Structure Function Relation of Porous 2D Material via SGCMC Simulation and Statistical Models

by

Athikom Wanichkul

B.S. in Civil and Environmental Engineering
B.S. in Electrical Engineering and Computer Science
Massachusetts Institute of Technology, 2022

Submitted to the Department of Civil and Environmental Engineering
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN CIVIL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

© 2024 Athikom Wanichkul. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Athikom Wanichkul
Department of Civil and Environmental Engineering
May 10, 2024

Certified by: Franz-Josef Ulm
Professor of Civil and Environmental Engineering, Thesis Supervisor

Accepted by: Heidi Nepf
Donald and Martha Harleman Professor of Civil and Environmental Engineering
Chair, Graduate Program Committee

Structure Function Relation of Porous 2D Material via SGCMC Simulation and Statistical Models

by

Athikom Wanichkul

Submitted to the Department of Civil and Environmental Engineering
on May 10, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN CIVIL ENGINEERING

ABSTRACT

To improve the design for structural resilience and reduced environmental impact, we need to make the structure function relation of concrete more accurate, accessible, and cost-effective. First, we formulate and implement the Semi-Grand Canonical Monte Carlo (SGCMC) simulation for fracture mechanics, which is a stochastic method that is capable of capturing both the initiation and the propagation of fractures in a medium. We then optimize the performance of our SGCMC simulation to reduce its time complexity from $O(n^{2.38})$ to $O(n^{1.24})$ and its space complexity from $O(n^2)$ to $O(n)$. The key step to performance optimization is exploiting the sparsity of the stiffness matrix. We also deploy our code to run multiple simulations concurrently on a super-computing infrastructure to achieve scalability. Then, we try to achieve an even more accessible and cost-effective structure function relation by applying statistical modeling to predict the strength of a two-dimensional porous material without running the simulation. We generate samples by randomly placing circular pores with radii drawn from a log-normal distribution until we reach the target porosity and run our SGCMC simulations on the generated samples to create a data set to train our statistical models. We defined several parameters, including the two-point correlation function, the multi-scale disorder index, the distribution of pore radius as recovered by Circle Hough Transformation (CHT), and the area moments of the pores to parameterize the porous geometry of the samples beyond the porosity, which is a well-known and very important parameter. We found our best model to be a Gradient Boosting Decision Trees (GBDT) regression model, whose out-of-sample R2 is 0.904, as opposed to the baseline model of linear regression with the porosity, whose out-of-sample R2 is 0.752.

Thesis supervisor: Franz-Josef Ulm
Title: Professor of Civil and Environmental Engineering

Acknowledgments

I thank my advisor, Franz-Josef Ulm, for his mentorship and his unwavering trust. I thank my colleague, Ariel Attias, for our discussions and for sharing our frustrations. I thank my favorite MIT CEE administrator, Sarah J Smith, for the snacks, the chats, the helps, and the dungeon runs in World of Warcraft. I thank my roommates, current and ex, Abhijatmedhi Chotrattanapituk, Pasapol Saowakon, and Krittamate Tiankanon, for the amazing friendship and all the consults. Most importantly, I thank my mother and my family for their unconditional love and support. You'll never walk alone. μ'sic forever ♪♩♩♩♩♩♩♩♪

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Concrete is one of the most, if not the most, widely used materials in modern construction, thanks to its structural and economic benefits as well as its availability [1], [2]. However, it is also a very energy-intensive material that accounts for approximately 7% of global carbon dioxide emissions [3]. Therefore, reducing the environmental impact of concrete is one of the potential advancements in concrete research [4]. One such way to reduce the environmental impact, as well as the economic cost, of concrete is to minimize the excess use of concrete due to the over-specification of the design while ensuring that the design is not under-specified and is capable of withstanding both the normal and extreme loads. The necessity of resilience is especially important today as global warming results in increased frequency and severity of these extreme load cases [5]. To achieve these goals, we need to make modeling of properties, especially the strength, of concrete more accurate, more accessible, and more cost-effective.

In the context of this thesis, we define the structure function relation as the relationship between the structure, such as the shape and geometry, and the properties (function), such as the ultimate strength, of materials. While it is possible to directly obtain this relationship by fabricating samples and physically running tests in a laboratory, such an approach can be very costly and time-consuming, especially for concrete as it needs time, up to over 180 days, to achieve its maximum strength [6], [7]. On the other hand, it is extremely difficult to analytically model the strength of concrete due to the complexity and uncertainty of concrete structure on multiple scales from the micro-scale to the meso-scale.

Therefore, scientists and engineers have developed empirical relationships as well as simulation algorithms that allow us to calculate and predict the properties of concrete at varying accuracy and computational cost. Currently, one of the most commonly used computational methods is the Finite Element Method (FEM). However, there are certain limitations to FEM that we would like to address. One such limitation is that FEM is a deterministic algorithm. Thus, it cannot capture the stochastic nature of fracture propagation. In addition, classical fracture mechanics in Griffith's sense requires a pre-existing micro-fracture to address the process of fracture propagation [8]. To overcome these issues, we seek to simulate the fracture mechanics through a stochastic method, such as the Semi-Grand Canonical

Monte Carlo (SGCMC) simulation so as to capture both the initiation and the propagation of fractures in the medium. Thus, the first part of this thesis involves implementing the SGCMC algorithm for fracture mechanics, which has been formulated in Ulm (2022) [9], and optimizing its performance as the SGCMC algorithm can be computationally expensive.

Once we achieve the structure function relation via SGCMC simulation, we further approach the structure function relation via various statistical models, ranging from simpler ones such as linear regression to more complex ones such as gradient boosting decision trees (GBDT). There are two benefits to this approach. Firstly, we can achieve a much cheaper and faster structure function relation at the cost of accuracy. Secondly, if we discover features that can be controlled in the fabrication process without incurring too much cost, we may be able to modify the fabrication process to produce stronger concrete at little additional cost. While unlikely, its impact could be significant.

For the second part of this thesis, we define the scope of our statistical modeling as the ultimate strength of two-dimensional porous homogeneous materials. We chose the ultimate strength as the target y-variable because it is a key property of materials essential to structural design. Accurate predictions of the ultimate strength can help us reduce the overspecification of the design and thus the unnecessary usage of materials. We are interested in porous materials because they are simple prototypes of heterogeneous materials such as concrete, while also being ubiquitous as they can represent a wide range of materials, ranging from filter papers to bones to concrete. As for the choice to focus on the two-dimensional case, we make this decision for the simplicity and the cost of computation.

It is a well-known fact that the key parameter of concrete's strength is porosity, which is directly linked to the water-to-cement ratio [7]. However, it is naturally impossible to describe the entire geometry of the pores with one single number, as shown in Laubie (2017) [10]. Hence, we aim to define and test additional texture parameters with our statistical model approach.

To create such statistical models, we first need to create a data set by running SGCMC simulations on a set of sample images, which we artificially generated from a prior distribution, since we couldn't obtain a data set of actual concrete images under a microscope. Once we have the data set, we can then train the various statistical models and compare their performance. While we have achieved some results with our statistical models, it is important to note that there is still a lot more potential with this method that we haven't captured due to our time constraints. However, we hope that this work demonstrates the potential of the approach that can be realized in the future.

1.2 Background

1.2.1 Concrete

One common misconception about concrete is that people often use the term cement and concrete interchangeably [11]. However, concrete is, in fact, a composite material where cement is mixed with water to create the binding phase that holds aggregates, which are typically sand and gravel, together to form concrete. When cement and water mix, they chemically react in a hydration reaction to form Calcium-Silica-Hydrates (C-S-H), which are the main contributors to the strength of cement-based materials. A concrete mix is initially a slurry and turns into a solid as it sets. However, even if the concrete is already solid, the cement inside the concrete can continue to hydrate and harden, albeit at a slower rate. Depending on the type of concrete, it can take 28 to 90 days, or even longer, to fully harden. [6], [7], [12]

Multi-Scale Mechanical Model of Concrete

We can consider concrete as a three-scale material. On the first and smallest level, we have the C-S-H gel, which consists of the C-S-H solid and the gel pores. Then, on the second level, we consider the hydration gel and the capillary pores caused by the surplus water not consumed in the hydration reaction. These capillary pores are detrimental to the strength of concrete due to the stress concentrations. Finally, on the third and largest level, the concrete is a heterogeneous mixture of the binding phase from the second level and the aggregates, which can be considered as rigid inclusions as their stiffness is significantly higher than that of the binding phase. Thanks to this multi-scale model of concrete, the analytical approach to the structure function relation of concrete can be very complicated and difficult [12], [13].

