(\Delta u)_i + 2\sum_{\substack{i\in\Lambda,j\in\Lambda^c\\ i\sim j}}x_i(u_j-x_j) + B_\Lambda \tag{2.9}$$

This is a very nice result. For a specific u, it allows us to replace $\sum_{(i,j)\in\mathscr{E}_\Lambda}(x_i-x_j)^2$ in the form found in the Gaussian density (2.4), up to a constant boundary term B_Λ. Indeed, as the B_Λ only depends on η, which is given, the boundary term does not alter the distribution (2.2). This specific u is of course the one such that the two sums in (2.9) are zero. That is, (i) for all $i \in \Lambda$, $(\Delta u)_i = 0$ (u is harmonic in Λ) and (ii) $u_j = x_j = \eta_j$ for $j \in \Lambda^c$. If we now apply this argument to the massless GFF, with $x = \varphi$ such that $\varphi_i = \eta_i$ for $i \in \Lambda^c$, we find that for u satisfying conditions (i) and (ii), up to the boundary term,

$$\mathscr{H}^{\eta}_{\Lambda}(\varphi) = -\frac{\beta}{4d}(\varphi - u)\cdot\Delta_\Lambda(\varphi - u) = \frac{1}{2}(\varphi - u)\cdot\left(-\frac{1}{2d}\Delta_\Lambda\right)(\varphi - u) \tag{2.10}$$

As the density of the GFF scales with $e^{-\mathscr{H}^{\eta}_{\Lambda}}$, we can conclude that φ_Λ is a Gaussian vector with mean u and covariance matrix $\left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1}$. We also say that φ_Λ is a Gaussian vector with mean u and precision matrix $-\frac{1}{2d}\Delta_\Lambda$. This precision matrix will be especially helpful when we try to generate a (modified) Gaussian Free Field in Chapter 3.

The question now is, of course, how to determine u. Recall that u must satisfy the two conditions (i) and (ii) as defined earlier. In other words, u is a solution to the following Dirichlet problem in Λ with boundary condition η:

$$\begin{cases} (\Delta u)_i = 0, & \forall i \in \Lambda, \\ u_j = \eta_j, & \forall j \in \Lambda^c. \end{cases} \tag{2.11}$$

We will do this by inverting the precision matrix $-\frac{1}{2d}\Delta_\Lambda$. Apart from leading to a solution for (2.11), the inverse will immediately give the covariance matrix of the Gaussian vector φ_Λ.

First of all, remark that $-\frac{1}{2d}\Delta_\Lambda = I_\Lambda - P_\Lambda$, with I_Λ the identity matrix on Λ and $P_\Lambda = \big(P(i,j)\big)_{i,j\in\Lambda}$ with:

$$P(i,j) = \begin{cases} \frac{1}{2d}, & i \sim j, \\ 0, & \text{otherwise.} \end{cases} \tag{2.12}$$

One should recognise that the $\big(P(i,j)\big)_{i,j\in\mathbb{Z}^d}$ are in fact transition probabilities of a symmetric simple random walk $X = (X_n)_{n\geq 0}$ on $\mathbb{Z}^d$. That is, $\mathbb{P}_i\big(X_{n+1} = k | X_n = j\big) = P(j,k)$, where $\mathbb{P}_i$ (and similarly $\mathbb{E}_i$ and Var_i) indicate the distribution for the random walk starting in i, i.e. $\mathbb{P}_i(X_0 = i) = 1$. Indeed, at each time step k the random walk jumps with equal probability to one of its $2d$ nearest neighbours.

Let us now state an important result regarding the first exit time τ_{Λ^c} of the random walk from a finite $\Lambda \subset \mathbb{Z}^d$. This is the first time the random walk takes a value which does not lie in Λ, i.e. $\tau_{\Lambda^c} := \inf\{n \geq 0 : X_n \in \Lambda^c\}$.

Theorem 2.1.10. For Λ a finite subset of $\mathbb{Z}^d$, we have $\mathbb{P}_i(\tau_{\Lambda^c} < \infty)$. More precisely, there exists a $c = c(\Lambda) > 0$ such that for all $i \in \Lambda$, we have $\mathbb{P}_i(\tau_{\Lambda^c} > n) \leq e^{-cn}$.

Proof. See Friedli and Velenik, Proof of Lemma 8.12 [5]. □

Theorem 2.1.10 is necessary to give an expression for the inverse of $I_\Lambda - P_\Lambda$.

Theorem 2.1.11. The matrix $I_\Lambda - P_\Lambda$ is invertible and the inverse matrix $(I_\Lambda - P_\Lambda)^{-1}$ is given by the Green's function $G_\Lambda = \left(G_\Lambda(i,j)\right)_{i,j\in\Lambda}$ in Λ of a simple random walk on $\mathbb{Z}^d$. That is, the matrix elements of G_Λ are given by $G_\Lambda(i,j) := \mathbb{E}_i\left[\sum_{n=0}^{\tau_{\Lambda^c}-1} \mathbb{1}_{\{X_n=j\}}\right]$.

Proof. See Friedli and Velenik, Proof of Lemma 8.13 [5]. □

Note that the Green's function $G_\Lambda(i,j)$ is the expected number of visits that the random walk makes at j before leaving Λ, if the walk started at i. Using Theorem 2.1.11, one can show that the solution to the Dirichlet problem (2.11) is given by $u = (u_i)_{i\in\mathbb{Z}^d}$ where $u_i := \mathbb{E}_i\left[\eta_{X_{\tau_{\Lambda^c}}}\right]$. Consequently, under μ_Λ^η, $\varphi_\Lambda \simeq \mathcal{N}(u_\Lambda, G_\Lambda)$. In other words, we have found the distribution of the Discrete Gaussian Free Field. Important to remark is that the covariance matrix depends only on how we choose Λ. The boundary condition η can thus only affect the distribution of φ_Λ via its mean [5].

2.2. Level-set Percolation

Now that we have a better idea of what the Gaussian Free Field is, let us take a look at what we will actually do with it. Consider a DGFF $(\varphi_i)_{i\in\mathbb{Z}^d}$. What follows also holds for other random fields, but we will solely be working with Discrete Gaussian Free Fields for the moment. For each $h \in \mathbb{R}$, we define the set $E^{\geq h} := \left\{i \in \mathbb{Z}^d : \varphi_i \geq h\right\} \subset \mathbb{Z}^d$. This set is called the level-set above h, or the excursion set. It contains all the lattice points for which the associated spin is large than some threshold value h.

In short, level-set percolation revolves around the question whether $E^{\geq h}$ contains infinite connected components. We speak of a connected component between two lattice points $K, K' \in \mathbb{Z}^d$ whenever there exists a path along nearest-neighbours $(i)_{i=K}^{K'}$ such that $\varphi_i \geq h$ for all i on the path [10]. A group C of nearest-neighbour lattice sites with $\varphi_i \geq h$ for all $i \in C$ is called a cluster. We can therefore say that K and K' belong to the same cluster. We have an infinite connected component whenever a cluster contains infinitely many lattice sites. The existence of such infinite connected components, also called infinite clusters, is a main topic in percolation theory. In fact, the word percolation comes from the Latin *percolare*, which is a contraction of *per* (meaning "through") and *colare* (meaning "to sieve")[13]. Intuitively, one could see the lattice $\mathbb{Z}^d$ as some porous material. If an infinite cluster exists then a fluid may "sieve through" the lattice.

It may be clear that the existence of an infinite cluster depends on how "high" our threshold value h is. The higher h becomes, the less spins will be larger than h, so it will be more likely that there are no infinite clusters. However, one can ask themselves where the transition lies between there being or not being an infinite cluster. This transition happens at the so-called critical value.

Definition 2.2.1. Given a DGFF $(\varphi_i)_{i\in\mathbb{Z}^d}$, we define the critical value by:

$$h_* := \inf\left\{h \in \mathbb{R} : \mathbb{P}\Big(E^{\geq h}\text{contains an infinite cluster}\Big) = 0\right\} \in [-\infty,\infty]. \tag{2.13}$$

The critical value can be seen as the lowest possible value of h for which we do not have an infinite cluster [1]. The probability inside this definition is often called the percolation probability. If $|h_*| < \infty$, one says that there is a percolation transition [2]. In the context of physics, a percolation transition is a phase transition. The most obvious example of a phase transition in general would be the sudden transition from ice to water. Just below the critical temperature of 0°C, we have ice and just above the critical temperature, we have water. Phase transitions also occur in other spin models. Probably the most well-known of these models is the Ising model. This model can also be built on $\mathbb{Z}^d$, just as the DGFF. In contrast to the DGFF however, the single-spin space of the Ising model is $\{-1,+1\}$. The Hamiltonian of the model favours nearest-neighbours with the same spin (both $+1$ or both -1). This is again similar to the DGFF. The Ising model on Z^2 is one of the most basic models that exhibits a phase transition.

2.2.1. BEHAVIOUR CLOSE TO THE CRITICAL VALUE

It is very insightful to spend some time on studying the behaviour of a cluster in the lattice near the critical value. This section works towards a result which will allow us to give an estimate of the critical value h_* based on finite approximations of a DGFF. Let us start however with a simpler percolation model. We follow the texts of Christensen and of Stauffer and Aharony here [14][15].

Consider the integer line $\mathbb{Z}$. Each integer is a lattice site and is either occupied, with probability p or unoccupied, with probability $1-p$. This happens independently from all the other lattice sites. We speak of a Bernoulli percolation model, as the occupation variable for each site i is a Bernoulli random variable with parameter p. Notice the similarity here with the earlier level-set percolation model on the DGFF. One could say that a lattice site i with $\varphi_i \geq h$ is occupied. This happens with a certain probability p, which depends on the threshold value h. One should be careful however, as for the level-set percolation model on the DGFF a site i being occupied or not depends on the values φ_j of the neighbouring sites $j \sim i$. Although this is a fundamental difference, many of the results will hold for both models.

Consider the one-dimensional Bernoulli percolation model on $\mathbb{Z}$. We are particularly interested in clusters in this model, and how to describe them quantitatively. A cluster is a collection of nearest-neighbour occupied sites. A first quantity we define is the cluster number $n_s(p)$. This is the amount of clusters with s sites divided by the total amount of lattice sites. In other words, $n_s(p)$ the number of clusters containing s lattice sites, per lattice site. This number is of course a function of p; intuitively a larger p will lead to more clusters. Due to the definition per lattice site, n_s is independent of the lattice size N. For example, if we would have a

d-dimensional hypercubic lattice with side-length N, the average number of clusters would be $N^d n_s(p)$. Second, we define the percolation threshold p_c. This is the probability p at which a cluster containing infinitely many sites appears for the first time. This quantity corresponds to the critical value h_* defined in Definition 2.2.1. However, keep in mind that $h_* \in \mathbb{R}$ is a threshold height for spins, while $p_c \in [0,1]$ is an occupation probability. In Section 3.3.2 we will see how to go from p_c to h_* and vice versa.

In the one-dimensional case, we have percolation whenever a cluster spans from $-\infty$ to $+\infty$. This will only happen whenever all the sites on the integer line are occupied, as just one unoccupied site would 'break' the cluster. Therefore, $p_c = 1$. Indeed, the probability $\Pi(p,N)$ that a finite lattice with size N contains a spanning cluster at occupation probability p is p^N. Also, note that by definition, $\lim_{N\to\infty} \Pi(p,N) = \mathbb{1}_{\{p \geq p_c\}}$. But $\lim_{N\to\infty} p^N = \mathbb{1}_{\{p=1\}}$, so $p_c = 1$.

Remember that the goal of this section was to study the behaviour of clusters around the percolation threshold p_c. Therefore, let us take a closer look at the quantity $n_s(p)$. In the 1D case, we have a cluster of size s whenever s occupied sites have one unoccupied site to the left and one unoccupied site to the right. We could say that the probability of an arbitrary site being the left unoccupied site is

$$n_s(p) = (1-p)p^s(1-p) = (1-p)^2 p^s = (p_c - p)^2 e^{-\frac{s}{s_\xi}},$$

where we defined the cutoff cluster size s_ξ as

$$s_\xi = -\frac{1}{\ln(p)} = -\frac{1}{\ln\left(p_c - (p_c - p)\right)}.$$

As $p \uparrow p_c$, the latter goes to $(p_c - p)^{-1}$ due to the Taylor expansion $\ln(1-x) \approx -x$ for $x \to 0$. Intuitively, one can see the cutoff cluster size as a maximal size for cluster at a given p. It is very unlikely for a cluster to have size larger than s_ξ, due to the exponential decay in the expression for $n_s(p)$. As $p \uparrow p_c$, we see that s_ξ diverges as a power law.

Definition 2.2.2. For $p \to p_c$, we define the critical exponent σ for the cutoff cluster size s_ξ by

$$s_\xi \sim |p_c - p|^{-\frac{1}{\sigma}}. \tag{2.14}$$

We have stated this power law behaviour as a definition, because it is very typical for percolation models close to p_c. Different models may have a different value for σ. For our 1D percolation, we see that $\sigma = 1$. One important remark is that other models might have $p_c < 1$. Most of the time, the power law behaviour holds for both limits $p \uparrow p_c$ and $p \downarrow p_c$. In the remainder of this text, we will assume this is always the case, unless explicitly said otherwise.

Not only the cutoff cluster size shows this power law behaviour. Several other quantities related to clusters do so as well, albeit with a different exponent. Note that we may not be able to analytically derive the exponents for more complicated models. In any case, we will present a few others which are needed to understand the method to find the percolation threshold from finite-size simulations, described in Section 3.3.2.

The first of such quantities is the average cluster size $S(p) = \sum_{s=1}^{\infty} s w_s$. Here w_s is the probability that the cluster to which an occupied site belongs contains s sites, so:

$$\begin{aligned} w_s &= \mathbb{P}\big(\text{arbitrary site belongs to } s\text{-cluster} \mid \text{site is occupied}\big) \\ &= \frac{\mathbb{P}\big(\text{arbitrary site belongs to } s\text{-cluster and is occupied}\big)}{\mathbb{P}\big(\text{site is occupied}\big)} = \frac{s n_s(p)}{p} \end{aligned}$$

By realising that the probability that an arbitrary site belongs to any cluster, which is given by $\sum_{i=1}^{\infty} s n_s(p)$, is in fact just the probability p, Christensen shows that $S(p) = \frac{1+p}{1-p}$. Thus for $p \uparrow p_c = 1$, we have $S(p) \sim (p_c - p)^{-1}$. Again, we recognise a power law, but now with the exponent $\gamma = 1$. This exponent is defined in general by Definition 2.2.3.

Definition 2.2.3. For $p \to p_c$, we define the critical exponent γ for the average cluster size $S(p)$ by

$$S(p) \sim |p_c - p|^{-\gamma}. \tag{2.15}$$

If $p_c < 1$, there exists an infinite cluster whenever $p > p_c$. This cluster is not taken into account when computing $S(p)$.

Moving on, we define the strength (also order parameter) $P(p)$. This is the probability that an arbitrary site belongs to the infinite cluster. For $p < p_c$, we have trivially $P(p) = 0$ as there is no infinite cluster yet. For models where $p_c < 1$, just as the Bethe lattice described in Christensen, we can let $p \downarrow p_c$. It can be shown that $P(p)$ exhibits power law behaviour whenever $p > p_c$.

Definition 2.2.4. For $p \downarrow p_c$, we define the critical exponent β for the strength $P(p)$ by

$$P(p) \sim |p_c - p|^{\beta}. \tag{2.16}$$

It is when $P(p)$ becomes larger than zero, for $p > p_c$ that we speak of a phase transition with critical point $p = p_c$. We transition from a phase where there are only finite clusters, to a phase where a large, percolating cluster is formed.

Finally, we define the correlation function $g(\mathbf{r})$ as the probability that a site at position $\mathbf{r}$ from an occupied site belongs to the same finite cluster. In one dimension, we clearly have $g(\mathbf{r}) = p^r$, as all sites in between should be occupied as well.

We can rewrite this as $g(\mathbf{r}) = e^{\ln p^r} = e^{-\frac{r}{\xi}}$. Here we introduced the correlation length $\xi = -\frac{1}{\ln p}$, which again goes to $(p_c - p)^{-1}$ for $p \to p_c$ in one dimension. This analytic computation to retrieve the correlation length from the correlation function works out nicely for the 1D case. However, in higher dimensions the clusters become more complex and so $g(\mathbf{r})$ becomes very difficult to calculate. We therefore give a explicit expression, or rather a new definition, for the correlation length in Definition 2.2.5.

Definition 2.2.5. The correlation length ξ is defined as

$$\xi^2 = \frac{\sum_{\mathbf{r}} r^2 g(\mathbf{r})}{\sum_{\mathbf{r}} g(\mathbf{r})}. \tag{2.17}$$

We have essentially defined the correlation length as the root-mean-square finite cluster size. We can show that this definition yields the same scaling behaviour for ξ in one dimension as we found earlier.

$$\xi^2 = \frac{\sum_{\mathbf{r}} r^2 g(\mathbf{r})}{\sum_{\mathbf{r}} g(\mathbf{r})} = \frac{\sum_{r=0}^{\infty} r^2 p^r}{\sum_{r=0}^{\infty} p^r} = \frac{\left(p\frac{\mathrm{d}}{\mathrm{d}p}\right)^2 \sum_{r=0}^{\infty} p^r}{(1-p)^{-1}} = \frac{p(1+p)(1-p)^{-3}}{(1-p)^{-1}} = \frac{1+p}{(1-p)^2}.$$

Indeed, $\xi^2 \sim (1-p)^{-2}$ so $\xi \sim (1-p)^{-1} = (p_c-p)^{-1}$ for $p \to p_c$. Again, we may capture this behaviour as a general power law.

Definition 2.2.6. For $p \to p_c$, we define the critical exponent ν for the correlation length ξ by

$$\xi \sim |p_c - p|^{-\nu}. \tag{2.18}$$

In one dimension, $\nu = 1$. Looking at the formulation $g(\mathbf{r}) = e^{-\frac{r}{\xi}}$, one can interpret the correlation length as a cutoff length. The probability that a site a distance $r > \xi$ away from an occupied site belongs to the same cluster becomes very small due to the exponential decay. This is the same as saying that ξ gives the typical linear size of the largest finite cluster in the lattice [16]. Because of this, the correlation length can be used as a physical length scale; it is a 'measuring stick' for cluster size. We can however only do this when $p \neq p_c$. Indeed, by (2.18), ξ diverges at $p = p_c$. In that case, our measuring stick has an infinite length and is thus completely useless; everything is shorter than the measuring stick. This absence of a length scale, unique to $p = p_c$, is called scale invariance. It does not matter how far we zoom in or zoom out when looking at percolation on a lattice, it will always look the same. In other words, it is self-similar. We see this behaviour in the context of fractals. For example, consider Figure 2.1. This is a so-called Koch curve. It can be created by iteratively replacing one line segment by four smaller line segments, as depicted in Figure 2.2. If we zoom in on one of the coloured segment, we will see

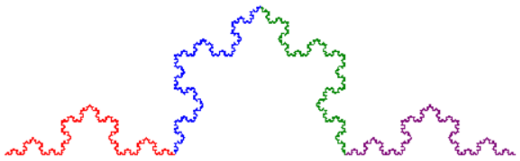


Figure 2.1: The Koch curve. If we zoom in on one of the coloured segment, we see exactly the same curve as before [17].

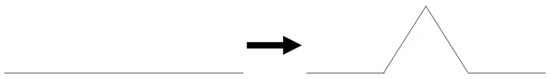


Figure 2.2: Sketch of the iterative procedure to obtain a Koch curve. If we repeat this transformation infinitely many times on each newly formed line segment, we end up with Figure 2.1.

exactly the same curve as before. This can be infinitely repeated. The Koch curve is therefore a self-similar (fractal) object.

We have said earlier that this scale-invariance only occurs at $p = p_c$, however this is not entirely true. When we consider a percolation system with $p \neq p_c$ on a length scale far less than ξ, it seems for us just as if ξ is infinite. In the words of Hunt, Ewing and Ghanbarian: "if one examines the system at length scales smaller than the largest self-similar structure [viz. the largest finite cluster], the medium appears to be self-similar". When working on length scales larger than ξ, this fractal geometry disappears and we are left with a lot of small patches of occupied sites in an otherwise unoccupied lattice for $p < p_c$, and vice versa for $p > p_c$. We conclude that there is only one relevant length scale when studying percolation, and that is ξ. Depending on whether we are looking at a percolation system on a scale smaller than ξ or larger than ξ, we will observe different behaviour.

This is a very powerful idea, which is at the basis of the finite-size scaling relation at the end of this Section. Before we get there however, let us treat one example where it becomes really clear how a percolation system behaves depending on its size relative to the correlation length.

We denote by $M(N)$ the mass (number of occupied sites) of the percolating cluster in some region with linear distance N. If this cluster were to be densely packed within the region, we would expect that $M(N) \sim N^d$, with d the Euclidean dimension we are working in. However, the percolating cluster is not densely packed and is accompanied by large unoccupied regions when $p = p_c$. It is a fractal object. This is illustrated in Figure 2.3. There, one can see three realisations of the Bernoulli site percolation model on a 30×30 lattice, each with a different occupation probability p. The value of p is not determined analytically, however it has been determined by simulation that $p_c \approx 0.593$ [18]. In Figure 2.3a, we have set $p = 0.25$. One can see that most sites are unoccupied, and the clusters that are present are still relatively small. Figure 2.3c shows the model for $p = 0.75$, far above the critical probability. In that case we observe that most sites are occupied. If we would consider small $N \times N$ squares with increasing values of N within the larger 30×30 lattice, we would expect that $M(N)$ grows linearly with the area N^2 of the small square. This is indeed what happens for $p = 0.75 > p_c$. However, for $p \to p_c$, we see something different. In Figure 2.3b, we have $p = 0.6$. This is close to, but greater than, the critical probability. Notice that the largest cluster contains a lot of holes, i.e. groups of neighbouring unoccupied cells. Moreover, some holes contain smaller clusters of their own. As Stauffer and Aharony put it, there are holes on many length scales [15]. There exists an 'infinite' cluster at $p = 0.6$, but this cluster contains a lot of holes. It is not as packed as the infinite cluster at larger p such as $p = 0.75$.

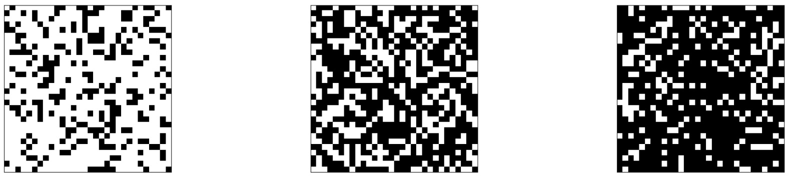

(a) $p = 0.25$. (b) $p = 0.6$. (c) $p = 0.75$.

Figure 2.3: Bernoulli percolation with occupation probability p on a 30×30 lattice. The black squares are occupied sites, the white squares are unoccupied sites. Underneath each figure, the used value of p is specified.

This also implies that we no longer have $M(N) \sim N^2$. It is shown in [15] that at $p = p_c$, $M(N) \sim N^D$, with $D = \frac{91}{48} \approx 1.9$. We call D the fractal dimension. This dimension captures the fractal behaviour observed around criticality, and depends on the percolation model under consideration. For example, in the case of three dimensional Bernoulli site percolation, $D = 2.5$.

As stated earlier, the only relevant length when investigating critical behaviour is the correlation length ξ. Let us make this idea concrete for the mass $M(N)$. When $N \ll \xi$, the percolating cluster is a fractal object, so $M(N) \sim N^D$. If $N \gg \xi$, the cluster seems to be homogeneous in shape. We can therefore divide the lattice into $(N/\xi)^d$ boxes of linear size ξ. For each of these boxes, $M_{\text{box}} \sim \xi^D$, as the linear lattice size is now comparable to ξ. Thus the total mass of the percolating cluster is $M(N,\xi) = (N/\xi)^d \xi$. We can combine the results for $N \ll \xi$ and $N \gg \xi$ into equation (2.19).

$$M(N,\xi) \sim \begin{cases} N^D, & N \ll \xi, \\ \xi^D \left(\frac{N}{\xi}\right)^d, & N \gg \xi. \end{cases} =: N^D m\left(\frac{N}{\xi}\right). \tag{2.19}$$

In the second equality we defined the function $m(x)$, which equals 1 for $x \ll 1$ and x^{d-D} for $x \gg 1$.

FINITE-SIZE SCALING

Finally, we can introduce the important relation we will need in the next chapter. This is in fact the generalisation of (2.19), for arbitrary cluster quantities. We would like to know how an arbitrary cluster quantity behaves on a finite lattice with linear size N. As before, the only relevant scale in this problem is the correlation length ξ. For $N \gg \xi$, nothing changes compared to the infinite lattice case. If $N \ll \xi$, the smallest length scale is N. In general, if we have some quantity $X \sim |p - p_c|^{-\chi} \sim \xi^{\frac{\chi}{\nu}}$ whenever $N \gg \xi$, then we find:

$$X(N,\xi) \sim \begin{cases} N^{\frac{\chi}{\nu}}, & N \ll \xi, \\ \xi^{\frac{\chi}{\nu}}, & N \gg \xi. \end{cases} = \xi^{\frac{\chi}{\nu}} f_1\left(\frac{N}{\xi}\right) \tag{2.20}$$

Alternatively, we can write $X(N,p) \sim |p - p_c|^{-\chi} f_2\left(N^{\frac{1}{\nu}}(p - p_c)\right)$. The functions f_1 and f_2 are explicitly given by:

$$f_1(x) = \begin{cases} x, & x \ll 1, \\ 1, & x \gg 1. \end{cases}, \quad f_2(x) = \begin{cases} x^{\chi}, & x^{\nu} \ll 1, \\ 1, & x^{\nu} \gg 1. \end{cases}.$$

Keep in mind that these relations are all asymptotically valid, and will work better and better for larger and larger N.

2.3. THE DGFF WITH RANDOM CONDUCTANCES

IT has been said before, the goal of this research is to simulate a modified version of the Discrete Gaussian Free Field. This modified DGFF has arbitrary conductances between the lattice points. Let us define what we mean by that.

Again, we consider a Discrete Gaussian Free Field $(\varphi_i)_{i\in\mathbb{Z}^d}$. Recall from Section 2.1.2 that for any finite $\Lambda \subset \mathbb{Z}^d$, the restriction $\varphi_\Lambda = (\varphi_i)_{i\in\Lambda}$ is normally distributed with mean u and covariance matrix $\left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1}$, where $u = (u_i)_{i\in\Lambda}$ satisfies the Dirichlet problem (2.11).

We assign a conductance to each edge in $\mathbb{Z}^d$. That is, for all $i, j \in \mathbb{Z}^d$ with $i \sim j$, we introduce a constant $\kappa_{i,j} \in [\lambda, 1]$, where $\lambda \in (0,1]$. Now define a new graph Laplacian Δ^κ_Λ on Λ, similar to (2.5), but with one key difference.

$$\Delta^\kappa_{ij} = \begin{cases} -\sum_{k\sim i} \kappa_{i,k}, & i = j, \\ \kappa_{i,j}, & ||i - j||_1 = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{2.21}$$

Notice that in the original graph Laplacian, each nearest-neighbour edge had a contribution of 1 to the Laplacian. Now, this contribution can vary between λ and 1. In other words, the original graph Laplacian is a special case of 2.21 with $\kappa_{i,j} = 1$ for all $i, j \in \Lambda$. The DGFF with random conductances is then given by

$\varphi^\kappa = (\varphi_i^\kappa)_{i\in\mathbb{Z}^d}$. Here $\varphi_\Lambda^\kappa = (\varphi_i^\kappa)_{i\in\Lambda}$ is normally distributed with the same mean u as φ_Λ, but now with covariance matrix $\left(-\frac{1}{2d}\Delta_\Lambda^\kappa\right)^{-1}$. This modified GFF is associated with the Hamiltonian (2.22) [1]. Note that this is all happening in the massless setting.

$$\mathcal{H}_\Lambda^\kappa(\omega) := \frac{\beta}{4d}\sum_{(i,j)\in\mathcal{E}_\Lambda}\kappa_{i,j}(\omega_i-\omega_j)^2 \tag{2.22}$$

Intuitively, the conductances determine how strongly two lattice points are "connected" to each other. A smaller conductance allows for the squared difference of spins in the Hamiltonian (2.22) to be slightly larger. In other words, a larger conductance between two lattice points makes that the corresponding spins lie closer to each other.

As long as the $\kappa_{i,j}$ are chosen between λ and 1 with $\lambda \in (0,1]$ for all nearest-neighbours i and j, we may choose any conductance pattern we fancy and construct a DGFF with it. For example, we could consider a constant conductance pattern, where every now and then a small group of edges with a different conductance appear. Physically, this could model some kind of impurity within an otherwise homogeneous crystal lattice. In Chapter 3, we present a method that allows us to simulate such a DGFF. Although most of the simulations carried out in this project are with standard DGFFs, we will look at one non-constant conductance pattern, which we call the checkerboard pattern. A lattice with side-length N is built up by smaller cubes with side-length n. Within each separate cube the conductances are the same, but two different cubes may have a different conductance. In the checkerboard lattice, there are two types of cubes; one type with conductance a and the other type with conductance b. These cubes alternate each other such that no two cubes share a common side. This is illustrated in two dimensions in Figure 2.4. One can imagine how this checkerboard scales into three dimensions, yielding an alternating pattern of cubes instead of squares. Note that squares (cubes) that share an edge with the boundary of the lattice itself, are linked to that boundary with the conductances as within the square (cube).

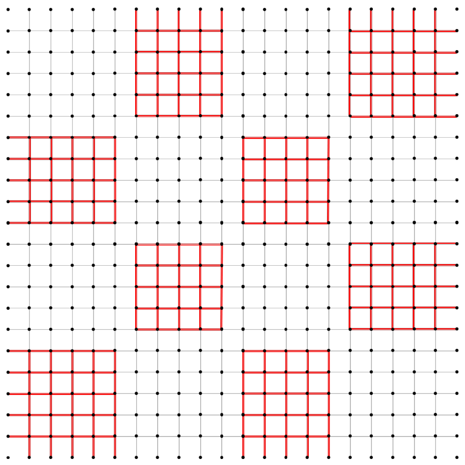


Figure 2.4: Checkerboard pattern on a 20×20 lattice, where the squares building up the pattern are 5×5. The lattice is surrounded by a boundary layer. All light grey edges have conductance a, while every red edge has conductance b.

3
Generating the Discrete Gaussian Free Field

Now that we have seen the theoretical framework around the Gaussian Free Field, we can take a look at how to construct approximations of this field on a finite lattice with a computer. This will be done for the standard DGFF, as well as for the DGFF with arbitrary conductances.

3.1. Sampling a standard GFF with eigenfunctions

First, let us investigate how one can generate a standard Discrete Gaussian Free Field, as defined in section 2.1 using the unaltered graph Laplacian Δ_Λ. Keep in mind that this is a special case of (2.21), with all conductances set to 1. This investigation will make us understand what goes wrong when we want to generate the modified DGFF, as defined in section 2.3. We will take a look at two methods. The first one is proposed by Sheffield to generate a DGFF on an d-dimensional rectangle in $\mathbb{Z}^d$ with periodic boundary conditions (so a d-dimensional torus) [3]. The second method has been described by Chafaï and may be used on a d-dimensional cube in $\mathbb{Z}^d$ with zero boundary conditions (also called Dirichlet boundary conditions) [4]. In essence, both methods apply the same idea. Because all the conductances are equal, we can come up with eigenfunctions for the graph Laplacian, allowing us to diagonalise the covariance matrix corresponding to the DGFF φ.

3.1.1. The Periodic Boundary Case

In his paper, Sheffield considers a two-dimensional $m \times n$ grid as his region Λ. This research focuses on three-dimensional boxes; we will work with the region $\Lambda = \{0, 1, \dots, N-1\}^3$. The boundaries of this $N \times N \times N$ cube are periodic. From the perspective of the lattice point, it seems as if it lies within a lattice surrounded by six copies of itself, one at each face of the cube. Each lattice point has $2d = 6$ nearest-neighbours; two for each dimension. In this section though, we will keep everything one-dimensional for simplicity. However, the idea is easily generalised to higher dimensions.

The method of Sheffield gives an elegant solution to the problem of generating a DGFF, using Discrete Fourier Transforms. Using the Fast Fourier Transform, one can generate instances of a DGFF with great efficiency. Now, recall from Section 2.1.2 that a DGFF is nothing else than a Gaussian field with covariance matrix $\Sigma_\Lambda = \left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1}$. The mean of the DGFF depends on the specified boundary conditions, and can be found via (2.11). We will return to the mean of the DGFF produced by Sheffield in a bit, but suppose for now that it is just equal to 0 everywhere. This does not alter the idea for finding the covariance matrix, which is our main goal for now. Let us take a closer look to the graph Laplacian Δ_Λ. As said earlier, we will restrict ourselves to the one-dimensional case now. The key intuition here is that, for all $k \in \{0, \dots, N-1\}$, the discrete complex exponentials $e_N^k : \mathbb{Z}/N\mathbb{Z} \to \mathbb{C}$ defined by $e_N^k(j) = e^{\frac{2\pi\imath kj}{N}}$ are eigenfunctions of the graph Laplacian in the periodic boundary case. We denote the imaginary unit here by $\imath$, to avoid confusion with our use of i for an arbitrary lattice site in $\mathbb{Z}^d$. The formulation with $\mathbb{Z}/N\mathbb{Z}$ is very natural for these periodic boundaries. For example, consider the lattice site $j = \overline{N-1}$, i.e. $j \equiv N-1 \mod N$, at the right end of the lattice. Note that in 1D the lattice is just a chain of N points. The site to the right of j is the site $\overline{(N-1)+1} = \overline{N} = \overline{0}$, which is the leftmost site of the chain. This is just as we would expect from periodic boundary conditions. Let us now show our claim by observing that for all $j \in \mathbb{Z}/N\mathbb{Z}$ and $k \in \Lambda$:

$$\begin{aligned}
\Delta_\Lambda e_N^k(j) &= -2e_N^k(j) + e_N^k(j+1) + e_N^k(j-1) \\
&= \left(-2 + e^{\frac{2\pi\iota k}{N}} + e^{-\frac{2\pi\iota k}{N}}\right) e^{\frac{2\pi\iota k j}{N}} \\
&= \left(e^{\frac{\pi\iota k}{N}} - e^{-\frac{\pi\iota k}{N}}\right)^2 e^{\frac{2\pi\iota k j}{N}} \\
&= -4\sin^2\left(\frac{\pi k}{N}\right) e_N^k(j).
\end{aligned}$$

Because of our definition on $\mathbb{Z}/N\mathbb{Z}$, the above expressions are well-defined, even for j on the boundary of Λ. As k was left arbitrary, we conclude that e_N^k is an eigenfunction of Δ_Λ for every $k \in \{0, 1, \ldots, N-1\}$, with eigenvalues $\lambda_N^k := -4\sin^2\left(\frac{\pi k}{N}\right)$. Remark that in the general d-dimensional case, the region Λ is the Cartesian product of d lines $\mathbb{Z}/n\mathbb{Z}$, i.e. $(\mathbb{Z}/n\mathbb{Z})^d$. The eigenfunctions on $(\mathbb{Z}/N\mathbb{Z})^d$ will just be the products of the respective eigenfunctions on $\mathbb{Z}/N\mathbb{Z}$. That is, if $e_N^{k_1 \ldots k_d}$ is an eigenfunction on $(\mathbb{Z}/N\mathbb{Z})^d$, it can be written as the product $\prod_{m=1}^d e_N^{k_m}$, where the $e_N^{k_m}$ are the eigenfunction on $\mathbb{Z}/N\mathbb{Z}$ we have defined earlier. This assures that everything works out nicely when going from 1 to multiple dimensions, as we claimed earlier.

The eigenfunctions (i.e. eigenvectors) and eigenvalues of Δ_Λ are now known, so we can write $\Delta_\Lambda = W_N D W_N^{-1}$, with:

$$W_N = N^{-1/2} \begin{pmatrix} e_N^0(0) & e_N^1(0) & \cdots & e_N^{N-1}(0) \\ e_N^0(1) & e_N^1(1) & \cdots & e_N^{N-1}(1) \\ \vdots & \vdots & \ddots & \vdots \\ e_N^0(N-1) & e_N^1(N-1) & \cdots & e_N^{N-1}(N-1) \end{pmatrix}, \tag{3.1}$$

and D the diagonal matrix $D = \operatorname{diag}(\lambda_N^0, \ldots, \lambda_N^{N-1}) = 4 \cdot \operatorname{diag}(\frac{1}{4}\lambda_N^0, \ldots, \frac{1}{4}\lambda_N^{N-1}) =: 4\tilde{D}$. The factor $N^{-1/2}$ in (3.1) and the rewriting of D as $4\tilde{D}$ will be explained later on. As $\Sigma_\Lambda = \left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1}$, we can write for $d = 1$ that $\Sigma_\Lambda = 2W_N(-D)^{-1}W_N^{-1}$. This decomposition will help us to find an easy-to-compute expression for the DGFF φ_Λ. The attentive reader has probably already remarked that since $\lambda_N^0 = 0$, D is singular and we can in fact not compute the inverse of D. Therefore, we use the following convention. From now on, with a slight abuse of notation, we denote by $(-D)^{-1}$ the diagonal matrix $\operatorname{diag}\left(0, \frac{1}{\lambda_N^1}, \ldots, \frac{1}{\lambda_N^{N-1}}\right)$. Also, $(-\tilde{D})^{-1} = \frac{1}{4}(-D)^{-1}$. The consequence of this convention is that we exclude the zero mode e_N^0 from further calculations. Intuitively, this can be understood as follows. The eigenfunction e_N^0 is a constant function, as $e_N^0(j) = e^{\frac{2\pi\iota \cdot 0 \cdot j}{N}} = 1$ for all $j \in \Lambda$. When we consider periodic boundary conditions, we can add any constant value to an already periodic DGFF; this does not affect the periodicity. By excluding the zero mode, we actually fix the DGFF to have a zero average on Λ, as the remaining modes $(e_N^1, \ldots, e_N^{N-1})$ which build up the DGFF all have zero average. For this reason, the method of Sheffield generates a so-called zero average Gaussian Free Field. This is a zero-mean GFF $\varphi = (\varphi_i)_{i\in\Lambda}$ with covariance matrix Σ_Λ that satisfies $\sum_{i\in\Lambda}\varphi_i = 0$ almost surely [19]. In the rest of this Section, we will treat the zero-average DGFF as if it is a Gaussian Field with distribution $\mathcal{N}(0,\Sigma_\Lambda)$, just as we have done so far. As we are predominantly interested in the covariance matrix, this treatment does not alter the computations.

We first give a rough idea of where we are going with this. In general, to sample a Gaussian with covariance matrix C, we define the square root of the matrix X as $A := C^{1/2}$ such that $AA^T = C$. By Theorem A.1.4, if x has the identity I as covariance matrix, then the vector Ax has covariance matrix $AIA^T = C$. We want to do the same thing for $C = \Sigma_\Lambda$ to find a DGFF φ_Λ, making use of the decomposition we have just found. However, this decomposition makes use of complex matrices, which requires some care; A might become complex as well, meaning that Ax becomes a complex vector. This does not make sense as φ_Λ is real. To circumvent this issue, we will let $\varphi_\Lambda = \mathrm{Re}(Az)$, where z is a complex Gaussian vector. Like this, Az will be a complex Gaussian vector, from which we take the real part to get a real φ_Λ.