1.2.2 Stiffness Matrix

A stiffness matrix is a tool that helps us algebraically represent a system of element stiffness equations in the form of $F = Kd$. We derive the stiffness matrix for a two-dimensional system of elements that connect two nodes and only carry axial force (i.e., two-point interactions), such as a truss. Consider an element in its local coordinate system, its stiffness matrix is given by:

$$k = k_e \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{1.1}$$

where k_e is the stiffness of the element. We then transform this element stiffness matrix into the global coordinate system as:

$$K = T^T k T \tag{1.2}$$

where T is the transformation matrix, which is given by:

$$T = \begin{pmatrix} cos\theta & sin\theta & 0 & 0 \\ 0 & 0 & cos\theta & sin\theta \end{pmatrix} \tag{1.3}$$

We can then assemble the global stiffness matrix of the entire system using the element stiffness matrix of each element in the system [14].

1.2.3 Space and Time Complexity of Algorithms

In order to evaluate the performance of an algorithm, we need to understand how does its memory usage (space) and execution time scale with the size of the input. The O-notation is a common tool that is used to communicate the asymptotic upper bound of an algorithm. The formal definition of $O(g(n))$ is the set of functions $f(n)$ such that there exist positive constants c and n_0 such that $O \leq f(n) \leq cg(n)$ for all $n \geq n_0$. Therefore, the slower $g(n)$ grows with n, the better the $O(g(n))$ algorithm is [15].

1.2.4 Decision Tree Ensemble Models

A decision tree ensemble model is a machine learning model that uses, as stated in its name, an ensemble of decision trees to make predictions of the target y-variable and combines those predictions into a single final prediction. A decision tree is a model that makes a prediction based on what is essentially a set of `if-else` tests based on the input features in a flowchart-like structure. Therefore, decision tree models are easy to interpret and are capable of capturing non-linear relationships. In this thesis, we use random forest models and gradient boosting decision trees, which are specific types of decision tree ensemble models [16]–[18].

1.3 Research Questions

We formally define the following research questions to address in this thesis:

1. How can we capture the stochastic nature of fracture mechanics in a computational approach to the structure function relation of concrete?
2. How can we better parameterize the porous geometry of a two-dimensional sample beyond porosity?
3. How can we better model the structure function relation of concrete to bypass the computational method and simulations?

1.4 Thesis Overview

With our motivation, scope, and background knowledge defined in Chapter 1, we will begin discussing the first part of this thesis - the implementation and optimization of the SGCMC algorithm for fracture mechanics as a means of accurate and cost-effective structure function relation - in Chapter 2 and show our results in Chapter 3. We then address the second part of this thesis - an even more cost-effective structure function relation of two-dimensional porous material via statistical models - in Chapters 4 and 5. We use the SGCMC simulation we implemented in Chapter 2 to generate a data set in Chapter 4 and analyze the data set in Chapter 5 to create various statistical models. Finally, we discuss our contributions, limitations, and future suggestions in Chapter 6.

Chapter 2

Semi-Grand Canonical Monte Carlo (SGCMC) Simulation

In this chapter, we discuss the formulation, implementation, and optimization of the Semi-Grand Canonical Monte Carlo (SGCMC) simulation. In Section 2.1, we discuss the theory behind and the formulation of the SGCMC algorithm from the literature. We then discuss the design choices and some interesting details of our implementation of the algorithm in Section 2.2. Finally, we discuss in Section 2.3 the performance optimizations that make the SGCMC simulation run faster and use less memory.

2.1 Formulation of SGCMC Simulation

Following Ulm (2022) and Mulla (2020), we formulate the Semi-Grand Canonical Monte Carlo (SGCMC) simulation by considering the material sample as an ensemble of bonds where each bond has two potential states: *broken* (0) and *unbroken* (1). For each step of the Monte Carlo trial (MC trial), we consider the system frozen (i.e., each change in the bond state does not impact the particle position) and randomly pick a bond from the ensemble (with equal probability i.e., uniformly random) and switch its state, from the old state (denoted as o) to the new state (denoted as n), with probability:

$$P(o \to n) = \frac{P(n)}{P(o)} = \frac{\exp\left(-\beta u(i)[n]\right)}{\exp\left(-\beta u(i)[o]\right)} = \exp\left(-\beta \Delta u(i)\right) \leq 1 \tag{2.1}$$

where $\beta^{-1} = k_B T \propto E_k$ and $\Delta u(i) = u(i)[n] - u(i)[o]$ are the kinetic temperature and the change in potential energy as a result of the proposed state change of the randomly picked i-th bond in the ensemble, respectively. We can decompose the potential energy of a bond as:

$$u(i) = u_0(i) + u_\lambda(i) \tag{2.2}$$

where $u_0(i)$ and $u_\lambda(i)$ are the bond's ground-state energy and the bond's elastic energy, respectively. Since we keep the number of bonds constant by performing the MC trial on the state of a bond instead of randomly inserting (or deleting) a particle, our ensemble is, by definition, a semi-grand canonical ensemble.

Then, for every N, the number of bonds in the system, MC trials, we update the load on the system, if specified, as well as the (now unfrozen) position of particles in the system $x = x_0 + \xi$ where ξ is the displacement from the initial position. There are multiple ways to achieve this update, as discussed in Ulm (2022). For this thesis, we have chosen the lattice element method where we discretize the sample as a regular lattice of particle nodes with bonds connecting each particle node with its neighbor particle nodes within a shell radius and solve for

$$K\xi = F \tag{2.3}$$

where K and F are the stiffness matrix and the force vector, as derived in Section 1.2.2. We call this step the NVT update and we alternate between the MC trials and NVT update until we achieve the target number of NVT updates. [9], [19].

2.2 Implementation of SGCMC Simulation

We implement the SGCMC algorithm as formulated above in Section 2.1 in MATLAB [20], with some starting code from my thesis advisor. We choose to use MATLAB as our programming language in this section for the purpose of compatibility with my advisor and my colleague.

2.2.1 Interface Definition

The first step to implementing the SGCMC simulation is to make the important design decisions that will govern our implementation process. One such decision is the specification of our SGCMC code, which we package as a single function that can be called from any pipeline. While it is obvious that the output of this function should be the result of the SGCMC simulation of the input, it is important that we properly define the interface through which our function interacts with its caller such that it accommodates all the necessary information in the input and output, both in the present and in the foreseeable future, while being easy to understand and use.

Thus, we define the inputs of our SGCMC function to include the information needed to define a simulation problem as follows:

- The sample's lattice, given by:
  - The gray-scale image of the sample's porous geometry (I) such that the pixel value of 1 corresponds to a pore and 0 corresponds to the material medium
  - The threshold of pixel value that constitutes a pore (t_H)
  - The number of particle nodes per sample side (N_x, N_y)
  - The radius of each particle node (R)
  - The number of interaction shells
  - The magnitude of particle node perturbation

- The number of NVT updates
- The prescribed load, defined as the strain tensor components (ε_{xx}, ε_{yy}, and ε_{xy}), for each NVT update
- The boundary conditions
- The stiffness and the ground-state energy of the bonds
- The kinetic temperature

Similarly, we define the outputs of our SGCMC function to include the following information, written to a result file:

- The entire stress-strain curve from every NVT update
- The problem statement and the initial lattice for convenience
- The snapshots of intermediate results for diagnostic purpose
- The success flag, if the algorithm ran to completion without any exceptions, or the thrown exception object

2.2.2 Internal Representation

Next, we have to consider the internal representation of our sample lattice. Since we are only considering two-point interaction in our current model, we can consider a lattice as an undirected graph where vertices represent particles and edges represent the bond. We represent this graph with the sparse version of the adjacency matrix by keeping track of the bonds connecting two particle nodes. For each bond, we have to keep track of the two nodes it connects and the state (broken or unbroken), the length, the stiffness, and the energy of the bond. While the state, the length, and the energy of the bond may change as the simulation progresses, the end nodes will remain constant throughout the entire process. In addition to the bond representations, we also need to keep track of the location, represented by x and y coordinates, of every particle node.

2.2.3 Lattice Creation

To initialize a lattice, we first consider the most basic case – a dense regular lattice with a square unit cell. In this case, we can set up a grid of regularly spaced particle nodes and add a bond for every pair of adjacent nodes. This method is simple and efficient and results in a lattice similar to an x-braced structure as shown in Fig. 2.1 below. However, in order to simulate the fracture of a porous material, we need to create the pores by removing certain particle nodes from the lattice. As defined in the interface in Section 2.2.1, we take in an image as an input to specify the porous geometry of our sample lattice. Consider an N_x-by-N_y gray-scale image (same resolution as our lattice discretization). It is easy to mask our particle nodes with this image such that a particle, along with the bonds connected to it, is removed if the value of its corresponding pixel is higher than the specified threshold

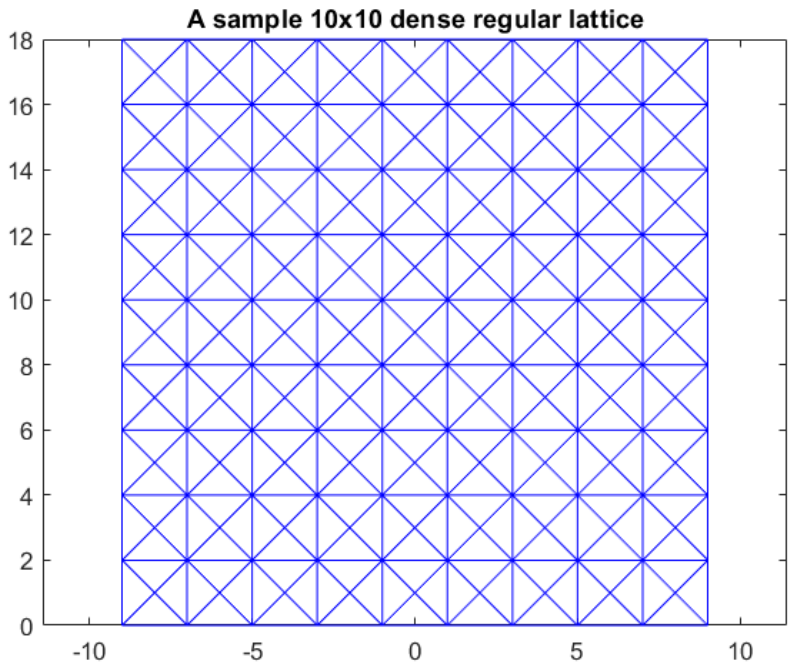

Figure 2.1: A sample 10x10 dense regular lattice with 1 interaction shell.

from Section 2.2.1 (i.e., $I_{x,y} > t_H$). If the input image is not of N_x-by-N_y resolution, we need to re-scale it to match our lattice discretization. One way to achieve this is by using MATLAB's `imresize` function. This input image can either be taken from actual samples under a microscope or generated from a prior distribution. We will further discuss this image generation process in Section 4.1.

However, once we include additional optional parameters, such as random perturbation of the nodes, the additional shell of interaction, and the future accommodation for different unit cells, our lattice generation can become complex. Thus, we make a design decision to prioritize the correctness and generality of the lattice generation algorithm at the cost of performance, especially since the lattice is generated only once per simulation and only takes up a small portion of the total computation time. As such, we opt to use the K-nearest neighbor (KNN, implemented via MATLAB's `knnsearch`) algorithm, as well as a maximum initial bond length based on the number of interaction shells, to determine the bond interactions between particle nodes.

2.2.4 MC Trials

The implementation of MC trials is, while important, a straightforward endeavor that does not leave much to be discussed. As formulated in Section 2.1, for each MC trial, we:

- Pick a bond by randomizing the index of the bond (i)
- Store the old bond energy in a temporary variable

- Switch the bond state
- Calculate the new bond energy
- Calculate the energy change and thus the probability (`p`) of accepting this MC trial
- Accept the new bond state if `rand < p`, otherwise restore the old bond state in the internal representation

Since we formulated in Section 2.1 that $P(o \to n) = \exp(-\beta \Delta u(i)) \leq 1$, we theoretically need to cap the probability `p = min(p, 1)`. However, since `rand` uniformly randomizes a value from the interval $(0, 1)$, this step is already implicitly handled in our implementation.

One interesting detail of our implementation is our decision to calculate the bond length l and the elastic bond energy $u_\lambda(i)$ of every bond during the NVT update. The rationale is to future-proof our implementation for additional logic, such as a critical bond length l_c where the bond breaks or goes into plastic deformation. Since the position of the particle nodes is frozen during MC trials, these stored values remain correct until the next NVT update, where they are re-calculated. This design choice also has a performance benefit, which will be further discussed in Section 2.3.

2.2.5 NVT Updates

After every N MC steps, we perform an NVT update. As formulated in Section 2.1, we need to solve for the displacement vector $\xi = K^{-1}F$. However, it is recommended in MATLAB documentation that `xi = K \ F`, which is implemented by function `mldivide`, is used instead of `xi = inv(K) * F`, for better accuracy and computational speed [21]. One key change we made to the NVT update from our formulation is that, instead of solving directly for the displacement vector ξ, we solve for the incremental displacement (the change in displacement vector since the previous NVT update) from the incremental load (the change in specified strain since the previous NVT update). The idea for this change is to bridge between two equilibrium states from the previous NVT update.

A caveat arises when the stiffness matrix K has a high condition number, which is defined as the sensitivity of the solution to the input of the system of equations. Thus, we propose the following minimization problem when our stiffness matrix is ill-conditioned:

$$\begin{aligned} \min_{\xi} \quad & \frac{1}{2}\xi^T K \xi + F^T \xi \\ \text{s.t.} \quad & K\xi = F \\ & \xi_{min} \leq \xi \leq \xi_{max} \end{aligned} \tag{2.4}$$

The idea of this minimization problem is to solve the original system of equations $K\xi = F$ while minimizing the incremental potential energy to obtain the most stable solution when the system of equations has multiple answers within the precision range. We can solve this quadratic minimization problem using the function `quadprog` from MATLAB's optimization toolbox [22]. In addition to the benefits above, this minimization problem also allows us to implicitly handle boundary conditions, if not explicitly specified in the inputs.

Another caveat is that we sometimes obtain infinity `inf` or undefined `NaN` values in our result. Our troubleshooting suggests that these invalid values arise when bonds break and a non-empty set of particle nodes are disconnected from the rest of the lattice, which is bounded by the boundary conditions. As a result, these broken-off chunks become free from the rest of the system. We address this issue by adding sufficiently large but finite bounds to the acceptable displacement vector as $\xi_{min} \leq \xi \leq \xi_{max}$. Adding this bound solved our issue with unbounded solutions.

Finally, after we solve for the displacement vector, we update our particle positions, as well as the length and the elastic energy of every bond in the ensemble.

2.2.6 Result Management

The last key to our implementation of the SGCMC simulation lies in how we manage and save our hard-earned results. To do so, we decided to create a `result_manager` class to handle everything related to our function output. We instantiate the object at the start of the simulation to record the inputs, the initial lattice, and the problem definition so that our result files are standalone and easy to process. Then, the most important part of our result is the overall stress tensor on the lattice, as calculated from the virial theorem, on every NVT step. Combined with the strain definition in the input, we can plot the stress-strain curve of our sample under the prescribed load. In addition to the stress, we also record the number of unbroken bonds and the total elastic potential energy of bonds in the system on every NVT update.