Let us now make this idea a bit more rigorous. We define $z = x + \iota y$, where $x, y \sim \mathcal{N}\left(0, \frac{1}{2}I_N\right)$ are independent, Gaussian vectors. In other words, $z \simeq \mathcal{N}_c(0, I_N)$ is a complex Gaussian vector. Working with such complex vectors requires a bit of care. The reader is referred to Appendix A.2 for a small discussion on this topic. We know, by Theorem A.2.7, that $Az \simeq \mathcal{N}_c(0, AA^*)$ for any $N \times N$ (possibly complex) matrix A. Here A^* denotes the conjugate transpose (or Hermitian conjugate) of A. Thus, if we can find some A such that $AA^* = 2\Sigma_\Lambda$, we will have $Az \simeq \mathcal{N}_c(0, 2\Sigma_\Lambda)$. By Corollary A.2.9 this results in $\varphi_\Lambda = \mathrm{Re}(Az) \simeq \mathcal{N}(0,\Sigma_\Lambda)$. We claim that $A = W_N(-\tilde{D})^{1/2}$ does the job. This claim is stated in Theorem 3.1.1.

Theorem 3.1.1. Define the matrix $A := W_N(-\tilde{D})^{-1/2}$ and let $z \simeq \mathcal{N}_c(0, I_N)$. Then the random vector $\varphi_\Lambda := \mathrm{Re}(Az)$ is a DGFF on $\Lambda = \{0, \ldots, N-1\}$.

Before we get to the proof however, we need to look at the eigenfunctions e_N^k of Δ_Λ in a bit more detail. It is because of these eigenfunctions that we are able to use Discrete Fourier Transform, as indicated at the beginning of this section. Lemma 3.1.2 is at the basis of that.

Lemma 3.1.2. The functions $\{N^{-1/2}e_N^k\}_{k\in\mathbb{Z}/N\mathbb{Z}}$ form an orthonormal basis for the vector space $\ell^2(\mathbb{Z}/N\mathbb{Z})$.

The vector space $\ell^2(\mathbb{Z}/N\mathbb{Z})$ contains all $f : \mathbb{Z}/N\mathbb{Z} \to \mathbb{C}$ with $||f||_{\ell^2(\mathbb{Z}/N\mathbb{Z})} < \infty$, where $||f||_{\ell^2(\mathbb{Z}/N\mathbb{Z})} := \left(\sum_{j\in\mathbb{Z}/N\mathbb{Z}} |f(j)|^2\right)^{\frac{1}{2}}$ is the norm of this space. Furthermore, $\ell^2(\mathbb{Z}/N\mathbb{Z})$ is a Hilbert space, with inner product $\langle f, g\rangle := \sum_{j\in\mathbb{Z}/N\mathbb{Z}} f(j)\overline{g(j)}$ for all $f, g \in \ell^2(\mathbb{Z}/N\mathbb{Z})$.

Proof of Lemma 3.1.2. First of all, $\ell^2(\mathbb{Z}/N\mathbb{Z})$ is an N-dimensional vector space and $\{N^{-1/2}e_N^k\}_{k\in\mathbb{Z}/N\mathbb{Z}}$ is a collection of N functions, so it remains to show that this collection $\{N^{-1/2}e_N^k\}_{k\in\mathbb{Z}/N\mathbb{Z}}$ is an orthonormal set. However, the functions $\{e_N^k\}_{k\in\mathbb{Z}/N\mathbb{Z}}$ are all eigenfunctions, corresponding to distinct eigenvalues, of Δ_Λ. For symmetric matrices with distinct eigenvalues, it is known that the eigenfunctions are orthogonal. The factor $N^{-1/2}$ turns the collection of complex exponentials into an orthonormal set, since:

$$\langle e_N^k, e_N^k\rangle = \sum_{j\in\mathbb{Z}/N\mathbb{Z}} N^{-1/2}e_N^k(j)\overline{N^{-1/2}e_N^k(j)} = \frac{1}{N}\sum_{j\in\mathbb{Z}/N\mathbb{Z}} e^{\frac{2\pi\iota kj}{N}}e^{-\frac{2\pi\iota kj}{N}} = \frac{N}{N} = 1,$$

for any $k\in\mathbb{Z}/N\mathbb{Z}$. We conclude that $\{N^{-1/2}e_N^k\}_{k\in\mathbb{Z}/N\mathbb{Z}}$ forms an orthonormal basis for $\ell^2(\mathbb{Z}/N\mathbb{Z})$. □

Consequently, Theorem 7.4 in [20] tells us that for any function $f\in\ell^2(\mathbb{Z}/N\mathbb{Z})$, we can write $f = \sum_{k\in\mathbb{Z}/N\mathbb{Z}} N^{-1/2}\langle f, N^{-1/2}e_N^k\rangle e_N^k =: \frac{1}{\sqrt{N}}\sum_{k\in\mathbb{Z}/N\mathbb{Z}}\hat{f}(k)e_N^k$. Here we defined the Discrete Fourier Transform of the function f as $\hat{f}:\mathbb{Z}/N\mathbb{Z}\to\mathbb{C}$ with $\hat{f}(k) = \langle f, N^{-1/2}e_N^k\rangle = \frac{1}{\sqrt{N}}\sum_{j\in\mathbb{Z}/N\mathbb{Z}} f(j)e^{-\frac{2\pi\iota kj}{N}}$. At the same time, we define the inverse Discrete Fourier Transform of $\hat{f}(k)$ as the function $f:\mathbb{Z}/N\mathbb{Z}\to\mathbb{C}$ with $f(j) = \frac{1}{\sqrt{N}}\sum_{k\in\mathbb{Z}/N\mathbb{Z}}\hat{f}(k)e^{\frac{2\pi\iota jk}{N}} = \langle\hat{f}, N^{-1/2}e_N^{-k}\rangle$. We use a factor $\frac{1}{\sqrt{N}}$ up front while [20] uses a factor $\frac{1}{N}$. This is just a question of definition. With this convention, it is easy to check that we can rewrite the DFT as $\hat{f} = W_N^* f = \big(\langle f, N^{-1/2}e_N^k\rangle\big)_{k\in\mathbb{Z}/N\mathbb{Z}}$, where W_N is as in (3.1). Similarly, the inverse DFT can be written as $f = W_N\hat{f}$.

The matrix W_N has one very important property. It is unitary, since the columns of the matrix are the functions $N^{-1/2}e_N^k$. The latter were orthonormal according to Lemma 3.1.2. Note that this is why we defined W_N with the $N^{-1/2}$ in front. A direct consequence of W_N being unitary is that $W_N^* = W_N^{-1}$. It is trivially seen that W_N^* must also be unitary. We are now capable of proving Theorem 3.1.1.

Proof of Theorem 3.1.1. First of all, we show that AA^* is equal to $2\Sigma_\Lambda$:

$$\begin{aligned}AA^* &= W_N(-\tilde{D})^{-1/2}\big(W_N(-\tilde{D})^{-1/2}\big)^*\\ &= W_N(-\tilde{D})^{-1/2}\big(-\tilde{D}^{-1/2}\big)^T W_N^*\\ &= W_N(-\tilde{D})^{-1}W_N^{-1}\\ &= 4W_N(-4\tilde{D})^{-1}W_N^* = 2\Sigma_\Lambda,\end{aligned}$$

Here we used that $(-D)^{1/2}$ is real at the second equality and that W_N is unitary at the third equality. In the last equality we use that $D = 4\tilde{D}$ and the diagonalisation $\Sigma_\Lambda = 2W_N(-D)^{-1}W_N^{-1}$. Theorem A.2.7 implies that $Az \simeq \mathcal{N}_c(0, 2\Sigma_\Lambda)$. By Corollary A.2.9, $\varphi_\Lambda := \mathrm{Re}(Az) = x \simeq \mathcal{N}(0,\Sigma_\Lambda)$ so we are done. □

Theorem 3.1.1 gives us a very elegant manner to generate a DGFF with periodic boundary conditions. Using the Fast Fourier Transform, it is very fast to work out the inverse DFT of $(-D)^{1/2}z$. Performing some basic arithmetic and taking the real part then immediately gives a DGFF on Λ. The implementation of this method can be found on the Github page of this project. A link to the repository can be found in Appendix B.

3.1.2. The Zero Boundary Case

The method of Chafaï is similar to the method of Sheffield described in the previous section. However, Chafaï works with a slightly different region, namely $\Lambda = \{1,\dots,N-1\}^d$. Once again, we take $d = 1$ for simplicity. This choice of region is made such that for the field φ we have the zero boundary conditions $\varphi(0) = \varphi(N) = 0$. This will give more elegant expressions later on.

For this region, the discrete Laplacian Δ_Λ has eigenfunctions $s_N^k(\cdot) := \sqrt{\frac{2}{N}}\sin\left(\frac{\pi k \cdot}{N}\right)$ with eigenvalues $\nu_N^k := -4\sin^2\left(\frac{\pi k}{2N}\right)$. Indeed, for all $j \in \Lambda$ and $k \in \{1,\dots,N-1\}$ we find that:

$$\begin{aligned}
\Delta_\Lambda s_N^k(j) &= -2s_N^k(j) + s_N^k(j+1) + s_N^k(j-1)\\
&= \sqrt{\frac{2}{N}}\left[-2\frac{e^{\frac{\pi\iota kj}{N}} - e^{-\frac{\pi\iota kj}{N}}}{2\iota} + \frac{e^{\frac{\pi\iota k(j+1)}{N}} - e^{-\frac{\pi\iota k(j+1)}{N}}}{2\iota} + \frac{e^{\frac{\pi\iota k(j-1)}{N}} - e^{-\frac{\pi\iota k(j-1)}{N}}}{2\iota}\right]\\
&= \sqrt{\frac{2}{N}}\left[\frac{e^{\frac{\pi\iota kj}{N}}}{2\iota}\left(-2 + e^{\frac{\pi\iota k}{N}} + e^{-\frac{\pi\iota k}{N}}\right) - \frac{e^{\frac{-\pi\iota kj}{N}}}{2\iota}\left(-2 + e^{\frac{\pi\iota k}{N}} + e^{-\frac{\pi\iota k}{N}}\right)\right]\\
&= \sqrt{\frac{2}{N}}\frac{e^{\frac{\pi\iota kj}{N}} - e^{-\frac{\pi\iota kj}{N}}}{2\iota}\left(e^{\frac{\pi\iota k}{2N}} - e^{-\frac{\pi\iota k}{2N}}\right)^2\\
&= \sqrt{\frac{2}{N}}\sin\left(\frac{\pi kj}{N}\right)\left(-4\sin^2\left(\frac{\pi k}{2N}\right)\right) = \nu_N^k s_N^k.
\end{aligned}$$

Notice as well that the eigenfunctions are zero for $j = 0$ and $j = N$, in compliance with the zero boundary conditions. Furthermore, the factor $\sqrt{\frac{2}{N}}$ in the definition of s_N^k assures that the eigenfunctions have norm 1.

Indeed, for $k \in \{1, \dots, N-1\}$ we have:

$$\begin{aligned}
\langle s_N^k, s_N^k \rangle &= \frac{2}{N} \sum_{j=1}^{N-1} \sin^2\left(\frac{\pi k j}{N}\right) \\
&= \frac{2}{N} \sum_{j=1}^{N-1} \frac{1-\cos\left(\frac{2\pi k j}{N}\right)}{2} \\
&= \frac{2}{N}\left[\frac{N-1}{2} - \frac{1}{4}\sum_{j=1}^{N-1}\left(e^{\frac{2\pi\iota k}{N}}\right)^j - \frac{1}{4}\sum_{j=1}^{N-1}\left(e^{-\frac{2\pi\iota k}{N}}\right)^j\right] \\
&= \frac{2}{N}\left[\frac{N-1}{2} - \frac{1}{4}\left(\frac{1-e^{\frac{2\pi\iota kN}{N}}}{1-e^{\frac{2\pi\iota k}{N}}} - 1\right) - \frac{1}{4}\left(\frac{1-e^{-\frac{2\pi\iota kN}{N}}}{1-e^{-\frac{2\pi\iota k}{N}}}\right) - 1\right] = 1.
\end{aligned}$$

Now, we claim that we may write the (m,n)-th entry of the Laplacian, i.e. $(\Delta_\Lambda)_{mn}$ as:

$$(\Delta_\Lambda)_{mn} = \sum_{k\in\Lambda} v_N^k s_N^k(m) s_N^k(n) \tag{3.2}$$

Let us quickly show this claim. With the eigenfunctions (viz. eigenvectors) and eigenvalues known, we can write Δ_Λ as PDP^{-1}, where

$$P = \begin{pmatrix} s_N^1(1) & s_N^2(1) & \cdots & s_N^{N-1}(1) \\ s_N^1(2) & s_N^2(2) & \cdots & s_N^{N-1}(2) \\ \vdots & \vdots & \ddots & \vdots \\ s_N^1(N-1) & s_N^2(N-1) & \cdots & s_N^{N-1}(N-1) \end{pmatrix}, \quad D = \begin{pmatrix} v_N^1 & 0 & \cdots & 0 \\ 0 & v_N^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_N^{N-1} \end{pmatrix}.$$

Notice that P is real and unitary, as its columns are the eigenvectors s_N^k of Δ_Λ. Since the s_N^k are the eigenvectors of a symmetric matrix, they are orthogonal. We have already shown these eigenvectors have norm 1. Therefore, $P^{-1} = P^* = P^T$. Now, denote by $\mathbb{1}_N^m$ the vector that is everywhere 0, except at the m-th entry. Then the matrix $\mathbb{1}_N^m\left(\mathbb{1}_N^n\right)^T$ is everywhere zero, except at the (m,n)-th entry. Consequently, we can write the matrix D as $\sum_{k\in\Lambda} v_N^k \mathbb{1}_N^k\left(\mathbb{1}_N^k\right)^T$. This in turn means that:

$$\begin{aligned}
\Delta_\Lambda &= PDP^T \\
&= P\left(\sum_{k\in\Lambda} v_N^k \mathbb{1}_N^k\left(\mathbb{1}_N^k\right)^T\right)P^T \\
&= \sum_{k\in\Lambda} v_N^k P\mathbb{1}_N^k\left(\mathbb{1}_N^k\right)^T P^T \\
&= \sum_{k\in\Lambda} v_N^k \left(P\mathbb{1}_N^k\right)\left(P\mathbb{1}_N^k\right)^T \\
&= \sum_{k\in\Lambda} v_N^k s_N^k \left(s_N^k\right)^T,
\end{aligned}$$

where in the last equality we used that $P\mathbb{1}_N^k$ is just the k-th column of P, i.e. s_N^k. Finally, we have an expression that resembles (3.2), except that the former is for the entire matrix. To retrieve the (m,n)-th element, we can compute $\left(\mathbb{1}_N^m\right)^T \Delta_\Lambda \mathbb{1}_N^n$, which equals $\left(\mathbb{1}_N^m\right)^T \left(\sum_{k\in\Lambda} \nu_N^k s_N^k \left(s_N^k\right)^T\right)\mathbb{1}_N^n = \sum_{k\in\Lambda} \nu_N^k \left(\left(\mathbb{1}_N^m\right)^T s_N^k\right)\left(\left(\mathbb{1}_N^n\right)^T s_N^k\right)^T$. The latter is equal to 3.2.

With this claim now proven, we are able to give an explicit expression for a Discrete Gaussian Free Field on Λ. Recall that a DGFF is a Gaussian field determined by the covariance matrix $\Sigma_\Lambda = \left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1}$. Because of the zero boundary condition, we know that the mean of this Gaussian field is also zero. Just as we did for Sheffield, we can create a Gaussian vector distributed with mean 0 and covariance matrix Σ_Λ from a standard normal random vector $z = (z_k)_{k\in\Lambda} \simeq \mathcal{N}(0,I)$. Indeed, $\Sigma_\Lambda^{1/2} z \simeq \mathcal{N}\left(0, \Sigma_\Lambda^{1/2} I \left(\Sigma_\Lambda^{1/2}\right)^T\right)$, so $\Sigma_\Lambda^{1/2} z \simeq \mathcal{N}(0,\Sigma_\Lambda)$. Using (3.2), we can find an explicit expression for $\Sigma_\Lambda^{1/2} = \left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1/2}$. Realise that $\left(-\frac{1}{2d}\Delta_\Lambda\right)^{-1/2}$ has the same eigenvectors as Δ_Λ, but with eigenvalues $\left(-\frac{1}{2d}\nu_N^k\right)^{-1/2}$. Thus, for arbitrary $m,n \in \Lambda$:

$$\left(\Sigma_\Lambda^{1/2}\right)_{mn} = \sum_{k\in\Lambda} \left(-\frac{1}{2d}\nu_N^k\right)^{-1/2} s_N^k(m) s_N^k(n). \tag{3.3}$$

Consequently, the random vector $\varphi := \Sigma_\Lambda^{1/2} z$ is a Gaussian Free Field on Λ and is given by Equation (3.4):

$$\varphi = \left(\sum_{n\in\Lambda} \left(\Sigma_\Lambda^{1/2}\right)_{mn} z_n\right)_{m\in\Lambda} = \left(\sum_{n\in\Lambda}\sum_{k\in\Lambda} \left(-\frac{1}{2d}\nu_N^k\right)^{-1/2} s_N^k(m) s_N^k(n) z_n\right)_{m\in\Lambda}. \tag{3.4}$$

This is a very useful expression; once the matrix elements $\left(\Sigma_\Lambda^{1/2}\right)_{mn}$ have been computed for all $m,n\in\Lambda$, we can generate as many realisations of a DGFF on Λ as we want by performing the matrix-vector multiplication in Equation (3.4) with every time a newly sampled $z \simeq \mathcal{N}(0,I)$.

3.2. Adding Random Conductances: CG Sampling

By adding randomness to the conductances, the method described in the previous section will fail; we do not have a ‘nice’ eigenfunction decomposition anymore. Therefore, we resort to another method, the Conjugate Gradient (CG) sampler, to generate a realisation of the modified DGFF. Due to the method’s flexibility, we can try out every set of conductances we could think of.

The method of Conjugate Gradients (CG) is a well-known method for solving large linear systems. However, with a small addition, it can be turned into an efficient way to generate approximate samples from high-dimensional Gaussian distributions [8]. This can be seen in the following way. Consider a multivariate Gaussian distribution $\mathcal{N}(\mu,\Sigma) = \mathcal{N}\left(A^{-1}c, A^{-1}\right)$. Here we introduced the precision matrix $A := \Sigma^{-1}$ and the so-called potential vector $c := A\mu$. Since A^{-1} is the covariance matrix, it must be symmetric and positive definite. One can easily see this must also be the case for A. The corresponding density function scales with $e^{-\phi(x)}$, where we define $\phi(x) := \frac{1}{2}x^T Ax - c^T x$. Indeed, as A is symmetric:

$$\begin{aligned}\frac{1}{2}(x - A^{-1}c)^T A(x - A^{-1}c) &= \frac{1}{2}\left[x^T Ax - x^T A(A^{-1}c) - (A^{-1}c)^T Ax + (A^{-1}c)^T A(A^{-1}c)\right] \\ &= \frac{1}{2}x^T Ax - \frac{1}{2}x^T c - \frac{1}{2}c^T x + \frac{1}{2}c^T A^{-1}c \\ &= \frac{1}{2}x^T Ax - c^T x + (\text{term independent of } x).\end{aligned}$$

Taking the exponent of the opposite of both sides, $e^{-\frac{1}{2}(x-A^{-1}c)^T A(x-A^{-1}c)}$ becomes the Gaussian density and the final expression becomes $Ke^{-\phi(x)}$, with K a scaling constant.

The values of x that lead to a small $\phi(x)$, lead to a larger $e^{-\phi(x)}$ and so are more likely to be drawn as a sample from $\mathcal{N}\left(A^{-1}c, A^{-1}\right)$. In other words, sampling from $\mathcal{N}\left(A^{-1}c, A^{-1}\right)$ is equivalent to minimising $\phi(x) = \frac{1}{2}x^T Ax - c^T x$. Recall that $\phi(x)$ has a critical point at $x = x'$ whenever $\nabla\phi(x') = 0$. As A is symmetric, one can easily show that $\nabla\phi(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - c = Ax - c$. Note that A is equal to the Hessian matrix $H_\phi(x)$ of $\phi(x)$. Since A is positive definite, we know that the critical point x' for which $Ax' - c = 0$, is a unique minimum. In conclusion, minimising $\phi(x)$ is equivalent to solving the linear system $Ax = c$.