Another key task for our result manager object is taking snapshots of the intermediate results for diagnostic purposes. We achieve this by recording the position of particle nodes and the state of the bonds every 10 NVT updates. Although the number of NVT steps between each snapshot is arbitrary, the idea is to take enough snapshots for fracture propagation and future analysis without the result file getting too large. Finally, we set the success flag of the result manager and save it to a `.mat` file at the end of the simulation. We also wrap the entire SGCMC function in a `try` - `catch` block so that we can save any results we have, as well as the error message and the stack trace, for diagnostic purposes in case our code encounters an unexpected error.

2.3 Performance Optimization

With the steps above, we have successfully implemented a working SGCMC simulation. However, it is essential that we improve its performance to be fast and use less memory, especially since the SGCMC simulation performs multiple NVT updates, making it slower than a standard finite element analysis. We take the following steps to optimize our implementation.

2.3.1 Performance Profiling

The first step to optimizing our SGCMC implementation is identifying the bottleneck of our algorithm. We perform the following algorithm analysis:

- Let n be the number of particle nodes in our lattice.
- The space complexity of the MC steps is the space complexity of our internal representation, which is $O(n)$.
- The space complexity of the NVT steps is, however, governed by the stiffness matrix, which is of dimension $2n \times 2n$. Thus, the space complexity of the NVT steps is $O(n^2)$.
- The time complexity of a single MC trial is $O(1)$. Therefore, the time complexity of MC steps is $N \times O(1)$. Since the number of bonds N scales linearly with the number of particle nodes under the assumption of local connectivity, the time complexity of MC steps is $O(n)$.
- The time complexity of the NVT steps is, however, governed by the equation solving of Equation 2.3 or Equation 2.4, which depends on `mldivide` or `quadprog` function. While the time complexities of these functions are not explicitly stated as they depend on the exact solving algorithm used (which depends on the input, as listed in MATLAB documentation [21]), the general time complexity of these matrix operations relies on the matrix multiplication, whose time complexity is $O(n^{2.38})$ [23].

Thus, we identify that NVT steps are the main bottleneck of our algorithm. We also confirm this diagnosis with MATLAB's `run and time` tool, whose result will be shown in Chapter 3.

2.3.2 Exploiting the Sparsity of Stiffness Matrices in NVT Updates

The key step to optimizing NVT updates is exploiting the sparsity of the stiffness matrix. Although the stiffness matrix is of dimension $2n \times 2n$ (the number of degrees of freedom) in a two-dimensional case, most of its entries are zero. For a bond connecting between the i-th and the j-th particle nodes, its stiffness matrix is given by:

$$K_{2pt}(i,j) = k_b \begin{pmatrix} \cos^2\theta & \sin\theta\cos\theta & -\cos^2\theta & -\sin\theta\cos\theta \\ \sin\theta\cos\theta & \sin^2\theta & -\sin\theta\cos\theta & -\sin^2\theta \\ -\cos^2\theta & -\sin\theta\cos\theta & \cos^2\theta & \sin\theta\cos\theta \\ -\sin\theta\cos\theta & -\sin^2\theta & \sin\theta\cos\theta & \sin^2\theta \end{pmatrix} \tag{2.5}$$

where k_b and θ are the bond stiffness and angle, respectively. If we consider the local stiffness matrix as a 2×2 block matrix and the global stiffness matrix as an $n \times n$ block matrix of 2×2 blocks, we can see that there are a total of n non-zero diagonal terms and $2N$ non-zero

off-diagonal terms. Considering one shell of interaction in two dimensions, there are $N < 8n$ bonds. Thus, there are at most $68n$ non-zero entries in the $2n \times 2n$ stiffness matrix. By using the sparse matrix representation, which stores the (i, j, v) indices and values of non-zero entries, we can reduce the space complexity of the NVT updates to $O(n)$, similar to other parts of the code.

On top of the reduced memory usage, utilizing sparse matrix representation for the stiffness matrix also significantly reduces the computation time of NVT updates. The functions `mldivide` and `quadprog` have different implementations for dense and sparse input and can execute much faster when the input is presented as a sparse matrix than it can when the input is presented as a dense matrix, even though the content remains the same, under the assumption that the matrix is indeed sparse.

While it is possible to convert our stiffness matrix to a sparse matrix from our original dense matrix implementation to achieve the aforementioned speed-up, doing so has a significant overhead and does not give us the benefit of linear space complexity. Therefore, it is important that we also modify our stiffness matrix assembly to directly create a sparse stiffness matrix with `K = sparse(i,j,v,m,n)` sparse matrix constructor. It is easy to create the list of (i, j, v) for the off-diagonal terms as each term is a result of exactly one bond. However, each term along the (block) diagonal of the global stiffness matrix is a sum of the corresponding terms from the local stiffness matrices of the bonds that are connected to the particle node. Therefore, we need to process them accordingly. An example implementation of the sparse stiffness matrix assembly can be found in the appendix.

2.3.3 Other Optimizations in NVT Updates

We can further improve the performance of NVT updates by using the back-slash operator (which calls `mldivide`) over `quadprog` when applicable, as `mldivide` is substantially faster than `quadprog`. To do so, we need to verify that the stiffness matrix is well-conditioned. While the calculation of matrix condition number through `cond` or the reciprocal of `rcond` can be computationally expensive, we can calculate its estimate using `condest`, which is much faster. Since we are already working with a sparse stiffness matrix, we naturally satisfy the pre-conditions of `condest` that require a sparse input, Although `condest` is relatively computationally expensive compared to `mldivide`, it is cheaper than `quadprog`. According to our profiling, which will be shown in Chapter 3, it is worth checking if the matrix is well-conditioned and use `mldivide` over `quadprog` when applicable as long as the stiffness matrix is well-conditioned at least 20% of the time.

Another thing to consider is that `quadprog` is much faster when the upper and lower bounds are not specified (i.e., no $\xi_{min} \leq \xi \leq \xi_{max}$). According to our profiling, it takes approximately 5 to 7 times longer to solve the minimization problem with `quadprog` when the bounds are specified. Since the bounds are added to catch a rare case that results in invalid results, we can further improve the performance of our NVT update by running the minimization without bounds, then verifying the result against the bounds and re-running it with bounds if necessary.

2.3.4 Optimizations of MC Trials

Although the bottleneck of SGCMC is in the NVT update, we also consider some optimizations to the MC trials as well.

Acceptance Criterion VS Rejection Criterion

One optimization we make to the MC trials is switching from the rejection criterion to the acceptance criterion. Although Ulm (2022) formulated the acceptance criterion of $P(o \to n)$, the proposed pseudo-code and our implementation in Section 2.2 change the state of the bond and then restore it when the acceptance criterion is not met (i.e., rejection criterion) [9]. However, our profiling shows that an MC trial is accepted only approximately 1% of the time. Thus, we can reduce the number of operations and memory access by making the change after the MC trial is accepted instead of changing and then reverting said change. Even though this optimization is very minor and has negligible impact, there is no reason not to.

Calculating Elastic Bond Energy in NVT Updates

As mentioned in Section 2.2, we calculate the elastic bond energy $u_\lambda(i)$ of every bond during the NVT updates and use this information during MC steps without re-calculating them. While the original motivation was for future-proofing our design, it also has minor speed benefits. By calculating the bond energy of every bond at once, we can speed up the calculation by vectorization, which is automatically handled by MATLAB when we run calculations on an entire vector or matrix. Even though we may not use every calculated bond energy, vectorized operations are almost 10 times as fast as serial operations with `for` loop. Thus, it is a speed up if we can show that at least 10% of the calculated bond energy will be used in MC trials before the next NVT update.