The question now is how to solve $Ax = c$. Bear in mind that this is a very large system. Methods such as inverting A to compute $x = A^{-1}c$ are infeasible. On one hand the computation time will be too large, on the other hand a huge amount of memory will be needed to store the non-sparse inverse matrix A^{-1}. However, working out the matrix-vector products Av is possible, as A itself is sparse. We will exploit this observation with the CG method.

3.2.1. The CG Linear Solver Algorithm

Let us dive deeper into the CG method itself. We will first state the algorithm, as proposed in [8], to sample from $\mathcal{N}(0, A^{-1})$ in Algorithm 1. Then we will go over each step separately, partially following [21]. As input, the algorithm needs a vector x^0, that acts as an initial guess for the solution x of $Ax = b$ with $b \simeq \mathcal{N}(0, I_n)$. Also a routine is needed to work out the matrix-vector products Av for any vector v, where A is the precision matrix of the distribution we want to sample from. Finally, one gives a stopping tolerance, which will be explained later on.

The standard CG algorithm will approximate the solution x of the system $Ax = b$. In the initialisation step, the residual $r^0 = b - Ax^0$ is computed. The residuals in this algorithm can be interpreted as some kind of error. Intuitively, if an approximation x^k is close in some sense to x, then Ax^k should be close to Ax. Thus, a good approximation yields a small residual $Ax - Ax^k = b - Ax^k =: r^k$. When ini-

Algorithm 1: Conjugate Gradient Sampler from $\mathcal{N}(0, A^{-1})$.

```
Input: an $n \times 1$ vector $x^0$, an $n \times n$ symmetric positive definite matrix $A$ and
       a stopping tolerance $\varepsilon$.
Output: a(n approximate) solution $x^k$ to $Ax = b$ with $b$ a standard
        Gaussian vector $b \simeq \mathcal{N}(0, I_n)$ and a(n approximate) sample $y^k$
        from $\mathcal{N}(0, A^{-1})$.
let $b \simeq \mathcal{N}(0, I_n)$, $r^0 = b - Ax^0$, $p^0 = r^0$, $d_0 = \left(p^0\right)^T Ap^0$, $y^0 = x^0$ and $k := 1$;
while $||r^k||_2 \geq \varepsilon$ do
    1. compute the 1-D minimiser of $\phi$ in the direction $x^{k-1} + \gamma p^{k-1}$, i.e.
       $\gamma_{k-1} = \frac{\left(r^{k-1}\right)^T r^{k-1}}{d_{k-1}}$;
    2. let $x^k = x^{k-1} + \gamma_{k-1} p^{k-1}$;
    3. sample $z \simeq \mathcal{N}(0,1)$, and set $y^k = y^{k-1} + \frac{z}{\sqrt{d_{k-1}}} p^{k-1}$;
    4. compute the residual $r^k = -\nabla\phi(x^k) = r^{k-1} - \gamma_{k-1} Ap^{k-1}$;
    5. let $\beta_k = -\frac{\left(r^k\right)^T r^k}{\left(r^{k-1}\right)^T r^{k-1}}$, the new conjugate search direction is then given
       by $p^k = r^k - \beta_k p^{k-1}$;
    6. compute $d_k = \left(p^k\right)^T Ap^k$;
    7. set $k := k + 1$;
end
```

tialising, we also compute p^0. The vector p^k is the so-called search direction in the k-th iteration. The CG method is an iterative method; at the k-th iteration a new approximation x^k is found. This x^k is found from the previous approximation, x^{k-1}, by starting at x^{k-1} and 'walking' a certain distance in the direction of p^{k-1}. In other words, we are searching a better approximation for x in the direction of p^{k-1}. Determining the search direction for the next iteration depends on the iterative method chosen. As we will see, CG makes use of the already used search directions to find the newest one. However, to find p^0, there are no such previous search direction. Our initial guess is therefore the direction of steepest decent. Thus, $p^0 = -\nabla\phi(x^0) = -(Ax^0 - b) = b - Ax^0 = r^0$. Finally, we also set an intermediate quantity $d_0 = \left(p^0\right)^T Ap^0$, our first approximate sample from $\mathcal{N}(0, A^{-1})$ equal to $y^0 = x^0$ and the iteration counter $k = 1$.

Now let us move on to part of the algorithm that is repeated until we have reached, in some sense, a certain level of precision. We finish the algorithm at some iteration k once the norm of the residual at that iteration, $||r^k||_2$, dives under the given stopping threshold ε. As the residual indicates in a way that we are close to the exact solution x, it lends itself very well to be used in a stopping criterion.

Every time we enter the loop, we have an approximation x^{k-1} and a search direction p^{k-1}. First thing to do is to find out how far we must go in the direction of p^{k-1} to find a new, better approximation $x^k = x^{k-1} + \gamma p^{k-1}$. Here γ can be seen as the distance we need to go along x^{k-1}. To determine this γ this, recall that the CG method minimises $\phi(x)$. Thus we want to find γ such that $\phi(x^k)$ is as small as possible, i.e. minimise $\phi\big(x^{k-1} + \gamma p^{k-1}\big)$. We claim that this γ equals $\gamma_{k-1} := \frac{(p^{k-1})^T r^{k-1}}{(p^{k-1})^T A p^{k-1}}$. To show this claim, notice that:

$$\begin{aligned}
\phi(x^k) &= \phi(x^{k-1} + \gamma_{k-1} p^{k-1}) \\
&= \frac{1}{2}\Big(x^{k-1} + \gamma_{k-1} p^{k-1}\Big)^T A \Big(x^{k-1} + \gamma_{k-1} p^{k-1}\Big) - b^T \Big(x^{k-1} + \gamma_{k-1} p^{k-1}\Big) \\
&= \frac{1}{2}\gamma_{k-1}^2 \Big(p^{k-1}\Big)^T A p^{k-1} + \gamma_{k-1} \Big(p^{k-1}\Big)^T \Big(A x^{k-1} - b\Big) + (\text{terms independent of } \gamma_{k-1})
\end{aligned}$$

We may now differentiate the above expression with respect to γ^{k-1} and set that expression equal to zero to find the γ_{k-1} minimising $\phi(x^k)$.

$$\gamma_{k-1} = -\frac{\big(p^{k-1}\big)^T \big(A x^{k-1} - b\big)}{\big(p^{k-1}\big)^T A p^{k-1}} = \frac{\big(p^{k-1}\big)^T r^{k-1}}{\big(p^{k-1}\big)^T A p^{k-1}} \tag{3.5}$$

Note that this is indeed a minimum, as $\frac{\mathrm{d}^2 \phi(x^k)}{\mathrm{d}(\gamma^{k-1})^2} < 0$. This is not yet the expression for γ_{k-1} given in Algorithm 1. At the end of this discussion on the CG method, we will be able to rewrite γ_{k-1} to the expression used in the algorithm.

Apart from that, we have covered steps 1 and 2 of the algorithm. Let us skip step 3 for now. This step is added by Parker and Fox to turn the classic CG linear solver into a CG normal sampler. We will first cover the rest of the linear solver part of the algorithm and come back to Step 3 later.

Step 4 of the method computes the residual of the new approximation x^k we found in step 2. Notice that:

$$r^k = b - A x^k = b - A\Big(x^{k-1} + \gamma_{k-1} p^{k-1}\Big) = b - A x^{k-1} - \gamma_{k-1} A p^{k-1} = r^{k-1} - \gamma_{k-1} A p^{k-1}$$

With this new residual, we may now determine the search direction for the next iteration. This is where the CG solver differs from other iterative methods, such as the method of Steepest Decent. The latter takes $p^k = r^k = -\nabla\phi(x^k)$, i.e. the direction of maximal descent. CG requires that different search directions are conjugate (also called A-conjugate or A-orthogonal). That is, $\big(p^j\big)^T A p^k = 0$ for $j \neq k$. One can look at conjugate vectors as orthogonal vectors, not with respect to the standard inner product $\langle u, v\rangle := u^T v$, but with respect to the inner product $\langle u, v\rangle_A := u^T A v$. To achieve this requirement, we can perform a Gram-Schmidt inspired calculation:

$$p^k = r^k - \sum_{j<k} \frac{\big(p^j\big)^T A r^k}{\big(p^j\big)^T A p^j} p^j. \tag{3.6}$$

In a way, we adjust r^k to make it conjugate to the previous search directions by removing the components of r^k linked to these previous search directions. Observe that the expression (3.6) indeed yields a new search direction p^k, conjugate to all the previous ones $(p^m)_{m<k}$. This is shown below, under the inductive assumption that the $(p^m)_{m<k}$ are already mutually conjugate.

$$\begin{aligned}
\left(p^m\right)^T A p^k &\overset{(3.6)}{=} \left(p^m\right)^T A\left(r^k - \sum_{j<k} \frac{\left(p^j\right)^T A r^k}{\left(p^j\right)^T A p^j} p^j\right) \\
&= \left(p^m\right)^T A r^k - \sum_{j<k} \frac{\left(p^j\right)^T A r^k}{\left(p^j\right)^T A p^j} \left(p^m\right)^T A p^j \\
&= \left(p^m\right)^T A r^k - \frac{\left(p^m\right)^T A r^k}{\left(p^m\right)^T A p^m} \left(p^m\right)^T A p^m \\
&= \left(p^m\right)^T A r^k - \left(p^m\right)^T A r^k = 0
\end{aligned}$$

However, we now have a slight problem. In order to compute p^k, we need to store all the previous search directions. This may quickly become infeasible when working in large dimensions. There is, however, another way to write down p^k. To this end, let us introduce the concept of a Krylov subspace.

Definition 3.2.1. Let $A \in \mathbb{R}^{n\times n}$ and $y \in \mathbb{R}^n$. We define the Krylov subspace of dimension k (also called k-th Krylov subspace) by

$$\mathcal{K}_k(A; y) := \operatorname{span}\left\{y, Ay, \ldots, A^{k-1} y\right\} \tag{3.7}$$

When applied in the context of Conjugate Gradients, where A is symmetric and positive definite, this Krylov subspace has several nice properties.

Lemma 3.2.2. Let $k \geq 0$. Then the following are true:

(i) The residual r^k is in the Krylov subspace $\mathcal{K}_{k+1}(A; r^0)$.

(ii) If $r^k \neq 0$, $\{p^0, \ldots, p^k\}$ is a basis for $\mathcal{K}_{k+1}(A; r^0)$.

(iii) We have $x^k - x^0 \in \mathcal{K}_k(A; r^0)$.

Proof of Lemma 3.2.2. (i) Observe that $A \cdot \mathcal{K}_k(A; r^0) = \operatorname{span}\{Ar^0, A^2 r^0, \ldots, A^k r^0\} \subset \operatorname{span}\{r^0, Ar^0, A^2 r^0, \ldots, A^k r^0\} = \mathcal{K}_{k+1}(A; r^0)$. Let us prove $r^k \in \mathcal{K}_{k+1}(A; r^0)$ by induction. Clearly, $r^0 \in \mathcal{K}_1(A; r^0) := \operatorname{span}\{r^0\}$, so now suppose that $r^n \in \mathcal{K}_{n+1}(A; r^0)$. Then $r^n \in \mathcal{K}_{n+2}(A; r^0)$ too. By step 4 of Algorithm 1, $r^{n+1} = r^n - \gamma_n A p^n$. We claim that as $r^n \in \mathcal{K}_{n+1}(A; r^0)$, $p^n \in \mathcal{K}_{n+1}(A; r^0)$. Then by our first observation, $Ap^n \in \mathcal{K}_{n+2}(A; r^0)$ and so r^{n+1} is a linear combination of elements of $\mathcal{K}_{n+2}(A; r^0)$. Thus $r^{n+1} \in \mathcal{K}_{n+2}(A; r^0)$. We finish the first part of this proof by justifying our claim. This is in fact done with a strong induction argument. Recall from Algorithm 1 that $p^0 = r^0$, so clearly $p^0 \in \mathcal{K}_1(A; r^0)$. Now suppose $p^j \in \mathcal{K}_{j+1}(A; r^0)$ for all $1 \leq j \leq m$ and $r^{m+1} \in \mathcal{K}_{m+2}(A; r^0)$. We want to show $p^{m+1} \in \mathcal{K}_{m+2}(A; r^0)$. Using (3.6), we may rewrite p^{m+1} as a linear combination of r^{m+1} and all the p^j with $1 \leq j < m+1$. Consequently, $p^{m+1} \in \mathcal{K}_{m+2}(A; r^0)$. By strong induction, this proves our claim.

(ii) Let us now show that $\{p^0, \dots, p^k\}$ is a basis for $\mathcal{K}_{k+1}(A; r^0)$, whenever $r^k \neq 0$. By *(i)*, and by our earlier claim, we know that $\{p^0, \dots, p^k\} \subset \mathcal{K}_{k+1}(A; r^0)$. Since $\mathcal{K}_{k+1}(A; r^0)$ is spanned by $k+1$ vectors and since $\{p^0, \dots, p^k\}$ is a collection of $k+1$ vectors, it suffices to show that $\{p^0, \dots, p^k\}$ is a linearly independent set of vectors. This can be shown by contradiction. Recall that the $\{p^i\}_{i=0}^{n-1}$ are A-conjugate, i.e. $\left(p^i\right)^T A p^j = 0$ for $i \neq j$. Also realise that since A is positive definite, $\left(p^i\right)^T A p^j > 0$ for $i = j$. Since $r^k \neq 0$, we have by (3.6) that p^k is not, already by definition, a linear combination of $\{p^i\}_{i=0}^{k-1}$. However, suppose that there does exist some $m \in \{0, \dots, k\}$ such that p^m can be written as a linear combination $\sum_{i=0, i\neq m}^{n-1} \alpha_i p^i$ of the remaining search directions. We assume here, without loss of generality, that the α_i's are all nonzero. Then we find, for any $j \in \{0, \dots, k-1\}$ unequal to m, that:

$$\left(p^j\right)^T A p^m = \left(p^j\right)^T A \left(\sum_{i=0, i \neq m}^{n-1} \alpha_i p^i \right) = \left(p^j\right)^T A \left(\alpha_j p^j\right) \neq 0.$$

This would imply however that p^j and p^m are not A-conjugate; a contradiction. Therefore $\{p^0, \dots, p^k\}$ is a set of $k+1$ linearly independent vectors in $\mathcal{K}_{k+1}(A; r^0)$. We conclude that if $r^k \neq 0$, $\{p^0, \dots, p^k\}$ is a basis of $\mathcal{K}_{k+1}(A; r^0)$.

(iii) Finally, we show that $x^k - x^0 \in \mathcal{K}_k(A; r^0)$. It turns out this follows immediately from *(i)*. By Step 2 of Algorithm 1, we can rewrite:

$$\begin{aligned} x^k - x^0 &= x^{k-1} + \gamma_{k-1} p^{k-1} - x^0 \\ &= x^{k-2} + \gamma_{k-2} p^{k-2} + \gamma_{k-1} p^{k-1} - x^0 \\ &= \dots = x^0 + \sum_{j=0}^{k-1} \gamma_j p^j - x^0 = \sum_{j=0}^{k-1} \gamma_j p^j \end{aligned}$$

The latter is a linear combination of $\{p^j\}_{0 \leq j \leq k-1}$ and so $x^k - x^0 \in \mathcal{K}_k(A; r^0)$. This finishes the proof. □

Part *(iii)* of Lemma 3.2.2 allows us to look at the CG method in a new way. At each iteration k, we find a new approximation x^k for x. This approximation satisfies $\phi(x^k) \leq \phi(y)$ for all $y \in x^0 + \mathcal{K}_k(A; r^0) := \{y \in \mathbb{R}^n | \exists z \in \mathcal{K}_k(A; r^0) : y = x^0 + z\}$. That is, at iteration k we minimise $\phi(x)$ over $x^0 + \mathcal{K}_k(A; r^0)$. This realisation has an important consequence, which is stated in Lemma 3.2.3.

Lemma 3.2.3. If $r^k \neq 0$, then r^k is *(i)* orthogonal to $\mathcal{K}_k(A; r^0)$ and *(ii)* A-orthogonal to $\mathcal{K}_{k-1}(A; r^0)$.

Proof of Lemma 3.2.3. *(i)* As we pointed out earlier, at iteration k we find the minimiser x^k of $\phi(x)$ over $x^0 + \mathscr{K}_k(A; r^0)$. We claim that because of this, $\nabla\phi(x^k)$ is orthogonal to $\mathscr{K}_k(A; r^0)$. That is, $\left\langle \nabla\phi(x^k), z\right\rangle = 0$ for all $z \in \mathscr{K}_k(A; r^0)$. Intuitively, one can see x^k as the best approximation of x in the subspace $\mathscr{K}_k(A; r^0)$. The claim says that this best approximation satisfies the 'orthogonality condition' $\nabla\phi(x^k) \perp \mathscr{K}_k(A; r^0)$.

To show our claim, note that as $\mathscr{K}_k(A; r^0) = \operatorname{span}\{p^0, \ldots, p^{k-1}\}$ according to Lemma 3.2.2*(ii)*, it suffices to show that for all $i \in \{0, \ldots, k-1\}$, $\left\langle \nabla\phi(x^k), p^i\right\rangle = 0$. First of all, we perform a similar calculation to the one in the proof of part *(iii)* of Lemma 3.2.2:

$$x^k = x^{k-1} + \gamma_{k-1} p^{k-1} = \ldots = x^m + \sum_{j=m}^{k-1} \gamma_j p^j,$$

where $m \in \{0, \ldots, k-1\}$. We can use this with $m = i$ to compute $\left\langle \nabla\phi(x^k), p^i\right\rangle$:

$$\begin{aligned}
\left\langle \nabla\phi(x^k), p^i\right\rangle &= \left\langle Ax^k - b, p^i\right\rangle \\
&= \left\langle Ax^i + \sum_{j=i}^{k-1} \gamma_j A p^j - b, p^i\right\rangle \\
&= \left\langle Ax^i - b, p^i\right\rangle + \sum_{j=i}^{k-1} \gamma_j \left\langle A p^j, p^i\right\rangle \\
&\overset{(a)}{=} -\langle r^i, p^i\rangle + \gamma_i \left\langle p^i, A p^i\right\rangle \\
&\overset{(3.5)}{=} -\langle r^i, p^i\rangle + \frac{\left\langle r^i, p^i\right\rangle}{\left\langle p^i, A p^i\right\rangle} \left\langle p^i, A p^i\right\rangle = 0.
\end{aligned}$$

We used that the p^j's are A-conjugate at (a). We conclude that $\nabla\phi(x^k)$ is orthogonal to $\mathscr{K}_k(A; r^0)$. Note however that $r^k = -\nabla\phi(x^k)$, so we have shown that r^k is orthogonal to $\mathscr{K}_k(A; r^0)$, finishing the first part of this proof.