We can analytically show that the expected value of the number of unique bonds tested by N MC trials is $N(1-(1-\frac{1}{N})^N)$. Let X and X_i be the number of unique bonds picked and the random variable that takes the value of 1 when the i-th bond is picked at least once, respectively. We can show that $E[X_i] = 1 - P(\text{bond is never picked}) = 1-(1-\frac{1}{N})^N$. Then, using the linearity of expectation, we can show that the expectation of the number of unique bonds tested is

$$E[X] = E[\sum_{i=1}^{N} X_i] = \sum_{i=1}^{N} E[X_i] = NE[X_i] = N(1-(1-\frac{1}{N})^N) \tag{2.6}$$

We can then show that the percentage of the bonds picked by the MC trials is $\frac{1}{N}E[X]$ which converges to

$$\lim_{N\to\infty} \frac{1}{N}E[X] = 1 - \frac{1}{e} \approx 0.632 > 0.1 \tag{2.7}$$

Therefore, vectorized calculation of the bond energy of every bond during NVT update is, in fact, a speed-up.

Batch MC Trials

Another idea to speed up our MC trials is to perform multiple MC simulations simultaneously to leverage the vectorized calculations, similar to our previous idea. Since the particles are frozen during MC trials, MC trials done on different bonds are completely independent from each other. However, MC trials done on the same bond are still dependent, particularly when a trial is accepted and the bond state is flipped. Therefore, we have to properly account for this, incurring a substantial overhead. Coupled with the fact that MC steps are not the bottleneck of SGCMC, this is, once again, a minor improvement.

2.3.5 Parallelized SGCMC Simulations

While it is possible to use multiple threads on a single simulation, the speed-up gained is sub-linear with the amount of computation resources occupied due to the overheads from parallelization, especially since `quadprog`, which is the bottleneck, does not support MATLAB `ThreadPool`. We can, however, run multiple simulations simultaneously using super-computing infrastructure. Since each simulation can occupy its own thread without having to communicate with each other, we can achieve a linear speed-up with this high-throughput computing workflow. Since we designed our SGCMC function to read in the image of the sample's porous geometry, which is the only input parameter that differs from sample to sample, from a file and write its results to a result file, we only need to write a job processor that calls our SGCMC function. We implement our parallelized simulations on MIT Lincoln Laboratory's MIT SuperCloud, which allows us to simulate up to 384 samples concurrently [24]. However, there is one caveat with running MATLAB code on super-computing infrastructure. By default, MATLAB initializes the random number generator at the start of a session with seed 0 [21]. Therefore, if we are to run multiple simulations on the same sample on different threads, the results from each thread will be the exact same, defeating the purpose of our stochastic method. This could also interfere with our sample generation. To avoid such unintended results, it is critical that we set the seed of the random number generator with `rng(seed)` such that each thread has a different random number generator seed. One of the simplest ways to achieve this is to use the thread number as the seed.

2.4 Conclusion

We have, in this chapter, addressed our first research question with the SGCMC simulation that we formulated and implemented. We then optimized the performance of our SGCMC simulation by exploiting the sparsity of the stiffness matrix, as well as several other changes, so as to reduce the computational cost of our SGCMC algorithm. Finally, we made our SGCMC algorithm scalable by modifying it for launching multiple parallel simulations on super-computing infrastructures such as MIT SuperCloud.

Chapter 3

Simulation Results

In this chapter, we discuss the results of our SGCMC implementation from Chapter 2. First, we show some simulation results that demonstrate the stochastic nature of our algorithm in Section 3.1. We then discuss the post-processing necessary for accurate interpretation of our results in Section 3.2. Finally, we compare the computational performance of our optimized algorithm with the initial implementation in Section 3.3.

3.1 Sample Simulation Results

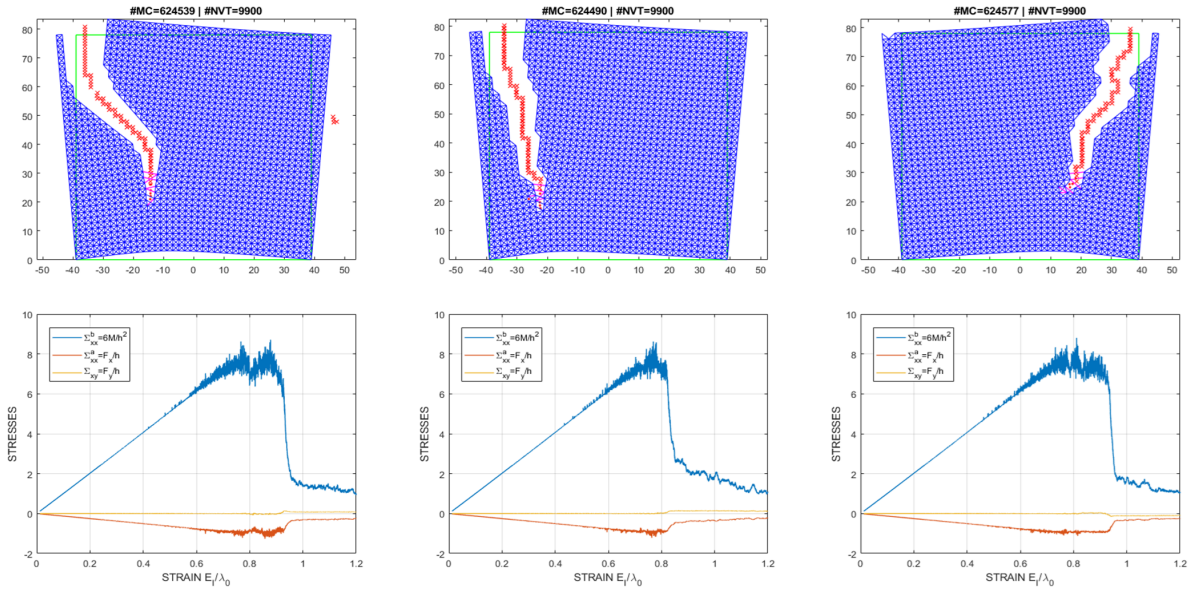

Figure 3.1: Results of 3 separate SGCMC simulations on the same simple dense lattice under the same bending load. The top sub-figures show the fracture propagation in the lattice while the bottom sub-figures show the associated stress-strain curve.

We show in Fig 3.1 some example results of the SGCMC simulation on a simple dense lattice under bending load. Although we run the simulation on the exact same lattice under the exact same load, the fracture propagation and the stress-strain curve of the 3

separate simulations are, though similar, different. Our SGCMC simulation demonstrates its stochastic nature as intended.

We can also see that the stress-strain curves behave according to the theory. At first, the stress increases proportionally to the strain according to Hooke's law. Then, the curve deviates from the initial linearity as the system re-distributes the stress due to the distributed fracture creation. Note the fluctuation of stress in this region. Finally, a phase transition occurs as the fracture propagates rapidly while the stress drops drastically and remains there for the rest of the simulation.

We also show in Fig 3.2 the distribution of stress-strain curves from multiple SGCMC simulations for the same problem, this time on a porous lattice under tensile load instead. We note the qualitative change of the stress-strain curve. As a result of the porous lattice, the phase transition is now less abrupt as the fracture propagates multiple times through each hole along the path.