(ii) If we consider an arbitrary $z \in \mathscr{K}_{k-1}(A; r^0)$, then it is easily seen that $Az \in \mathscr{K}_k(A; r^0)$. By part *(i)*, r^k is orthogonal to $\mathscr{K}_k(A; r^0)$, in particular, r^k is orthogonal to Az. In other words, $\langle r^k, z\rangle_A := \left(r^k\right)^T Az = 0$, so r^k is A-orthogonal to $\mathscr{K}_{k-1}(A; r^0)$. □

Lemma 3.2.3 tells us a lot. First of all, part *(i)* shows that the residual r^k is orthogonal to all previous search directions $p^0, \ldots, p^{k-1}$. Not only that, r^k is also orthogonal to all the previous residuals $r^0, \ldots, r^{k-1}$. This is because of (3.6). With this relation, we can reduce each p^j with $j < k$ to a linear combination of $r^0, \ldots, r^{k-1}$. In other words, $\operatorname{span}\{r^0, \ldots, r^{k-1}\} = \operatorname{span}\{p^0, \ldots, p^{k-1}\} = \mathscr{K}_k(A; r^0)$. Similarly, as r^k is A-orthogonal to $\mathscr{K}_k(A; r^0)$, we can conclude that r^k is A-orthogonal to $p^0, \ldots, p^{k-2}$ and $r^0, \ldots, r^{k-2}$. The observation that r^k is A-conjugate to $p^0, \ldots, p^{k-2}$ is the key to rewrite (3.6).

$$p^k = r^k - \sum_{j<k} \frac{\left(p^j\right)^T A r^k}{\left(p^j\right)^T A p^j} p^j = r^k - \frac{\left(p^{k-1}\right)^T A r^k}{\left(p^{k-1}\right)^T A p^{k-1}} p^{k-1} = r^k - \frac{\left(r^k\right)^T A p^{k-1}}{\left(p^{k-1}\right)^T A p^{k-1}} p^{k-1} \tag{3.8}$$

This already looks a bit like the expression for p^k given in Step 5 of Algorithm 1. That is, $p^k = r^k - \beta_k p^{k-1}$. However, we do not have the same β_k yet. Therefore, we rewrite equation (3.8) using Step 4 of the CG algorithm, which tells us that $r^k = r^{k-1} - \gamma_{k-1} A p^{k-1}$ and so $A p^{k-1} = \frac{1}{\gamma_{k-1}}(r_k - r_{k-1})$.

$$\begin{aligned}
p^k &= r^k - \frac{\left(r^k\right)^T A p^{k-1}}{\left(p^{k-1}\right)^T A p^{k-1}} p^{k-1} \\
&= r^k - \frac{\left(r^k\right)^T \frac{1}{\gamma_{k-1}}(r_k - r_{k-1})}{\left(r^{k-1} - \beta_{k-1} p^{k-2}\right)^T A p^{k-1}} p^{k-1} \\
&\overset{(1)}{=} r^k - \frac{\frac{1}{\gamma_{k-1}}\left(r^k\right)^T (r_k - r_{k-1})}{\frac{1}{\gamma_{k-1}}\left(r^{k-1}\right)^T (r_k - r_{k-1})} p^{k-1} \\
&\overset{(2)}{=} r^k + \frac{\left(r^k\right)^T r_k}{\left(r^{k-1}\right)^T r_{k-1}} p^{k-1} \\
&= r^k - \beta_k p^{k-1}.
\end{aligned}$$

This is exactly what we see in Step 5 of the CG algorithm. In equality (1), we used that $\left(p^{k-2}\right)^T A p^{k-1} = 0$, as the search directions are conjugate. At (2), we used that $\left(r^{k-1}\right)^T r^k = 0$.

Finally, we can also give the expression for γ_{k-1} which is given in Algorithm 1. Recall that we had already derived $\gamma_k = \frac{\left(p^k\right)^T r^k}{\left(p^k\right)^T A p^k}$ in (3.5). We can now rewrite p^k and apply orthogonality of r^k and p^{k-1} to find $\gamma^k = \frac{\left(r^k - \beta_k p^{k-1}\right)^T r^k}{\left(p^k\right)^T A p^k} = \frac{\left(r^k\right)^T r^k}{\left(p^k\right)^T A p^k}$. This is written more compactly as $\gamma_k = \frac{\left(r^k\right)^T r^k}{d_k}$, where $d_k := \left(p^k\right)^T A p^k$. With this, we have covered Step 5 and 6 of Algorithm 1. Step 7 is just there to update the iteration counter. After that, we start a new iteration and go through Steps 1 to 7 all over again, or we stop the algorithm if $||r^k||_2 < \varepsilon$ [21].

Notice that the algorithm cannot perform more than n iterations. Indeed, at iteration $k = n$, we will have minimised $\phi(x)$ over $x^0 + \mathcal{K}_n(A; r^0)$, which equals $x^0 + \text{span}\{p^0, \dots, p^{n-1}\}$ by Lemma 3.2.2*(ii)*. The vectors $p^0, \dots, p^{n-1}$ span the whole of $\mathbb{R}^n$. Thus, the CG algorithm terminates automatically after n iterations.

3.2.2. From Solver to Sampler

We have now covered the entirety of Algorithm 1, except Step 3. As promised, let us now go over this step. We denote by P_k the $n \times k$ matrix with columns $\{p^j\}_{j=0}^{k-1}$. Furthermore, denote by $\tilde{P}_k$ the $n \times (n-k)$ matrix with columns $\{p^j\}_{j=k}^{n-1}$. As all the search directions are A-orthogonal, they are linearly independent. This is shown in the proof of Lemma 3.2.2. Therefore, the matrix $P_n = \begin{pmatrix} P_k & \tilde{P}_k \end{pmatrix}$ is invertible and the matrix

$$D_n := \begin{pmatrix} D_k & 0 \\ 0 & \tilde{D}_k \end{pmatrix} = \begin{pmatrix} P_k^T A P_k & 0 \\ 0 & \tilde{P}_k^T A \tilde{P}_k \end{pmatrix} = P_n^T A P_n$$

is invertible and diagonal. The latter can easily be seen as $(D_n)_{ij} = \left(p^i\right)^T A p^j$, which is zero unless $i = j$. We can restate the above as

$$A^{-1} = P_n D_n^{-1} P_n^T = P_k D_k^{-1} P_k^T + \tilde{P}_k \tilde{D}_n^{-1} \tilde{P}_n^T$$

Now, Step 3 of Algorithm 1 tells us to pick $y^k = y^{k-1} + \frac{z}{\sqrt{d_{k-1}}} p^{k-1}$, with $z \simeq \mathcal{N}(0,1)$. This recursive formula can be written in terms of y^0 by substituting $y^{k-1} = y^{k-2} + \frac{z'}{\sqrt{d_{k-2}}} p^{k-2}$ and so on, i.e. $y^k = y^0 + P_k D_k^{-1/2} z^k$, with $z^k \simeq \mathcal{N}(0, I_k)$. But then, if we denote by $\mathbb{E}[y^k | y^0, b]$ and $\mathrm{Var}(y^k | y^0, b)$ the expectation and variance of the random vector y^k, found after k iterations of the CG sampler initialised with y^0 and b, we find:

$$\mathbb{E}\left[y^k \middle| y^0, b\right] = \mathbb{E}\left[y^0\right] \tag{3.9}$$

$$\mathrm{Var}\left(y^k \middle| y^0, b\right) = \mathrm{Var}\left(y^0\right) + \mathrm{Var}\left(P_k D_k^{-1/2} z^k\right) = \mathrm{Var}\left(y^0\right) + P_k D_k^{-1} P_k^T. \tag{3.10}$$

Thus, after $k = n$ iterations, we have that $P_n D_n^{-1} P_n^T = A^{-1}$ and so $y^n \simeq \mathcal{N}(0, A^{-1})$. Note that we assumed that $y^0 = 0$ here, such that $\mathbb{E}\left[y^0\right] = \mathrm{Var}\left(y^0\right) = 0$ does not contribute to the expectation and variance of the final sample.

The conclusion that we have a sample $y^n \simeq \mathcal{N}(0, A^{-1})$ after n iterations is very useful for the theoretical treatment of the CG method. However, when working with the finite precision of a computer, a number of things change. The biggest issue with the CG algorithm is the so-called loss of conjugacy. Due to floating point rounding errors, the CG method will ultimately generate new search directions p^k that are no longer A-conjugate to the previous ones, hence the term 'loss of conjugacy'. When this occurs, the algorithm will no longer follow the theory we have described in this Section. For a detailed account on this interesting topic, we refer the reader to Meurant and Strako [22]. If loss of conjugacy occurs at iteration k, the sample y^k will be the best we will ever get from the CG sampler. By (3.10), $\mathrm{Var}\left(y^k \middle| y^0, b\right)$ will be the best approximation of A^{-1} we can make using vectors from the Krylov subspace $\mathcal{K}_k(A; r^0)$. If we continue iterating after loss of conjugacy occurred, we will neither increase nor decrease the quality of the approximation [8].

3.3. Analysing the Percolative Behaviour

Now that we have a finite approximation of a Discrete Gaussian Free Field, we can start investigating the percolative behaviour on the lattice. In particular, we are interested in determining the critical value h_* defined in Definition 2.2.1.

3.3.1. Clustering: The Hoshen-Kopelman Algorithm

Given a realisation of a DGFF on a finite region, and a threshold value h which determines which lattice points are in the level-set above h, we want to find out how many clusters there are. Also, we would like to say how many sites each cluster contains. This can be achieved using the Hoshen-Kopelman algorithm. This algorithm gives every occupied site (a site with $\varphi_i \geq h$) a label, indicating that it belongs to a certain cluster.

The Hoshen-Kopelman algorithm is a so-called union-find algorithm. Given a collection of elements, in our case lattice sites, the Union-Find algorithm helps us to determine equivalence classes, in our case clusters. We are speaking of equivalence classes because the relation 'i and j are in the same cluster', denoted by $i \leftrightarrow j$, is an equivalence relation. Indeed, $i \leftrightarrow i$ (reflexive), $i \leftrightarrow j$ implies that $j \leftrightarrow i$ (symmetric) and $i \leftrightarrow k$ whenever $(i \leftrightarrow j \wedge j \leftrightarrow k)$ (transitive). We define two functions for this union-find method; `union(i,j)` and `find(i)`. The function `find(i)` helps us identify the equivalence class of which i is an element. It returns a representative element of this equivalence class. The `union(i,j)` function 'unifies' the equivalence classes that i and j are respectively elements of. It tells us that from this moment onward, i and j are in the same equivalence class by setting the representative elements of both classes equal to each other, i.e. `find(i)=find(j)`.

Let us make this general union-find toolset more concrete by applying it to the cluster-finding problem on the $N \times N \times N$ region $\Lambda = \{0, \ldots, N-1\}^3$. Each lattice point in Λ is identified by an index. This is a number between 0 and $N^3 - 1$. The index 0 is given to $i = (0,0,0)$. From there we index sites going along the positive x-direction. Once we cannot go further in the x-direction, we move up one site in the y-direction and resume. Once we have filled up an entire xy-plane, we move up one site in the z-direction. This is repeated until the entire region is indexed. The procedure is illustrated in Figure 3.1.

In Algorithm 2, one can find the main idea of the Hoshen-Kopelman (H-K) algorithm, tailored to our three-dimensional setting. This algorithm assigns a label to each lattice point, which tells us the cluster that specific lattice point is a part of.

Initially, we set the label of each site equal to its index. Now, the algorithm goes through every lattice site and checks whether or not it is occupied, i.e. if it is in the level-set or not. It does that by starting at site 0, and then going up incrementally towards site $N^3 - 1$. Each unoccupied site receives a special label: 'unoccupied'. At each occupied site, Hoshen-Kopelman looks for nearest-neighbour sites 'behind the site'. With the latter we mean the first three sites in the negative x-, y- and z-

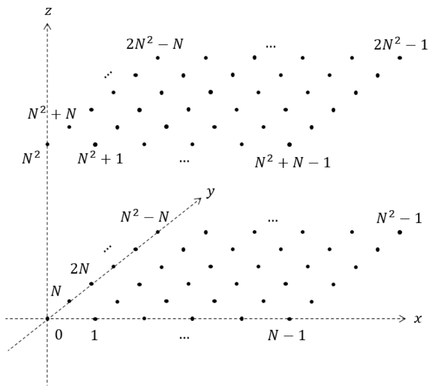


Figure 3.1: Illustration of the procedure for indexing lattice sites used in the Hoshen-Kopelman algorithm.

direction. Thus, if H-K is currently at site i, it checks if the sites $i-1$, $i-N$ and $i-N^2$ are occupied. If i is at some boundary, some of these three points might not exist. In that case we look at the remaining ones that do exist. We are interested in how many of these three nearest-neighbours are occupied. If none of the neighbours are occupied, this means that, at least for now, site i is part of an undiscovered cluster. This cluster is labelled with i. Therefore also site i gets to keep its index as its label. It is also possible that one of the three nearest-neighbours was occupied. In that case, site i belongs to the cluster of that occupied neighbour. We give i the label of the occupied cell. Note that what we are executing a `find()`-command to look for a representative element of this cluster and give the site i the same label as that element. Finally, it is also possible for a site i to have multiple (2 or 3) occupied neighbours. When this is the case, we possibly have neighbours with different cluster labels. For example, we have three occupied points. All these points have a label indicating to which cluster they belong, say k, l and m respectively ($k > l > m$ without loss of generality). The first thing to do is to perform a `union()`-operation on the occupied neighbouring sites. This will set the representative elements of the three cluster to the same element. This element is chosen to be $\min\{k, l, m\} = m$. The site i itself will receive that same label. Notice however that we are not done yet. There might have been other sites in the lattice with the label k or l. We have only updated the label of the representative elements k and l themselves. The other sites still have their old label. Fortunately, the solution for this issue is very simple. After we have run through all the sites, we go through them once again. This time however, we set the label of every node equal to `find(i)`. Two things can happen now. If i had already the most updated label, `find(i)` will just return the same label. If i had an outdated label though, which could happen in the scenario we sketched earlier, something more happens. The `find(i)` function will first go to the site corresponding to i's old label, but then it will realise that this site is in fact part of a bigger cluster with another representative element as its label. Then `find(i)` will return that representative element and this will be the new label for i.

Ultimately, we will have gone through all the lattice sites. Each occupied lattice site i will have a label. This label will be the index of the representative element of the cluster i belongs to. By construction, this element will be the lattice site in the cluster with the smallest index. We are now able to distinguish different clusters, as each cluster label is unique. Consequently, by counting all sites with a certain label, we can determine the size of the cluster with that label. This paves the way to the computation of all sorts of cluster-related quantities, which will turn out very helpful in the next section [23].

3.3.2. Finding the percolation threshold

To find the percolation threshold, we make use of a method described in Marinov and Lebowitz [9]. In their paper, they consider a zero-average Discrete Gaussian Free Field on a lattice with periodic boundary conditions. That is, they consider a DGFF which is generated by the method from Sheffield [3]. To find the percolation threshold, Marinov and Lebowitz look at the the empirical second moment of the cluster size, $\Gamma_L := \left\langle \sum_s s^2 N_s \right\rangle$. This quantity is an average over multiple samples of a DGFF, with s the cluster size and N_s is the amount of cluster with that size s. Notice that $N_s = N^d n_s$, with n_s the cluster number. We may therefore write $N^{-d}\Gamma_N = pS(p)$, with $S(p)$ the average cluster size. Recall that for $p > p_c$, we neglect the infinite cluster when determining $S(p)$. We will compute Γ_N for different values of N and p. Note that p now has a slightly different meaning than in Section 2.2.1. In the latter, each site of the lattice had a probability p to be occupied. Now, whether or not a site is occupied depends on the threshold value h. How can we link this to an occupation probability p? Say that we have some instance of a DGFF on Λ. Then we may define for each lattice site $i \in \Lambda$ the occupation variable

$$\rho_h(i) = \begin{cases} 1, & \varphi_i \geq h, \\ 0, & \varphi_i \leq h. \end{cases} \tag{3.11}$$

In this context, we define $p = \langle \rho_h(i) \rangle_{i \in \Lambda}$. Thus p is the average of all occupation variables in Λ. We can interpret it as the (lattice size independent) concentration of occupied sites. With this definition, p_c corresponds to the critical value h_*. In practice, if we want to find the level-set corresponding to some concentration p, we look for the $(1-p)$-quantile of the set $\{\varphi_i : i \in \Lambda\}$. Choosing this quantile as our h makes sure that a fraction p of the spins φ_i are in the level-set.

For large N and p close to p_c, we can say that $N^{-d}\Gamma_N = pS(N,p) \sim p|p-p_c|^{-\gamma}$, as in Definition 2.2.3. In that case, we may apply a finite-size scaling argument as described at the end of Section 2.2.1:

$$N^{-d}\Gamma_N \sim p|p-p_c|^{-\gamma} f_2\Big(N^{\frac{1}{\nu}}(p-p_c)\Big) \sim pN^{\frac{\gamma}{\nu}} F\Big(N^{\frac{1}{\nu}}(p-p_c)\Big) \tag{3.12}$$

Here we defined

$$F(x) = \begin{cases} 1, & x^{\nu} \ll 1, \\ x^{-\gamma}, & x^{\nu} \gg 1 \end{cases}$$

to match the form of Equation (5) in [9]. However, in this equation (5) the careful reader may note that the p is left out. We are only interested in the behaviour around the percolation threshold. The quantity $(p-p_c)^{-\gamma}$ is the only one that may diverge around this threshold. It is therefore the only p-dependent quantity that plays a role in the scaling behaviour. This is why Marinov and Lebowitz left it out.

Equation (3.12) implies that the ratio $R_N := \frac{\Gamma_{2N}}{\Gamma_N}$ will become independent of N when $p = p_c$ as N grows very large. Indeed,

$$R_N = \frac{\Gamma_{2N}}{\Gamma_N} \sim \frac{(2N)^d p (2N)^{\frac{\gamma}{\nu}} F\Big((2N)^{\frac{1}{\nu}}(p-p_c)\Big)}{N^d p N^{\frac{\gamma}{\nu}} F\Big(N^{\frac{1}{\nu}}(p-p_c)\Big)} = 2^{d+\frac{\gamma}{\nu}} \tag{3.13}$$

Thus, if we plot R_N as a function of p, for different values of N, the resulting curves should intersect at $p = p_c$. This is how we will find the percolation threshold! Moreover, the value of R_N at this intersection point is $2^{d+\frac{\gamma}{\nu}}$, yields the radio $\frac{\gamma}{\nu}$ as d is known. In our case, we are working with a three-dimensional Discrete Gaussian Free Field, so $d = 3$. Notice that the extra factor p that we have in (3.12) compared to [9] cancels anyway.

Algorithm 2: Hoshen-Kopelman algorithm

```
Input: a set of $N \times N \times N$ lattice points; each lattice point has an index
       between 0 and $N^3 - 1$ and can be either occupied (part of the
       level-set) or unoccupied (not part of the level-set)
Output: a label for each lattice point, indicating which cluster that lattice
        point belongs to
set the label of each lattice site equal to its index;
for $i \in \{0, \ldots, N^3 - 1\}$ do
    if site i is occupied then
        if sites $i - 1$, $i - N$ and $i - N^2$ are all unoccupied then
            the label of $i$ remains its index;
        end
        else if exactly one of the sites $i - 1$, $i - N$ and $i - N^2$ is occupied then
            $i$ takes the same label as the occupied cell;
        end
        else if exactly two of the sites $i - 1$, $i - N$ and $i - N^2$ are occupied then
            unify the clusters of the two occupied neighbours into one
              cluster;
            $i$ takes the same label as the occupied cell with the smallest label;
        end
        else if all of the sites $i - 1$, $i - N$ and $i - N^2$ are occupied then
            unify the clusters of all the occupied neighbours into one cluster;
            $i$ takes the same label as the occupied cell with the smallest label;
        end
    end
    else
        site $i$ will receive a label 'unoccupied'.
    end
end
for $i \in \{0, \ldots, N^3 - 1\}$ do
    if site i is occupied then
        denote the label of site $i$ by $l_i$, then find the representative element
          the site $l_i$ itself is a part of, this element is the new label of $i$;
    end
end
```

4
SIMULATIONS

In this fourth chapter, we will present a number of checks we executed to make sure our implementations of, amongst others, Hoshen-Kopelman and the CG sampling method were valid. We will discuss the results from these checks. After that we study the DGFF on a lattice with checkerboard conductances, and state a conjecture regarding this object.

4.1. Checks of Validity

In order to be sure that the code is working properly, we may perform a number of checks. Each check will go over a different part of the code. The idea is that if each check yields a positive result, we will have a completely working program, from DGFF simulation to computation of R_N. We go over two checks in this Section:

- We convince ourselves that the Hoshen-Kopelman algorithm and computation of R_N work correctly. To this end, we run a level-set percolation model on the DGFF with periodic boundary conditions and try to retrieve the percolation threshold that Marinov and Lebowitz found [9]. If this goes well, we have a working implementation of Hoshen-Kopelman and R_N is correctly calculated. Moreover, our implementation of the method described in 3.1.1 also works out.
- Examine the behaviour of the DGFF samples we retrieve from the CG sampler. In particular, compute R_N for samples with different linear sizes N and concentrations p. If the CG sampler works correctly, we would expect that samples from the CG method yield comparable results as samples from the Fourier-analytic methods described in Section 3.1.

4.1.1. Checks for Hoshen-Kopelman and R_N-computation

To see whether the Hoshen-Kopelman cluster-finding algorithm and the subsequent calculations to find Γ_N are correct, we will try to reproduce the results from Marinov and Lebowitz [9]. With the Fourier-analytic method from Section 3.1.1, we generate instances of a DGFF with periodic boundary conditions. We should retrieve similar results as in Figure 1 of [9]. If this test gives correct results, we can conclude, with a high degree of confidence, that our implementation of Hoshen-Kopelman, as well as the computation of R_N, are correct. Not only that, the DGFF sampler using the method from Sheffield [3] matches the results of Marinov and Lebowitz.

The experiment has been carried out as follows. For each p between 0.13 and 0.16 (with increments of 0.005), we computed R_N for $N = 10$, $N = 20$ and $N = 40$, where the respective Γ_N and Γ_{2N} were the average of $\sum_s s^2 N_s$ over 5000 DGFF instances. To get a sense for the uncertainty in the final result, we employ the method proposed in [9]. We split up the found Γ_N and Γ_{2N} in 10 groups. For each group we computed the average. Assuming that the averages were normally distributed, we computed the standard deviation of this population of 10 values and used it to quantify the uncertainty in the found values. The results of this procedure can be found in Figure 4.1. If we compare this plot to Figure 1 in Marinov and Lebowitz, we identify an issue. The intersection point indicates a percolation threshold of $p_c = 0.14 \pm 0.01$, smaller than the $p_c = 0.16 \pm 0.01$ expected from [9]. Also, the values found for R_N are also a bit higher than expected. Note that our value for the uncertainty in p_c is visually estimated from Figure 4.1.

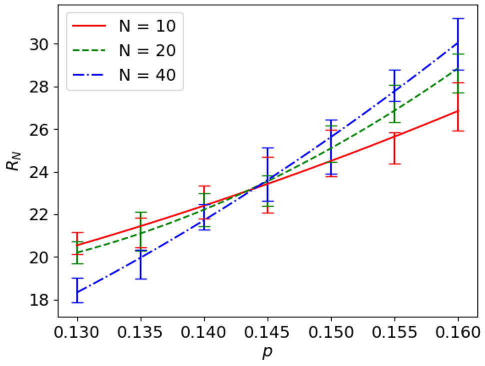


Figure 4.1: R_N as a function of p, for $N = 10$, $N = 20$ and $N = 40$ for the Discrete Gaussian Free Field with periodic boundary conditions. The intersection points of two subsequent curves N and $2N$ at $p \approx 0.145$ indicates the critical probability for this model.

Towards the end of the project, the idea came up that this odd behaviour is caused by the way we count clusters. The way it was done in the above simulations was by assuming 'zero boundary conditions' for clustering. That is, if we consider the simplified example Figure 4.2, where occupied sites are indicated in black, we would count two clusters: 1 and 2. However, Marinov and Lebowitz count clusters assuming periodic boundary conditions. This makes sense as their DGFF is also considered on a lattice with periodic boundary conditions. In that case, clusters 1 and 2 in Figure 4.2 would in fact be the same cluster.

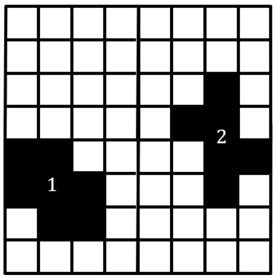


Figure 4.2: The two ways to count a clusters. In the case of zero boundary conditions, we would count two different clusters, 1 and 2. In the case of periodic boundary conditions, we would count just one cluster, as 1 and 2 are connected to each other via the boundary.

Because this insight came up in the final stages of the project, we were unable to test out this hypothesis. Our implementation of Hoshen-Kopelman described in Section 3.3.1 breaks down when counting clusters using periodic boundary conditions. This is because our H-K algorithm goes through all occupied sites only looking back at those sites that have already been considered. With periodic boundary conditions however, boundary sites that have already been considered might have an influence at the other side of the lattice. As H-K only looks at sites that have passed, it will miss the influence of these boundary sites. The algorithm sees them as 'not yet considered'. We could not adjust our implementation in time to account for this issue. Despite this, we may still give an explanation for why this is most likely the cause of our troubles, albeit not as satisfying as a working simulation.

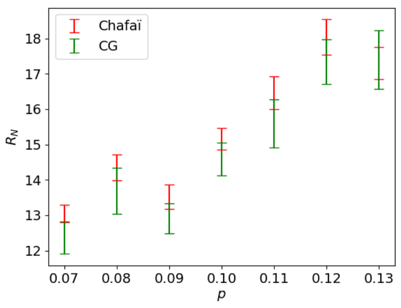
In our simulations, we plot $R_N = \frac{\Gamma_{2N}}{\Gamma_N}$. Let us focus on Γ_N for now. As we do not consider periodic boundary conditions, clusters which would otherwise be linked via a boundary are now counted as two, small clusters instead of as one, larger cluster. Recall that Γ_N is the average over multiple samples of the second moment of the cluster size, $\sum_s s^2 N_s$. Since we square the cluster size, one large cluster has a higher contribution to the second moment than two small clusters. Consider for example two clusters of size 2 that would be one cluster of 4 in the case of periodic boundary conditions. The contribution to the second moment from the two small clusters is $2^2 \cdot 2 = 8$, while the contribution from one large cluster is $4^2 \cdot 1 = 16$. We can thus conclude that zero boundary conditions yield a smaller Γ_N than periodic boundary conditions. Note that this effect is relatively stronger for smaller lattices. The larger the lattice, the more clusters we have and the more contributions to Γ_N. This discrepancy between zero and periodic boundary conditions becomes less and less relevant for growing N. Therefore, the ratio $\Gamma_N^{\text{ZBC}}/\Gamma_N^{\text{PBC}}$ will be smaller than the ratio $\Gamma_{2N}^{\text{ZBC}}/\Gamma_{2N}^{\text{PBC}}$. Thus,

$$R_N^{\text{ZBC}} = \frac{\Gamma_{2N}^{\text{ZBC}}}{\Gamma_N^{\text{ZBC}}} > \frac{\Gamma_{2N}^{\text{PBC}}}{\Gamma_N^{\text{PBC}}} = R_N^{\text{PBC}}.$$

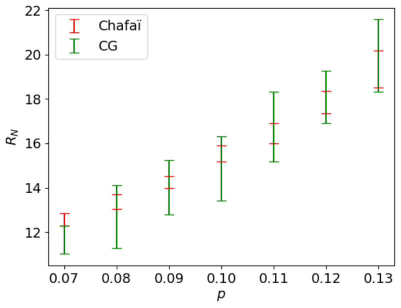
This would explain why the values of R_N from Figure 4.1 are higher than those in Marinov and Lebowitz.

4.1.2. Comparing the CG sampler with the Fourier-analytic methods

Let us move on to checking that the CG sampler yields similar results as the Fourier-analytic methods. The latter are used here as a benchmark. We will repeat the experiment from Section 4.1.1, but now compare the results obtained with the CG sampler to those obtained with the method described in Section 3.1.2. We have run simulations for $N = 5$ and $N = 10$, with p going from 0.07 to 0.13 with increments of 0.01. The averages Γ_N and Γ_{2N} were taken over 5000 samples and the errors have been computed as before. The CG sampler was initialised with $y^0 \simeq \mathcal{N}(0, I_{N^3})$ and had a stopping tolerance of 10^{-70} for the $N = 5$ run and a tolerance of 10^{-80} for the $N = 10$ run. One can find the outcome of the simulation for $N = 5$ in Figure 4.3a and for $N = 10$ in Figure 4.3b. Within the error margin, the results from both methods seem to coincide. We can also check whether both methods yield about the same percolation threshold. This is done in Figures 4.4a for Chafaï and 4.4a for CG.

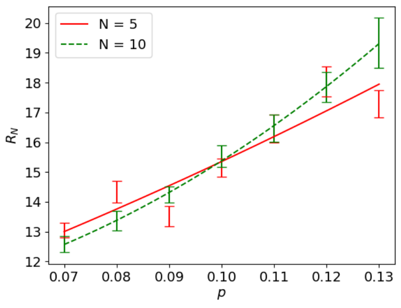


(a) $N = 5$.

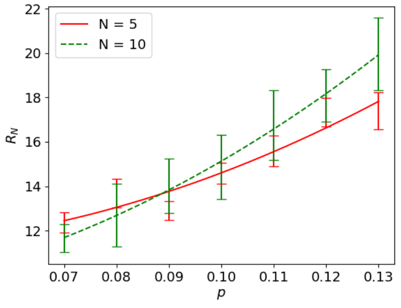


(b) $N = 10$.

Figure 4.3: Comparison of $R_N(p)$-values found from DGFF samples produced by the CG sampler and method from Chafaï for (a) $N = 5$ and (b) $N = 5$.



(a) Chafaï's method.



(b) CG Sampler.

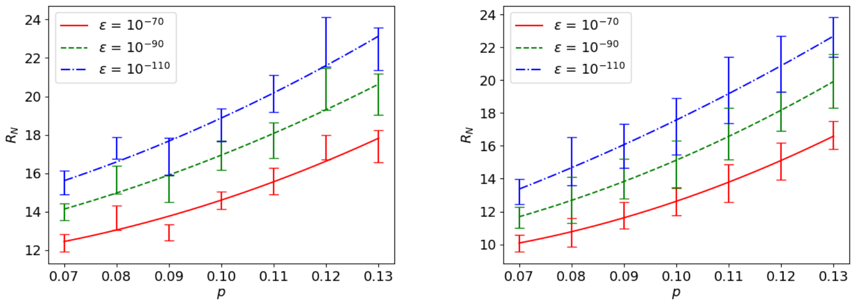
Figure 4.4: R_N as a function of p for $N = 5$ and $N = 10$, computed with samples from (a) the method from Chafaï and (b) the CG sampler.

From Figures 4.4a and 4.4b we would estimate $p_c^{\text{Chafaï}} = 0.10 \pm 0.01$ and $p_c^{\text{CG}} = 0.09 \pm 0.01$. These results certainly do not contradict each other. Again, the uncertainties in critical probability are visually estimated from the plots.

It should be noted that many error bars from the $N = 10$ run with the CG method are relatively wide. Apparently, averaging over 5000 samples is not enough. For future simulations, one might consider computing averages over more samples. Clearly, the computing power in this project was limited. This could have been seen already by looking at the lattice sizes considered here ($N = 5$ and $N = 10$). These are rather small, compared to say the $N = 20$ and $N = 40$ used in Section 4.1.1.

The limiting factor here was the sampler using the method from Chafaï. As discussed in Section 3.1.2, this method relies on the computation of the matrix $\Sigma_\Lambda^{1/2}$ defined as in (3.3). Once the matrix was computed, we could generate zero-boundary DGFF samples using (3.4) relatively quickly. However, for a linear lattice size N, $\Sigma_\Lambda^{1/2}$ is an $N^3 \times N^3$ matrix, where each entry is a sum over N^3 terms. This becomes quickly infeasible to compute without the use of parallel computing. We will return to this observation in Chapter 5.

Another observation we made during the simulations with the CG sampler, is that the chosen stopping tolerance had a significant effect on the sample the CG method produced. This is illustrated in Figure 4.5. There, one can see R_N as a function of p, found by simulating DGFFs with the CG method with different stopping tolerances. In particular, we tested the tolerances 10^{-70}, 10^{-90} and 10^{-110} for both $N = 5$ and $N = 10$, again with p running from 0.07 to 0.13 with steps of 0.01 and averaged over 5000 samples.



(a) $N = 5$. (b) $N = 10$.

Figure 4.5: Comparison of $R_N(p)$-values found from DGFF samples produced by the CG sampler with different stopping tolerances (a) $N = 5$ and (b) $N = 5$.

To quantify which stopping tolerance yielded the best results, we defined a squared difference between two sequences of R_N values. That is, given $r_N := \big(R_N(p)\big)_{p\in\mathscr{P}}$ and $r'_N := \big(R'_N(p)\big)_{p\in\mathscr{P}}$, with $\mathscr{P}$ the set of values p runs through, we define:

$$d\big(r_N, r'_N\big) := \sum_{p\in\mathscr{P}} \big|R_N(p) - R'_N(p)\big|^2 \tag{4.1}$$

as the distance between r_N and r'_N. If we have computed such sequences of R_N values for different stopping tolerances (or more general for two different simulation methods), we can use this quantity as a measure of similarity between the two methods. The idea is that the smaller the corresponding $d\big(r_N, r'_N\big)$, the more alike the samples coming from the two methods are. When comparing the runs with different stopping tolerances, we found the values in Table 4.1.

Table 4.1: Distances between the run with Chafaï's method and the run with CG sampler for the given stopping tolerance ε. Distances have been computed for both $N = 5$ and $N = 10$.

ε	10^{-70}	10^{-90}	10^{-110}
$N = 5$	2.839	23.603	100.900
$N = 10$	42.706	2.293	28.266

We decided to use the stopping tolerance which yielded the R_N-sequence closest to that of the run with the method from Chafaï in the sense of (4.1). That is, we picked $\varepsilon = 10^{-70}$ for the $N = 5$ run and $\varepsilon = 10^{-9}$ for the $N = 10$ run.

4.2. THE CHECKERBOARD PATTERN

To finish off this Chapter, we move away from the standard DGFF and study a more complex conductance pattern: the checkerboard pattern described in Section 2.3. Using the CG sampler, we want to generate DGFF samples with checkerboard conductances a and b. Thereafter, we will repeat the computation of R_N with these samples, as described in Section 3.3.2. We are curious whether we can summarise the behaviour of such a checkerboard DGFF (with two conductances) by only one "effective" conductance. We studied the problem for $a = 0.5$, $b = 1$, $N = 10$, for p going from 0.05 to 0.20 in increments of 0.025 and for Γ_N, Γ_{2N} taken as averages over 500 samples. For these values we performed two sets of simulations:

1. First, we simulated instances of a checkerboard DGFF with the given a and b to compute R_N.
2. Second, we computed R_N for instances of a DGFF with every conductance equal to c, where we let c run from 0.5 to 1 with increments of 0.1.

For these simulations, the uncertainty was determined just as before. We want to find out for which c the constant conductance DGFF comes closest to the checkerboard DGFF. Here we define "close" in the same mean-squared sense we introduced in (4.1). The found values of R_N are plotted as a function of p in Figure 4.6. Furthermore, we can find the distance between the R_N-sequence from the checkerboard run and the R_N-sequence from the constant conductance model for each c in Table 4.2.

Table 4.2: Distances between R_N-sequence from the DGFF with checkerboard conductances and R_N-sequences from constant conductance DGFFs with conductances equal to $c = 0.5, 0.6, \ldots, 1$.

c	0.5	0.6	0.7	0.8	0.9	1
Distance	142.476	40.208	21.017	103.215	156.876	349.101

Looking at Table 4.2, it seems that the DGFFs with constant conductance $c = 0.6$ and $c = 0.7$ are the closest to the checkerboard DGFF. Therefore, we repeated our

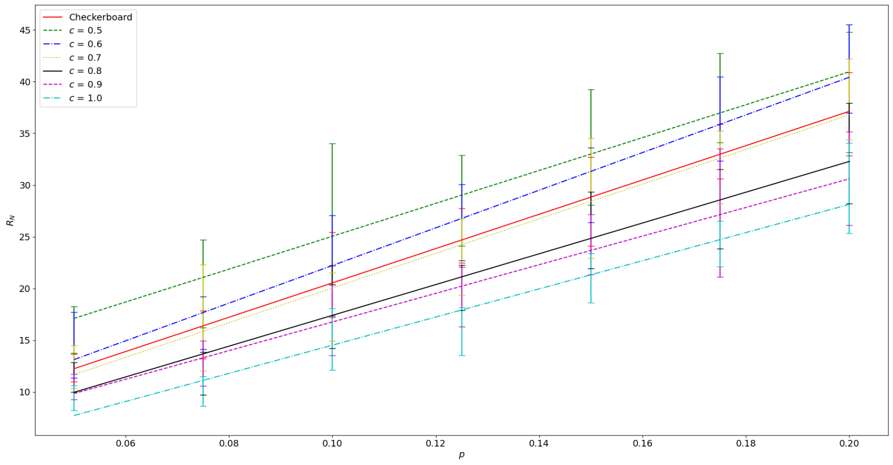


Figure 4.6: R_N as a function of p for the DGFF with checkerboard conductance pattern with $a = 0.5$ and $b = 1$ and for the different constant conductance DGFFs ($c = 0.5, 0.6, \ldots, 1$).

experiment with a DGFF that had constant conductance $c = 0.65$. However now we averaged over 1000 samples. This DGFF had a distance 9.774 from the checkerboard DGFF. In Figure 4.7, we compare the $R_N(p)$ from the checkerboard pattern to only the $R_N(p)$ of the DGFF with $c = 0.65$. The two models seem to agree really well regarding their $R_N(p)$-values. We can try to give a – not at all rigorous – explanation of this result. Consider the checkerboard conductance pattern. We can interpret a DGFF on a lattice with this pattern as a group of smaller, constant conductance DGFFs defined on 5×5 sub-lattices, with either conductance $a = 0.5$ or $b = 1$. These small sub-DGFFs are patched together in an alternating fashion, such that no two DGFFs defined with the same conductance are next to each other. Lattice sites in a sub-lattice with a conductance of 1, will be more connected to each other than lattice sites in a sub-lattice with a conductance of 0.5. That is, the spins in a DGFF with conductance 1 will likely be closer to each other than spins in a DGFF with conductance 0.5. Because of this, it is reasonable to assume that large clusters will form earlier in a DGFF with conductance 1 than in a DGFF with conductance 0.5. Eventually, we will have a percolating cluster in a DGFF with conductance 1 before the we will have a percolating cluster in a DGFF with conductance 0.5.

Let us now zoom back out and wonder when we will have a percolating cluster that spans over the whole lattice. When we let p gradually increase, we expect that for a certain $p = p_b$ all the sub-DGFFs with conductance $b = 1$ will percolate. However, at this point, the sub-DGFFs with conductances $a = 0.5$ will not yet percolate. Because the sub-DGFFs are arranged in the checkerboard pattern, the sub-DGFFs with conductance 0.5 will block percolation of the whole lattice. It is not until p is

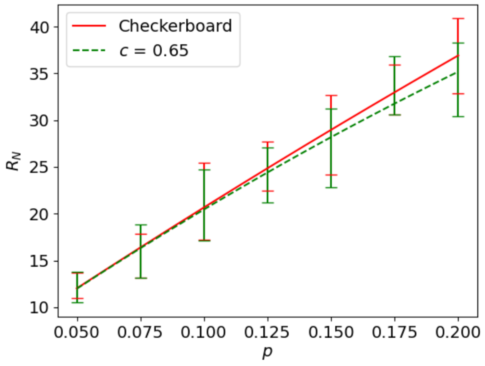


Figure 4.7: R_N as a function of p for the DGFF with checkerboard conductance pattern with $a = 0.5$ and $b = 1$ and for the constant conductance DGFF with $c = 0.65$.

large enough, say $p = p_a$ that the sub-DGFFs with conductance $a = 0.5$ will percolate as well. Only then will there be percolation through the entire lattice. In other words, the percolative behaviour of the DGFF in the entire lattice is dictated by the percolative behaviour of the sub-DGFFs on the sub-lattice with the lowest conductance.

This reasoning alone tells us that a checkerboard conductance DGFF with conductances a and b ($a < b$) shows the same percolative behaviour as a constant conductance DGFF with conductance $c = a$. This is not what Table 4.2 and Figure 4.7 tell us however. The checkerboard conductance DGFF behaves more like a constant conductance DGFF with conductance $c > a$. In the case of $a = 0.5$ and $b = 1$, we have found $c = 0.65$. This could be attributed to the realisation that we do not need to wait for percolation in the sub-DGFFs with conductance a. It suffices that a small number of sites at the corners of the sub-lattices with conductance a are occupied. This is illustrated in Figure 4.8 for the two-dimensional case. The same idea holds for the three-dimensional situation. Except there we need at least two occupied sites in the sub-lattice with conductance a to create a bridge between the two percolating sub-lattices.