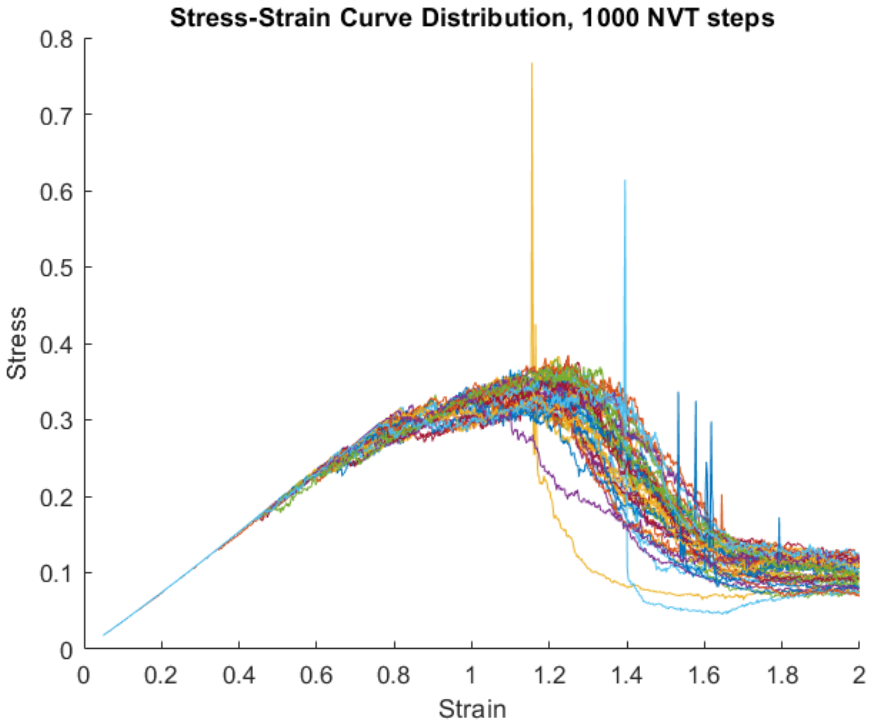

Figure 3.2: The distribution of the stress-strain curve of 48 separate SGCMC simulations on the same porous lattice under the same tensile load.

3.2 Post-Processing

Another take-away from Fig 3.2 is the existence of anomalous spikes. We believe these spikes to be artifacts of the stochastic method. Although they correct themselves in the next NVT step, they can interfere with our interpretation of the stress-strain curves. For instance, if

we extract the ultimate strength of the sample by directly taking the maximum stress, we will end up with outliers.

Therefore, it is essential that we post-process our stress-strain curves to address these anomalous spikes. The exact method may vary, depending on the quantity of interest. In our case, we are interested in the ultimate strength and propose the following options, as demonstrated in Fig 3.3:

- Take the n-th highest stress as the ultimate strength instead of the highest stress
- Mean filter
- Low-pass filter
- Minimum filter

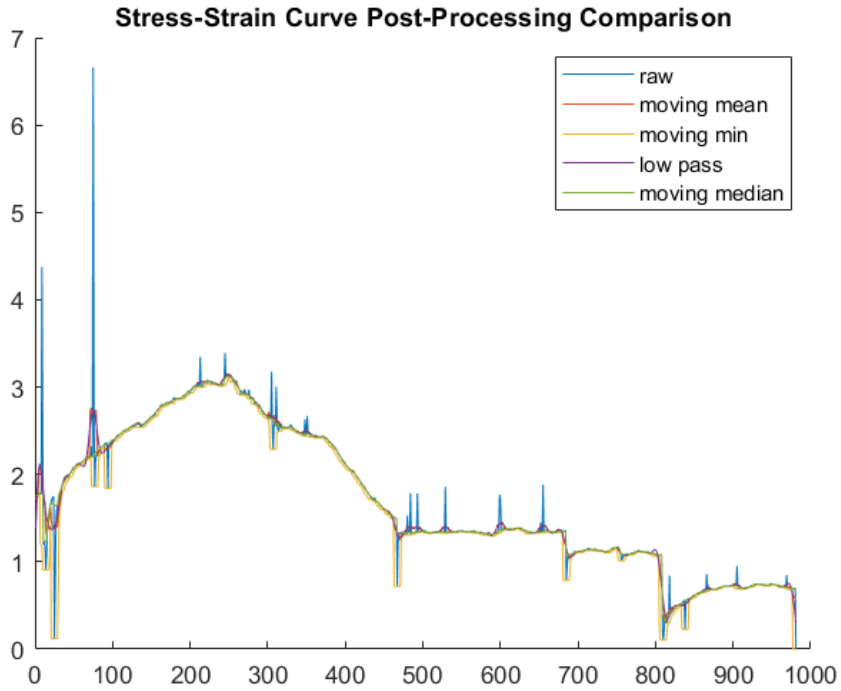

Figure 3.3: Comparison of post-processing methods on a sample stress-strain curve.

While taking the n-th highest stress works for our case, it only works for ultimate strength extraction and is not robust against multiple spikes. Meanwhile, even though mean filtering removes the spikes, they are averaged to nearby entries. Therefore, we eliminate taking the n-th highest stress and mean filtering from our options.

Low-pass filter is a good way to eliminate high frequency noises, including these random spikes, and gives us better controls than mean filter, which is a specific type of low-pass filter. However, low-pass filter can fail to eliminate spikes that occur at the very beginning

of the stress-strain curve. On the other hand, the minimum filter is extremely robust for the purpose of extracting the ultimate strength from the stress-strain curve, even though it only works for ultimate strength extraction like taking the n-th highest stress. In the end, we decide to use both the low-pass filter and the minimum filter to cross-check each other.

We also need to make sure that our prescribed strain is sufficient for the sample to reach the ultimate strength and, subsequently, the phase transition. While it is an easy task for a human to identify if the curve goes beyond its ultimate strength, it can be challenging to implement such logic algorithmically. We can, however, set up a simple test with good specificity at the cost of sensitivity. We accept a simulation as valid if the average stress of the last 10% of entries is below 70% of the maximum (post-processed) stress. While the exact number can vary, we confirm that the result of our validation agrees with our expectation and accept this as our validation method.

3.3 Performance Comparison

We compare the performance of our SGCMC code before and after our optimizations in Section 2.3 by running a quick simulation of both versions on a sample 100 x 100 dense regular lattice for 100 NVT updates and timing them with MATLAB's `run and time` tool. As shown (in a very simplified version) in Table 3.1, the optimized code is approximately 16 times faster than the original implementation. Although the actual execution time can vary greatly due to a range of reasons, including the percentage of well-conditioned stiffness matrix as well as the load on my personal computer during the simulation from other processes, we can confirm that our performance optimizations worked as intended.

Table 3.1: The results of run-and-time performance profiling of SGCMC simulation on a 100 x 100 dense regular lattice, before and after performance optimization, 100 NVT updates.

Function	Total Time (s) - Before	Total Time (s) - After
SGCMC	11355	701
NVT	10782	368

Another observation is that the NVT updates, although still a significant part of the SGCMC simulation time-wise, take up less percentage of the algorithm run time. As mentioned in section 2.3, we do not have theoretical time complexity of the optimized NVT updates. We can, however, profile the empirical time complexity of our NVT solving algorithm, as shown in Fig 3.4, for our use case. Since NVT updates are still the bottleneck of our code, the overall time complexity of our algorithm is $O(n^{1.24})$. Since the sparsity of our stiffness matrix also increases with n, the empirical time complexity is now super-linear instead of sub-cubic.

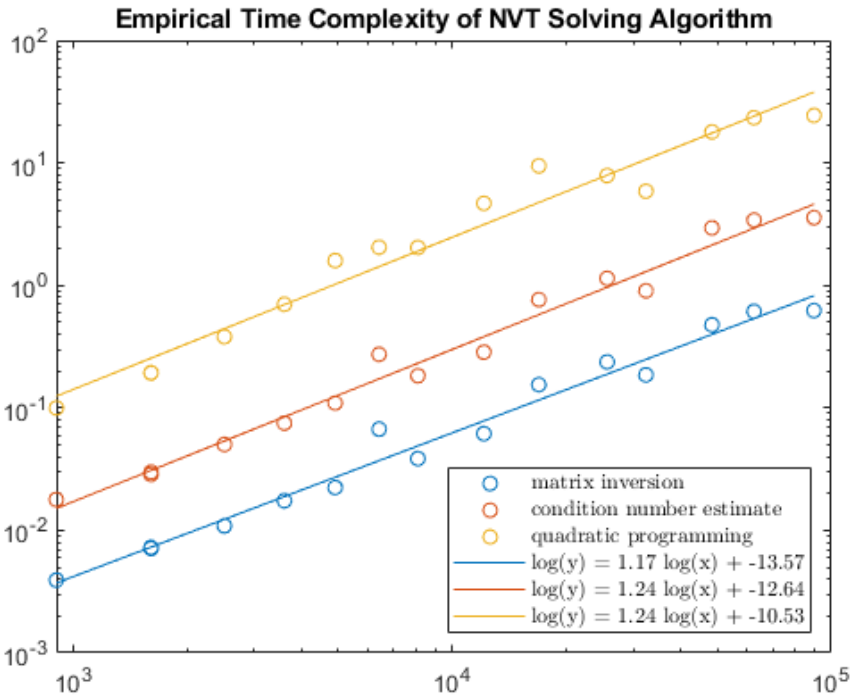

Figure 3.4: The empirical time complexity of mldivide, condest, and quadprog on sample stiffness matrices with sparse representation. The x-axis shows the number of particle nodes in the lattice. The y-axis shows the time taken to execute the function.