Based on this – again, not at all rigorous – argument to explain what we saw for just one set of values ($a = 0.5$, $b = 1$ and $c = 0.65$), we have formulated a conjecture in Conjecture 4.2.1.

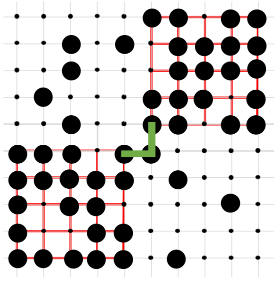


Figure 4.8: The level-set for $p \in (p_b, p_a)$. The red edges represent conductances b, the light-grey edges conductances a with $a < b$. In the red sub-lattice we already have percolation, but not yet in the light-grey sub-lattice. However, there just has to be one occupied site in the light-grey sub-lattice positioned correctly, that creates the green bridge, and we have a percolating cluster over the entire lattice.

Conjecture 4.2.1 (Checkerboard Conductances-Constant Conductances Equivalence)**.** Consider a three-dimensional lattice embedded with a checkerboard conductance pattern as described in Section 2.3, with conductances a and b such that $a < b$. The percolative behaviour of a Discrete Gaussian Free Field defined on this lattice is the same as the behaviour of a Discrete Gaussian Free Field defined on the same lattice but now with constant conductances with value c. This value is a weighted average of a and b, i.e.

$$c = \frac{f_a a + f_b b}{f_a + f_b}, \tag{4.2}$$

where $f_a > f_b$.

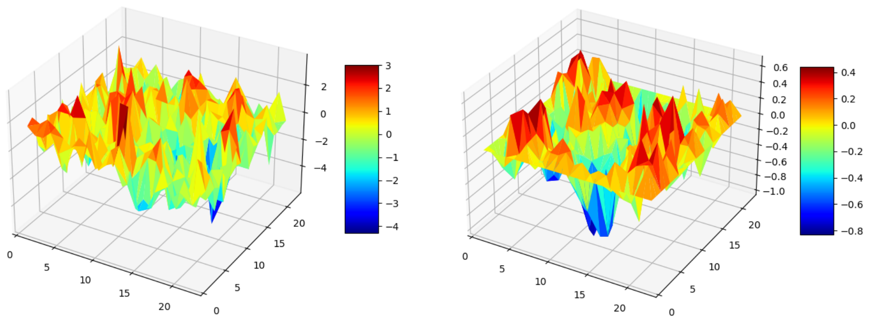
5 KNOWN ISSUES AND RECOMMENDATIONS

In Chapter 4, we already came across a number of problems with the code. We have not been able to find a solution or sound explanation for some of these problems. Let us list all these issues here. Apart from that, we will discuss a number of features which could be helpful to incorporate in the case of further research.

As already observed in Section 4.1.2, computing the matrix $\Sigma_\Lambda^{1/2}$ was a really time-intensive task. Because of this we did not manage to sample DGFFs using the method from Section 3.1.2 for a linear lattice size larger than $N = 10$. Indeed, computing R_N as defined in (3.13) already requires $\Sigma_\Lambda^{1/2}$ for Λ a $20 \times 20 \times 20$ box. This task however, is perfect to perform in parallel. The big matrix $\Sigma_\Lambda^{1/2}$ can be divided into smaller submatrices, that can then be computed simultaneously. We have not yet been able to make our code compatible with the HPC cluster of the TU Delft. For further research using the method, we would recommend making the necessary adjustments to the code in order to run it on a HPC cluster. This would not only be advantageous for computing $\Sigma_\Lambda^{1/2}$, but also for the general R_N-computation. This computation was made up of calculating Γ_N and Γ_{2N}, averages of the empirical second cluster moment over a large number of samples. Evidently, the more samples we average over, the smaller the uncertainty in our final value for $\mathbb{R}_N$. With parallel computing, we would be able to generate a large number of samples simultaneously. The time it takes to find a value for R_N with a reasonably small uncertainty would be drastically shortened. This in turn makes simulations with larger linear lattice sizes N more feasible.

Let us mention another point of interest. Every time we generated a DGFF using the CG sampler, the CG algorithm was initialised with a random vector $y^0 \simeq \mathcal{N}(0, I_{N^3})$. According to (3.10), the resulting sample y^k has variance $I_{N^3} + P_k D_k^{-1} P_k^T$.

For the best approximation of A^{-1} after k iterations, we would expect that a y^k with variance $P_k D_k^{-1} P_k^T$ gives the results the most similar to that of a real DGFF. Such a y^k could be found by initialising the CG algorithm with $y^0 = 0$. However, we found out during our simulations that initialising CG with $y^0 \simeq \mathcal{N}(0, I_{N^3})$ yielded more desirable results than initialising with $y^0 = 0$. It seems as if the starting sample y^0 has a significant influence on the shape of the sample y^k CG comes up with after k iterations. This is best explained by an image. In Figure 5.1, we can see two cross-sections from samples y^k returned by the CG algorithm. Figure 5.1a represents a sample y^k which is the result of a CG run with $y^0 \simeq \mathcal{N}(0, I_{N^3})$ and Figure 5.1b represents a sample y^k returned by a CG run with $y^0 = 0$. Both runs have stopping tolerance 10^{-90} and $N = 20$.



(a) $y^0 \simeq \mathcal{N}(0, I_{N^3})$.

(b) $y^0 = 0$.

Figure 5.1: Cross-section of sample y^k returned by the CG algorithm, initialised with (a) $y^0 \simeq \mathcal{N}(0, I_{N^3})$ and (b) $y^0 = 0$.

One should notice two things. First of all, the height scale of both DGFFs. When $y^0 \simeq \mathcal{N}(0, I_{N^3})$, the spin heights are spread out between -4 and 3. If $y^0 = 0$, the spin heights remain between -0.8 and 0.4. Second, the DGFF in Figure 5.1a has a more noisy character; with several spins just having a completely different value than their surroundings. This is in contrast with the DGFF in Figure 5.1b, where we see little to no single spins standing out. As we said earlier, it seems as if the starting sample is still explicitly present. This is strange, as we would expect that after enough iterations, different starting values for y^0 would yield the same kind of result.

Let us conclude this Chapter with a number of recommendations. We already mentioned that making the code compatible for parallel computing on a HPC cluster would greatly enlarge the possibilities of the CG sampler. Apart from that, it would be beneficial for the rigour of this paper if the experiment from Section 4.1.1 was repeated with an implementation of Hoshen-Kopelman that takes periodic boundary conditions into account. Finally, Conjecture 4.2.1 has been formulated based on an experiment which took only one pair of checkerboard conductances a and b into account. It would be wise to repeat the experiment for different (a, b)-pairs. Also looking at larger lattice sizes N and averaging over more samples for the R_N-computation is advisable. If with all these measures in place, Conjecture 4.2.1 still holds, an interesting question would be to find out whether the weights f_a and f_b from (4.2) are constant, or perhaps dependent on N.

6 CONCLUSION

A very insightful part of studying level-set percolation on the Discrete Gaussian Free Field (DGFF) is simulating such DGFFs. Various techniques already exist for generating DGFFs on lattices with unit conductances between the lattice points. However, for arbitrary conductances, there is no good alternative. In this project, we studied and implemented a Gaussian sampler based on the Conjugate Gradients (CG) method, and asked ourselves the question if it could be that good alternative. Finally, to show the CG sampler's flexibility concerning conductances, we studied level-set percolation on a DGFF with a checkerboard conductance pattern.

We conclude that the CG sampler is a promising, efficient method, cut out for approximately sampling high-dimensional Gaussian fields for which the inverse of the covariance matrix is known and is sparse. This is certainly the case for DGFFs. With the CG sampler, we were able to produce an approximation to a DGFF on a small lattice with zero boundary conditions and constant conductances that coincided with a known method from the literature. Apart from that, the simulations of the DGFF with checkerboard conductances allowed us to conjecture that a checkerboard conductance DGFF with conductances a and b with $a < b$ shows approximately the same percolative behaviour as a constant conductance DGFF with conductance c. This conductance c is some weighted average of a and b, where we expect the weight of a to be larger than the weight of b.

Important to mention however is that our program to generate a DGFF on a lattice with given conductances in order to study level-set percolation on this DGFF, does not come without problems. One issue we encountered was that whenever the stopping threshold of the CG sampler was below a certain level, which depended on the lattice size, the percolative behaviour drastically changed. Whereas stopping tolerances above this level yielded better and better approximations of a DGFF, once the tolerance became too small the approximations became worse again. The only explanation we could give for this was that for these small enough stopping tolerance, some kind of floating point error would start affecting the results. Finally, for further research we would recommend to make the code compatible with High Performance Computing clusters as soon as possible. This allows consideration of larger lattices and smaller uncertainties due to larger sample sizes.

References

[1] Alberto Chiarini and Maximilian Nitzschner. “Disconnection and entropic repulsion for the harmonic crystal with random conductances”. In: (2020). arXiv: 2012.05230 [math.PR].

[2] Jean Bricmont, Joel L. Lebowitz, and Christian Maes. “Percolation in strongly correlated systems: The massless Gaussian field”. In: *Journal of Statistical Physics* 48.5-6 (Sept. 1987), pp. 1249–1268. DOI: 10.1007/BF01009544.

[3] S. Sheffield. “Gaussian free fields for mathematicians”. In: *Probability Theory and Related Fields* 139 (2003), pp. 521–541.

[4] Djalil Chafaï. *Random walk, Dirichlet problem, and Gaussian free field*. Accessed: 2021-05-25.

[5] Sacha Friedli and Yvan Velenik. *Statistical Mechanics of Lattice Systems: A Concrete Mathematical Introduction*. Cambridge University Press, 2017. ISBN: 978-1-107-18482-4. DOI: 10.1017/9781316882603.

[6] Maxime Vono, Nicolas Dobigeon, and Pierre Chainais. “High-dimensional Gaussian sampling: a review and a unifying approach based on a stochastic proximal point algorithm”. In: (Oct. 2020).

[7] M. Hestenes and E. Stiefel. “Methods of conjugate gradients for solving linear systems”. In: *Journal of research of the National Bureau of Standards* 49 (1952), pp. 409–435.

[8] Albert Parker and Colin Fox. “Sampling Gaussian Distributions in Krylov Spaces with Conjugate Gradients”. In: *SIAM Journal on Scientific Computing* 34 (Jan. 2012). DOI: 10.1137/110831404.

[9] Vesselin Marinov and Joel Lebowitz. “Percolation in the Harmonic Crystal and Voter Model in three dimensions”. In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 74 (Sept. 2006), pp. 031–120. DOI: 10.1103/PhysRevE.74.031120.

[10] Jacopo Borga. “Percolazione per insiemi di livello del Gaussian free field”. In: (2018). URL: http://tesi.cab.unipd.it/57602/.

[11] Jean Jacod and Philip Protter. *Probability Essentials*. Springer, 2004. ISBN: 978-3-540-43871-7. DOI: 10.1007/978-3-642-55682-1.

[12] Elias M. Stein and Rami Shakarchi. *Real Analysis: Measure Theory, Integration, and Hilbert Spaces*. Princeton University Press, 2005. ISBN: 978-0-691-11386-9.

[13] *percolate*. https://www.merriam-webster.com/dictionary/percolate. Accessed: 2021-04-28.

[14] Kim Christensen. "Percolation Theory". In: (2002).

[15] D. Stauffer and A. Aharony. *Introduction To Percolation Theory: Second Edition*. CRC Press, 1994. ISBN: 978-0748402533.

[16] Allen Hunt, Robert Ewing, and Behzad Ghanbarian. *Percolation Theory for Flow in Porous Media*. Springer, 2014. ISBN: 978-3-319-03771-4. DOI: https://doi.org/10.1007/978-3-319-03771-4.

[17] Larry Riddle. *Koch Curve*. Accessed: 2021-07-06.

[18] H. Saleur and B. Derrida. "A combination of Monte Carlo and transfer matrix methods to study 2D and 3D percolation". In: *Journal de Physique* 46.7 (1985), pp. 1043–1057. DOI: 10.1051/jphys:019850046070104300. URL: https://doi.org/10.1051%5C%2Fjphys%5C%3A019850046070104300.

[19] Alessandra Cipriani and Bart van Ginkel. "The discrete Gaussian free field on a compact manifold". In: *Stochastic Processes and their Applications* 130.7 (2020), pp. 3943–3966. ISSN: 0304-4149. DOI: https://doi.org/10.1016/j.spa.2019.11.005. URL: https://www.sciencedirect.com/science/article/pii/S0304414919301310.

[20] Alex Amenta. "Lecture Notes: Fourier Analysis". In: (2019).

[21] Holden Lee. "Conjugate Gradient". In: (2020).

[22] Gérard Meurant and Zdenek Strakoš. "The Lanczos and conjugate gradient algorithms in finite precision arithmetic". In: *Acta Numerica* 15 (Jan. 2006), pp. 471–542. DOI: 10.1017/S096249290626001X.

[23] Tobin Fricke. *The Hoshen-Kopelman Algorithm*. Accessed: 2021-06-16.

[24] Heide H. Andersen et al. *Linear and Graphical Models*. Springer, 1995. ISBN: 978-1-4612-4240-6. DOI: 10.1007/978-1-4612-4240-6.

A

Some Notions on Real and Complex Gaussian Distributions

In Chapters 2 and 3, we more than often use Gaussian random vectors. Section 3.1.1 also deals with complex Gaussian vectors. Because we constantly work with them in this paper, several important properties of both real and complex Gaussian vectors are listed in this Appendix.

A.1. Gaussian Random Vectors

Although this has already been done in Definition 2.1.4 in the context of a Gaussian Free Field, let us again define when an N-dimensional Gaussian vector $x = (x_i)_{i=0}^N$ is Gaussian.

Definition A.1.1. Let $x = (x_i)_{i=0}^N$ be an N-dimensional random vector. We call x a Gaussian vector if for all vectors $c = (c_i)_{i=0}^N$, $c \cdot x := \sum_{i=1}^N c_i x_i$ is a Gaussian variable. Note that $c \cdot x$ could have zero variance.

Recall that a random variable X is Gaussian with mean $m \in \mathbb{R}$ and variance $\sigma^2 \in \mathbb{R}_{\geq 0}$ if it has the probability density function

$$f_X(u) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(u-m)^2}{2\sigma^2}}, \quad u \in \mathbb{R},\ \sigma^2 > 0. \tag{A.1}$$

If $\sigma = 0$, we speak of a degenerate Gaussian random variable, which is just the constant random variable $X = m$.

If x is a Gaussian vector, we denote this by $x \simeq \mathcal{N}(\mu, \Sigma)$. Here $\mu := \mathbb{E}[x] = (\mathbb{E}[x_i])_{i=1}^N$ is the mean of x and Σ is the covariance matrix of x. This $N \times N$ matrix is defined by its matrix elements $\Sigma(i,j) = \mathrm{Cov}(x_i, x_j) := E\big[(x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j])\big]$ for $i, j \in \{1, \ldots, N\}$. The covariance matrix is symmetric and positive semi-definite, as is shown in Theorem A.1.2.

Theorem A.1.2 (Properties of the covariance matrix)**.** The covariance matrix Σ, defined by $\Sigma(i,j) = \mathrm{Cov}(x_i, x_j)$ for $i, j \in \{1, \ldots, N\}$, is symmetric and positive semi-definite.

Proof. First note that $\Sigma(i,j) = \mathrm{Cov}(x_i, x_j) = \mathbb{E}\big[(x_i - \mathbb{E}[x_i])\big(x_j - \mathbb{E}[x_j]\big)\big] = \mathrm{Cov}(x_j, x_i) = \Sigma(j,i)$, so Σ is indeed symmetric.

To show positive semi-definiteness, recall that the variance is a non-negative quantity. In particular, for any constant vector c, $\mathrm{Var}(c \cdot x) \geq 0$. It is however also true that

$$\begin{aligned}
\mathrm{Var}(c \cdot x) &= \mathbb{E}\big[(c \cdot x - \mathbb{E}[c \cdot x])^2\big] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^N c_i x_i - \mathbb{E}\left[\sum_{i=1}^N c_i x_i\right]\right)^2\right] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^N c_i (x_i - \mathbb{E}[x_i])\right)^2\right] \\
&= \mathbb{E}\left[\sum_{i=1}^N \sum_{j=1}^N c_i c_j (x_i - \mathbb{E}[x_i])(x_j - \mathbb{E}[x_j])\right] \\
&= \sum_{i=1}^N \sum_{j=1}^N c_i c_j \mathrm{Cov}(x_i, x_j) = c \cdot \Sigma c.
\end{aligned}$$

Thus $c \cdot \Sigma c \geq 0$ for any c. This means that Σ is positive semi-definite. □

Just as for the Gaussian random variable X, we would like to define a density function for the Gaussian random vector x. This is not always possible however, as can be seen from Theorem A.1.3.

Theorem A.1.3. The Gaussian random vector $x \simeq \mathcal{N}(\mu, \Sigma)$ has a density on $\mathbb{R}^n$, given by (A.2), if and only if its covariance matrix Σ is regular, i.e. $\det \Sigma \neq 0$.

$$f_x(u) = \frac{1}{(2\pi \det \Sigma)^{N/2}} e^{-\frac{1}{2}(u-\mu)\cdot\Sigma^{-1}(u-\mu)}. \tag{A.2}$$

Proof. See Jacod and Protter, Proof of Corollary 16.2 [11]. □

Throughout this project, we often wish to sample a Gaussian vector with a certain covariance matrix C. To do that, we first sample a Gaussian $x \simeq \mathcal{N}(0, I_N)$ and then multiply it with some matrix A. This matrix A satisfied $AA^T = C$. The random vector Ax then turns out to have a normal distribution $\mathcal{N}(0, C)$. The reason this works is Theorem A.1.4.

Theorem A.1.4. Let $x \simeq \mathcal{N}(\mu, \Sigma)$ be a Gaussian vector. Then the random vector $y := Ax + b$ is again a Gaussian vector with mean $A\mu + b$ and covariance matrix $A\Sigma A^T$, i.e. $y \simeq \mathcal{N}(A\mu + b, A\Sigma A^T)$.

We will prove this theorem using characteristic functions. Lemma A.1.5 will be of help.

Lemma A.1.5. The random vector x is Gaussian with mean μ and covariance matrix Σ if and only if its characteristic function is given by:

$$\phi_x(u) := \mathbb{E}\left[e^{\imath u\cdot x}\right] = e^{\imath u\cdot\mu - \frac{1}{2}u\cdot\Sigma u} \tag{A.3}$$

Proof. See Jacod and Protter, Proof of Theorem 16.1 [11]. □

Proof of Theorem A.1.4. Suppose $x \simeq \mathcal{N}(\mu, \Sigma)$ and let $y = Ax + b$. The characteristic function of y is found to be:

$$\begin{aligned}
\phi_y(u) &= \mathbb{E}\left[e^{\imath u\cdot(Ax+b)}\right] \\
&= e^{\imath u\cdot b}\mathbb{E}\left[e^{\imath(A^T u)\cdot x}\right] \\
&= e^{\imath u\cdot b}\phi_x(A^T u) \\
&\overset{(A.3)}{=} e^{\imath u\cdot b} e^{\imath(A^T u)\cdot\mu - \frac{1}{2}(A^T u)\cdot\Sigma(A^T u)} \\
&= e^{\imath u\cdot b} e^{\imath u\cdot A\mu - \frac{1}{2}u\cdot A\Sigma A^T u} \\
&= e^{\imath u\cdot(A\mu+b) - \frac{1}{2}u\cdot(A\Sigma A^T)u}
\end{aligned}$$

By uniqueness of characteristic functions (Theorem 14.1 in [11]), we conclude that $y \simeq \mathcal{N}(A\mu + b, A\Sigma A^T)$. □

A.2. The Complex Normal Distribution

In Section 3.1.1, we introduced the complex Gaussian vector $z = x + \iota y$, where x and y were independent and identically $\mathcal{N}\left(0, \frac{1}{2} I_N\right)$-distributed Gaussian vectors. Such complex random variables (and vectors) are similar to real random variables in many ways, however they do deserve a small discussion. Therefore, we first formally introduce complex random variables and complex random vectors. After that we will take a closter look at complex Gaussian vectors, to finish off with Theorems A.2.7 and A.2.8. These are the main results that we use in Section 3.1.1. Our discussion is based on the book of Andersen, Hojbjerre, Sorensen and Eriksen [24].