3.4 Conclusion

We have, in this chapter, presented results of our sample SGCMC simulations which demonstrate the stochastic nature as expected and desired. We have also proposed a way to post-process our resulting stress-strain curve for accurate interpretation by utilizing both a low-pass filter and a minimum filter. Finally, we quantify the effect of our performance optimizations and find that we have reduced the time complexity of our algorithm from $O(n^{2.38})$ to $O(n^{1.24})$.

Chapter 4

Data Set Generation

In this chapter, we begin addressing the second part of this thesis where we approach the structure function relation of two-dimensional porous material with statistical models to achieve a more cost-effective structure function relation. However, in order to train statistical models, we need to first create a data set by generating samples, running SGCMC simulations on those samples, post-processing the results, and extracting the ultimate strength as our target y-variable. Since we have already demonstrated how to simulate and post-process the results in Chapter 2 and 3, we will be focusing on the sample generation in Section 4.1. We then tune the important hyperparameters of our data set creation process in Section 4.2 - 4.4.

4.1 Sample Generation

Ideally, we would like to use actual images of concrete under a microscope. However, since we could not obtain such data set, we generate our own samples based on the known distribution of pore radius. Since we are working in reduced units [9], which is perfect for our use-case focused on the porous geometry of the sample, we can set the radius of our particle nodes to 1 unit-length, which corresponds to the representative grain size of C-S-H gel of approximately 10 nanometers [25]. Although the exact distribution of pore radius varies from source to source, we can conclude that it is a log-normal distribution with mean pore radius around 50 to 100 nanometers [26]–[28]. We also decide to generate the input images such that each particle node corresponds to a 10 x 10 pixel chunk (i.e., the entire image will be 1010 x 1010 pixels if we want our lattice to be 101 x 101 particles). While it is obvious that our input images should be at least as discretized as our lattice, the choice of 10 x 10 pixels is a little arbitrary but sensible.

Therefore, we can generate each sample using the following method:

1. Randomize the underlying parameters of the sample, namely the prescribed porosity, the mean pore radius, and standard deviation of the pore radii.

2. Draw from the log-normal distribution the radius of a pore and randomly place the pore on the sample by randomizing the x and y coordinates of the center from the uniform distribution.

3. Calculate the porosity of the sample and repeat step 2 until the target porosity is met.

We sweep the underlying parameters, with uniform distribution, over the following ranges: the porosity from 3% to 30%, the mean pore radius from 1 to 10 unit-lengths, and the standard deviation of the pore radii from 10% to 50% of the mean pore radius. We show in Fig 4.1 some of our generated input images that are fed to our SGCMC simulation.

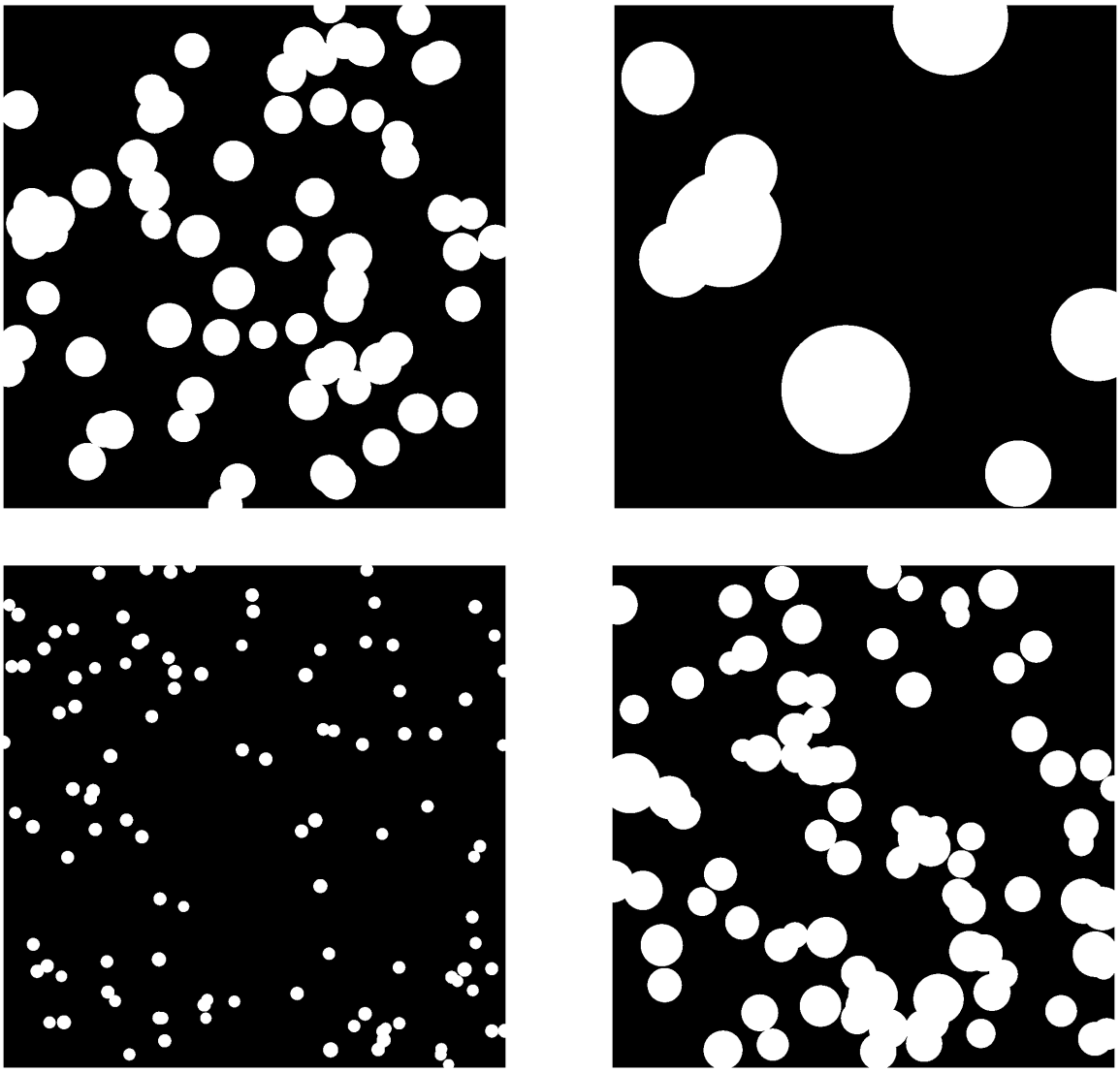

Figure 4.1: Some samples of generated input images.