First of all, note that we again use the symbol ι to denote the imaginary unit. In this discussion we work on the space $\mathbb{C}^N$ of complex N-dimensional vectors. If $c \in \mathbb{C}^N$, then we can write $c = a + \iota b$, where $a, b \in \mathbb{R}^N$. It can easily be seen that this yields an isomorphism $[\cdot] : \mathbb{C}^N \to \mathbb{R}^{2N}$, defined by: $[c] = \left(a\ b\right)^T$. We may also write a complex matrix $C \in \mathbb{C}^{N \times N}$ as $C = A + \iota B$, where $A, B \in \mathbb{R}^{n \times n}$ are real matrices. Furthermore, we define the partitioned matrix $\{C\} \in \mathbb{R}^{2N \times 2N}$ as:

$$\{C\} = \{A + iB\} = \begin{pmatrix} A & -B \\ B & A \end{pmatrix}$$

It is easy to see that $[Cc] = \{C\}[c]$. Finally, we work with the standard inner product on $\mathbb{C}^N$. That is, for $a = (a_i)_{i=1}^N$, $b = (b_i)_{i=1}^N$ in $\mathbb{C}^N$ we define $a \cdot b := \sum_{i=1}^N a_i \overline{b_i} = b^* a$ as the inner product.

With this small toolset, we are able to now define complex random variables.

Definition A.2.1 (Complex random variable)**.** Let X, Y be real random variables. Then the random variable given by $Z = X + \iota Y$ is a complex random variable.

We will only consider complex random variables in $\mathcal{L}^2(\mathbb{C})$, i.e. complex random variables with $\mathbb{E}[Z\overline{Z}] = \mathbb{E}[|Z|^2] < \infty$, where $\mathbb{E}$ is of course the expectation operator of a real random variable. For complex random variables, we introduce three important operators.

Definition A.2.2 (Operators of complex random variables)**.** Let $Z = X + \iota Y$ be a complex random variable. Then we define the expectation operator, $\mathbb{E} : \mathcal{L}^2(\mathbb{C}) \to \mathbb{C}$, of Z by $\mathbb{E}[Z] = \mathbb{E}[X] + \iota\mathbb{E}[Y]$. Now also let W be a complex random variable. We define the covariance operator, $\mathrm{Cov} : \mathcal{L}^2(\mathbb{C}) \times \mathcal{L}^2(\mathbb{C}) \to \mathbb{C}$, of Z and W by $\mathrm{Cov}(Z, W) = \mathbb{E}\left[(Z - \mathbb{E}[Z])\overline{(W - \mathbb{E}[W])}\right]$. Finally, we denote by $\mathrm{Var} : \mathcal{L}^2(\mathbb{C}) \to \mathbb{R}_{\geq 0}$ the variance of Z, defined by $\mathrm{Var}(Z) = \mathrm{Cov}(Z, Z)$.

We can generalise the previous notions to multidimensional objects, called complex random vectors.

Definition A.2.3 (Complex random vector)**.** We call the N-dimensional vector $x = (x_i)_{i=1}^N$ an N-dimensional complex random vector whenever x_i is a complex random variable for $1 \leq i \leq N$.

For these complex random vectors, we can again define expectation, covariance and variance operators, just as is done for real random vectors. Moreover, we can define a so-called complex covariance structure. Immediately after that we can finally define univariate complex normal distributions.

Definition A.2.4 (Complex covariance structure)**.** Let z be an N-dimensional complex random vector. The $2N$-dimensional random vector $[z]$ has a complex covariance structure if there exist matrices $\Sigma, A \in \mathbb{R}^{N\times N}$ such that

$$\operatorname{Var}([z]) = \begin{pmatrix} \Sigma & -A \\ A & \Sigma \end{pmatrix}.$$

Definition A.2.5 (Univariate complex normal distribution)**.** Consider a complex random variable $Z = X + \iota Y$. Then Z has a univariate standard complex normal distribution if and only if

(i) $[Z]$ has a bivariate normal distribution on $\mathbb{R}^2$.
(ii) $[Z]$ has a complex covariance structure.
(iii) $\mathbb{E}[Z] = 0$ and $\operatorname{Var}(Z) = 1$.

That is, $[Z] \simeq \mathcal{N}\left(0, \frac{1}{2} I_2\right)$. We write that $Z \simeq \mathcal{N}_c(0,1)$. Now, if $W = \mu + \sigma Z$ is a complex random variable, with $\mu \in \mathbb{C}$ and $\sigma \in \mathbb{R}_{\geq 0}$, we say that W has a univariate complex normal distribution. We denote this by $W \simeq \mathcal{N}_c(\mu, \sigma^2)$.

It can be shown that $W \simeq \mathcal{N}_c(\mu, \sigma^2)$ if and only if $[W] \simeq \mathcal{N}\left([\mu], \frac{\sigma^2}{2} I_2\right)$. Using this and Lemma A.1.5 we can first write down the characteristic function for $[W]$ and then for W itself:

$$\phi_{[W]}([u]) := \mathbb{E}\left[e^{\iota [u]\cdot[W]}\right] = e^{\iota [u]\cdot[\mu] - \frac{1}{2}[u]\cdot \frac{\sigma^2}{2} I_2 [u]} = e^{i[u]\cdot[\mu] - \frac{\sigma^2}{4}[u]\cdot I_2 [u]}, \quad \text{for } [u] \in \mathbb{R}^2.$$

The reader can convince him- or herself that this is equivalent to:

$$\phi_W(u) := \mathbb{E}\left[e^{\iota \operatorname{Re}(\overline{u} W)}\right] = e^{\iota \operatorname{Re}(\overline{u}\mu) - \frac{\sigma^2}{4}\overline{u} u}, \quad \text{for } u \in \mathbb{C}. \tag{A.4}$$

We can now define complex normal vectors, in a similar fashion to Definition A.1.1.

Definition A.2.6 (Complex Gaussian vector)**.** An N-dimensional complex random vector $z = (z_i)_{i=1}^N$ has a complex normal distribution if for all $c = (c_i)_{i=1}^N \in \mathbb{C}^N$, $c \cdot z$ has a univariate complex normal/Gaussian distribution.

From Definition A.2.6 it follows that each z_i is a complex normal random variable. Consequently, $\mathbb{E}[z_i]$ and $\operatorname{Var}(Z_i)$ exist and are finite, such that $\mu := \mathbb{E}[z] \in \mathbb{C}^N$ and $\Sigma := \operatorname{Var}(z) \in \mathbb{C}^{N\times N}$ exist. We write $z \simeq \mathcal{N}_c(\mu, \Sigma)$.

We are ready to prove our two main results, Theorem A.2.7 and Theorem A.2.8. Similar to the proof of Theorem A.1.4, we will make use of characteristic functions for the first of the two. We can generalise (A.3) to the multidimensional case to find that for a complex Gaussian vector z with mean μ and covariance matrix Σ:

$$\phi_z(u) := \mathbb{E}\left[e^{\iota \operatorname{Re}(u^* z)}\right] = e^{\iota \operatorname{Re}(u^*\mu) - \frac{1}{4} u^* \Sigma u}, \quad \text{for } u \in \mathbb{C}^N. \tag{A.5}$$

Theorem A.2.7. Let $z \simeq \mathcal{N}_c(\mu,\Sigma)$, let $b \in \mathbb{C}^N$ and let $A \in \mathbb{C}^{N\times N}$. Then the random vector $w := Az + b$ has distribution $\mathcal{N}_c(A\mu + b, A\Sigma A^*)$.

Proof. Again, we make use of characteristic functions. Suppose that $z \simeq \mathcal{N}_c(\mu,\Sigma)$ and let $w = Az + b$. We compute $\phi_w(u)$ for $u \in \mathbb{C}^N$, just as before:

$$\begin{aligned}\phi_w(u) &= \mathbb{E}\left[e^{\iota \mathrm{Re}(u^*(Az+b))}\right]\\ &= e^{\iota\mathrm{Re}(u^*b)}\mathbb{E}\left[e^{\iota\mathrm{Re}((A^*u)^*z)}\right]\\ &= e^{\iota\mathrm{Re}(u^*b)}\phi_x(A^*u)\\ &\overset{(A.5)}{=} e^{\iota\mathrm{Re}(u^*b)}e^{\iota\mathrm{Re}(u^*A\mu)-\frac{1}{4}u^*A\Sigma A^*u}\\ &= e^{\iota\mathrm{Re}(u^*(A\mu+b))-\frac{1}{4}u^*(A\Sigma A^*)u}\end{aligned}$$

Also for complex random vectors, we have uniqueness of characteristic functions (Theorem 1.9 in [24]), so $w \simeq \mathcal{N}_c(A\mu + b, A\Sigma A^*)$, as required. □

At a certain point in Section 3.1.1, we take the real part of some complex Gaussian vector w. We only know that $w \simeq \mathcal{N}_c(\mu,\Sigma)$. Can we retrieve the distribution of $\mathrm{Re}(w)$ from this? Theorem A.2.8 will help us answer this question.

Theorem A.2.8. An N-dimensional complex random vector w has a normal distribution $\mathcal{N}_c(\mu,\Sigma)$ if and only if $[w]$ has distribution $\mathcal{N}\left([\mu], \frac{1}{2}\{\Sigma\}\right)$.

Proof. Suppose that $w \simeq \mathcal{N}_c(\mu,\Sigma)$. Then we can find a matrix A such that $AA^* = \Sigma$. If we let $z \simeq \mathcal{N}_c(0, I_N)$, then Theorem A.2.7 tells us that w and $Az + \mu$ have the same distribution. Consequently, $[w]$ and $[Az+\mu] = \{A\}[z] + [\mu]$ also have the same distribution. Now, if we generalise our earlier remark right under Definition A.2.5 to the multidimensional case, we can easily see that:

$$z \simeq \mathcal{N}_c(0, I_N) \text{ if and only if } [z] \simeq \mathcal{N}\left(0, \frac{1}{2}I_{2N}\right).$$

By Theorem A.1.4, we know that $\{A\}[z] + [\mu]$ and so $[w]$ as well have distribution $\mathcal{N}\left([\mu], \{A\}\frac{1}{2}I_{2N}\{A\}^T\right) = \mathcal{N}\left([\mu], \frac{1}{2}\{A\}\{A\}^T\right)$. We claim however that $\{A\}\{A\}^T = \{\Sigma\}$ as $\Sigma = AA^*$. But then $[w] \simeq \mathcal{N}\left([\mu], \frac{1}{2}\{\Sigma\}\right)$. Since the converse is analogous, this is the result we wanted. It only remains to show our claim. To this end, we write $A = B + \iota C$. This implies that $\Sigma = AA^* = (B+\iota C)(B-\iota C) = (B^2 + C^2) + \iota(CB - BC)$. Now observe that:

$$\{A\}\{A\}^T = \begin{pmatrix} B & -C \\ C & B\end{pmatrix}\begin{pmatrix} B & C \\ -C & B\end{pmatrix} = \begin{pmatrix} B^2+C^2 & BC-CB \\ CB-BC & B^2+C^2\end{pmatrix} = \Sigma,$$

which finishes the proof. □

Determining the distribution of the real part of some zero-mean complex Gaussian vector with a real covariance matrix is now really just a special case of Theorem A.2.8.

Corollary A.2.9. Let $z \simeq \mathcal{N}_c(0, \Sigma)$ be an N-dimensional complex Gaussian vector, with Σ a real covariance matrix. Then $\mathrm{Re}(z) \simeq \mathcal{N}\left(0, \frac{1}{2}\Sigma\right)$.

Proof. Note that $\mathrm{Re}(z)$ are the first N entries of $[z]$. By Theorem A.2.8, and since Σ is real, we know that:

$$\begin{pmatrix} \mathrm{Re}(z) \\ \mathrm{Im}(z) \end{pmatrix} = [z] \simeq \mathcal{N}\left(0, \frac{1}{2}\begin{pmatrix} \Sigma & 0 \\ 0 & \Sigma \end{pmatrix}\right).$$

This implies that $\mathrm{Re}(z) \simeq \mathcal{N}\left(0, \frac{1}{2}\Sigma\right)$, as required. $\square$

B
The Project's Github Page

This Appendix contains a reference to the Github repository of this project. Moreover, some explanation of the most important scripts and functions can be found here.

SEVERAL lines of code have been written during the course of this project. In this Appendix, we will briefly cover the most relevant scripts and commands. The interested reader can find all of the Python code in the Github repository `modGFF`. This repository can be accessed via the link

`https://github.com/PimKeer/modGFF`.

CONDUCTANCES.PY

In order to generate the checkerboard pattern as in Figure 2.4 for the conductances on an arbitrary 3D lattice with linear size N (with N a multiple of 10), we created the function `wxyz(N, a, b)`. Here `N` is the linear size of the lattice, and `a`, `b` are the respective values of the two types of conductances. Each lattice point i is linked with three conductances. These conductances correspond to the edges containing i in the positive x-, y- and z- directions. The function returns three arrays, `wx`, `wy` and `wz`. These arrays all have length $(N+2)^3$. At index i, each array holds the conductance in either the x-, y- or z-direction associated with lattice point i. This is illustrated in Figure B.1. Note that the boundary points are also included in these arrays (explaining the +2). Also note that we are using the same indexing convention as described in Section 3.3.1, but now including the boundary points.

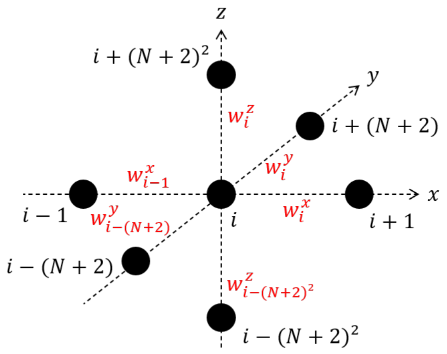


Figure B.1: The lattice site i and its nearest-neighbours, with corresponding conductances. The indices are in black, the conductances are in red. Notice that we are working in an $(N+2) \times (N+2) \times (N+2)$ lattice, i.e. the (zero) boundary is included.

MATRICES.PY

This script contained all functions that perform matrix-vector multiplications. They all work out the product Ax, where x is the input vector and A the matrix of interest. The advantage of working with a function specifying the matrix-vector multiplication instead of the whole matrix is that we only need one vector. In a lattice of linear size N, the precision matrices have dimensions $N^3 \times N^3$. For large N this is almost impossible to store. It is also a waste, as most matrix entries are often zero. This is certainly the case in this project, as we are only considering nearest-neighbour interactions. The most important matrix-vector product function is `C(u, wx, wy, wz, N)`. It takes as input an $(N+2)^3$-dimensional vector `u`, three $(N+2)^3$-dimensional conductance vectors `wx`, `wy` and `wz` and the linear lattice size N. The vector x is zero on the boundary. The function `C()` returns the matrix-vector product of Cx, with $C = \frac{1}{2d}\Delta_\Lambda$ the precision matrix for the DGFF found in Section 2.1.2. This is written down in Algorithm 3. The same indexing is used as in Figure B.1.

Algorithm 3: Matrix vector multiplication with $C = \frac{1}{2d}\Delta_\Lambda$.

```
Input: a linear lattice size $N$, an $(N+2)^3 \times 1$ vector $u$ that is zero on the
        boundary and $(N+2)^3 \times 1$ vectors $w^x$, $w^y$ and $w^z$.
Output: $(N+2)^3 \times 1$ vector $v = Cu$ with zero boundary.
for $i \in \{0,1,\dots,(N+2)^3\}$ do
    if $i$ is the index of a boundary point then
        set $v_i = u_i = 0$;
    end
    else
        set $v_i$ equal to:
            $$v_i = \frac{1}{6}u_i\Big(w_i^x + w_{i-1}^x + w_i^y + w_{i-(N+2)}^y + w_i^z + w_{i-(N+2)^2}^z\Big)$$
            $$- \frac{1}{6}u_{i+1}w_i^x - \frac{1}{6}u_{i-1}w_{i-1}^x - \frac{1}{6}u_{i+(N+2)}w_i^y - \frac{1}{6}u_{i-(N+2)}w_{i-(N+2)}^y$$
            $$- \frac{1}{6}u_{i+(N+2)^2}w_i^z - \frac{1}{6}u_{i-(N+2)^2}w_{i-(N+2)^2}^z;$$
    end
end
```

CLUSTER.PY

The `cluster.py` script contains our implementation of the Hoshen-Kopelman algorithm, as discussed in Section 3.3.1. The function `cluster(x, N)` needs a vector `x` of size N^3 consisting of zeros and ones. This vector represents a lattice of linear size N. If site i in this lattice is occupied, then the i-th entry of `x` equals 1, and 0 otherwise. As output, `cluster()` gives an array of size N^3, containing the cluster labels for each lattice site.

GENGFFSHEFFIELD.PY

The validity checks of the Hoshen-Kopelman algorithm and the R_N-computation were performed by generating DGFFs with periodic boundary conditions. This was done using the script `genGFFSheffield.py`, which contains functions for creating 2D and 3D samples of a DGFF on an arbitrary $m \times n$ or $m \times n \times o$ lattice respectively. These samples are based on the methods from Section 3.1.1.

GENGFFCHAFAI.PY

This is a very similar script to `genGFFSheffield.py`, except that one can create DGFF samples with zero boundary conditions here. This is done using the method treated in Section 3.1.2. With the function `savesqrtG(N)`, the script allows to compute the matrix $\Sigma_\Lambda^{1/2}$ for $\Lambda = \{1, \ldots, N-1\}^3$. This matrix can be stored and accessed later on. We can then generate as many samples as we want by employing (3.4). This is done with the command `genGFFChafai2(N,sG)`, where N is the linear lattice size and sG is the matrix $\Sigma_\Lambda^{1/2}$, computed beforehand.

CG.PY

The `cg.py` script is the heart of this project. In here, one can find the function `cgpf0(C, wx, wy, wz, N, epsilon, kmax)`. It returns an $(N+2)^3$ array containing an approximate sample of a $\mathcal{N}\left(0, C^{-1}\right)$-distribution on an $N \times N \times N$ lattice, with zero boundary conditions. To do this, `cgpf0()` follows Algorithm 1. The function has a number of arguments. First of all, `C` is the routine working out matrix-vector products, where the matrix is the precision matrix from which we should sample. In the setting of this project, `C` was set to $\frac{1}{2d}\Delta_\Lambda$ most of the time. The arrays `wx`, `wy` and `wz` again contain all the conductances in the lattice. `N` is the linear lattice size. The function `cgpf0()` accommodates two types of stopping criteria. The first one is the stopping tolerance `epsilon`. Once $||r^k||_2 <$ `epsilon`, the algorithm stops and returns the latest approximation. The second one, `kmax`, specifies a maximum amount of iterations. Once the algorithm has executed this many iterations, it stops as well. Important to note is that the output of `cgpf0()` also contains the zero boundary. We are often only interested in the inner $N \times N \times N$ region of the lattice. To this end, `cg.py` also contains the function `cut(x, N, n)`. This routine removes the outer `n` layers from an `N` × `N` × `N` array `x`.

THE MAIN_.PY SCRIPTS

Finally, there are a number of scripts to run specific simulations. Most of these perform the computation of R_N to find the percolation threshold. These scripts contain some variant of the function `gamma(x, h, N)`. Given some $N \times N \times N$ random vector `x` (in our case a realisation of a DGFF), this function performs two tasks. First, it determines the level-set with threshold `h`, as defined in Section 2.2. Second, it computes the empirical second moment of the cluster size $N^3 \sum_s s^2 n_s$. This is in fact the quantity Γ_N defined in Section 3.3.2, but as an average over one sample. The name of these scripts is given by `main_` followed by the name of the simulation (such as `checkerboard` for the runs with checkerboard conductances, or `cgbenchmark` to compare the CG sampler with Chafaï's method.