4.2 Determination of the Number of NVT Steps

Under the same algorithm and computational resources, there are 3 main hyperparameters that determine how long it takes to run our simulations, namely the number of NVT steps

per simulation, the number of particles in a sample, and the number of samples. The number of NVT steps governs the accuracy of the SGCMC simulation. The higher the number of NVT steps, the closer the simulation result is to the ground truth. On the other hand, the size of a sample, which is directly proportional to the number of particles in the sample, needs to be sufficiently large in order for our simulation results to be representative of the most dominant heterogeneity of the material. If the sample size is too small, our results will be affected by the edge effects and the pores will be the macro structure instead of the texture of the material. Finally, if the number of samples is too small, we will not have enough data to train our statistical models. While the simpler models such as linear regression can be trained on a small data set, complex models such as neural networks have higher training complexity and require more data for training [17]. Therefore, it is important that we balance these hyperparameters so that we have sufficient NVT steps for the accuracy, sample size for the representativity of the samples, and samples for model training, while keeping the computational cost under control.

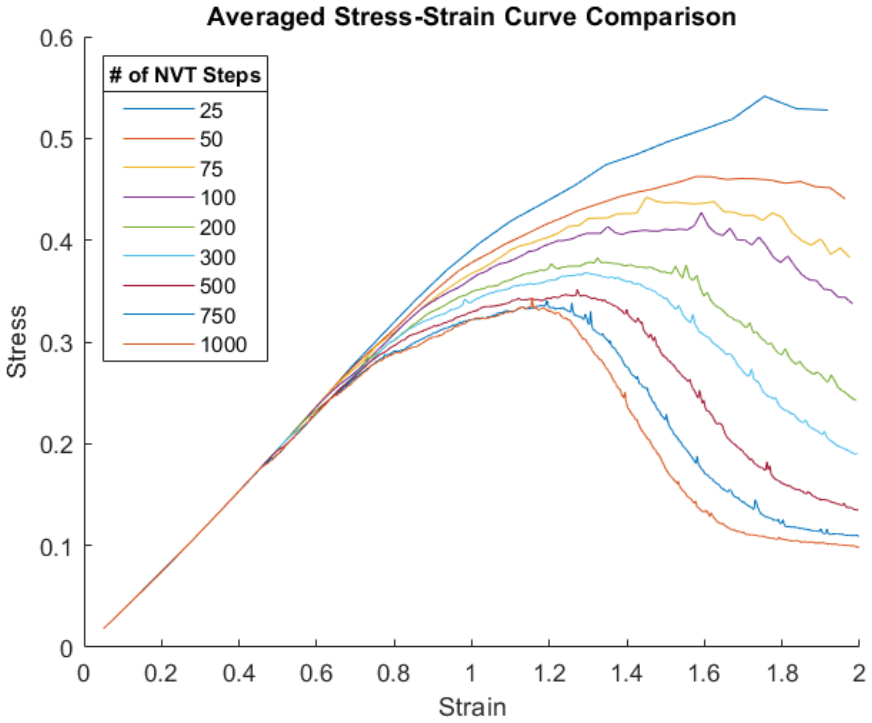

Figure 4.2: Comparison of SGCMC simulation results with different numbers of NVT steps.

To determine the appropriate number of NVT steps, we run simulations on the same problem multiple times with varying numbers of NVT steps such that there are multiple simulations for every number of NVT steps. We can then, for each number of NVT steps, aggregate the stress-strain curves by averaging them together to obtain the results shown in Fig 4.2 above. As demonstrated in the figure, the smaller the number of NVT steps, the less accurate the results are, and we can immediately tell that the simulations with

less than 100 NVT steps are inaccurate and qualitatively incorrect. We propose that, to quantitatively determine if we can reduce the number of NVT steps, we accept the reduction if the difference between the two averaged curves is smaller than the standard deviation. The rationale of the proposal is that if the difference between the mean curves is smaller than the standard deviation, then we cannot significantly distinguish the two results from each other. Therefore, we conclude that we need 750 NVT steps per simulation going forward in order to generate an accurate data set.

4.3 Determination of the Sample Size

As for the sample size, we conduct a similar profiling. However, since we have to change the size of the sample, we cannot run the simulation on the same sample anymore. Instead, we fix the underlying parameters of the sample generation and generate multiple samples for each sample size, ranging from 101 x 101 to 1001 x 1001 particles. However, since the larger samples take significantly longer to complete the simulation than the smaller samples, we did not let them run to completion, especially since we noticed that there is no significant change in the result when we change the sample size from 101 x 101 to 201 x 201 and subsequently 301 x 301, as shown in Table 4.1. As such, we decide to proceed with 101 x 101 sample size going forward.

Table 4.1: The effect of sample size on the ultimate strength.

Lattice Size	Count	Mean Strength	Strength S.D.
101 x 101	875	0.394	0.126
201 x 201	203	0.403	0.158
301 x 301	32	0.425	0.169

4.4 Determination of the Number of Samples

It is harder to estimate the required number of samples. Although we did not finish training neural networks due to the time constraints, at the time of data set generation, we intended to train Convolutional Neural Networks (CNNs) as our most complex models that require the most data. While the general expectation is that the number of data points should be at least 10 times the number of degrees of freedom in a model, this does not hold well in very complex models as there may be overlaps between data points. In addition, it is possible to reduce the number of required data points by training for domain adaptation from pre-trained models, such as pre-trained ResNet18 weights `torchvision.models.ResNet18_Weights` from PyTorch [29].

We take a look at some existing data sets with image input to find some points of reference. ImageNet has 14,197,122 entries while Architectural Heritage Elements (AHE) has only 10,235 images. However, it is impossible for us to generate millions of data points

with our resources under our time constraints. As a result, we try to generate (and simulate) as many samples as we can within our time limit and end up with 20,000 samples. While this should be more than enough for simpler models, such as linear regression, we are unsure if our data set is large enough for more complex models such as Gradient Boosting Decision Trees (GBDT) and Neural Networks. We can, however, later confirm if we have trained our model with sufficient training data by plotting the accuracy against the number of training samples.

4.5 Conclusion

We have, in this chapter, discussed our data set creation with a focus on the sample generation process. Since we do not have access to actual images of concrete under a microscope, we generate our own image inputs based on the log-normal pore size distribution and sweep our sample generation parameters over a reasonable range. We also discussed the significance and demonstrated the tuning process of the three hyperparameters that govern the computational cost and the quality of our data set.

Chapter 5

Analysis and Statistical Model Creation

In this chapter, we analyze our data set from Chapter 4 to create a cost-effective structure function relation of two-dimensional porous material using various statistical models. We define in Section 5.1 our metrics for model evaluation. Then, we discuss our hypothesized features in Section 5.2. We conduct our exploratory data analysis in Section 5.3. Finally, we present and compare the performance of our models with respect to the baseline model, which utilizes only porosity, in Section 5.4.

5.1 Model Evaluation

Since our target variable, the ultimate strength of the sample, is a continuous variable, the structure function relation creation is a classic regression problem. To obtain an unbiased estimator, we use the mean square error as the loss function to minimize. For the ease of interpretation, we use the coefficient of determination (R2), which is negatively proportional to the mean square error, as shown in the equation below:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 \tag{5.1}$$

$$R^2(y, \hat{y}) = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \bar{y}_i)^2} = 1 - \frac{\text{MSE}(y, \hat{y})}{Var(y)} \tag{5.2}$$

We can also interpret the coefficient of determination as the proportion of the variance of the target variable that is explained by our statistical model. However, it is important to note that a model can be trained to perfectly fit its training data but fail to generalize to new, unseen, data. Such a model is not useful as it cannot give accurate predictions in real inference scenarios. This behavior is called over-fitting. In order to select useful models that perform well on both seen and unseen data, we split our data set into a training set and a test set. The training set has 80% of the data and is used for model training while the test set has the remaining 20% of the data and is used for model evaluation.

5.2 Feature Creation

We define the input of our structure function relation as the porous geometry of a sample, which is represented by an image. However, most statistical models are not designed to handle image input. While it is possible to turn each pixel into a feature, doing so would result in significant over-parameterization, which may lead to over-fitting. Therefore, it is critical that we hypothesize and extract features that parameterize the porous geometry of a sample (i.e., texture parameters).

The first feature we create is the porosity (ϕ), which can be viewed as the first statistical moment of the pores, as it is a very well-established feature that has been shown to have significant effect on the strength of porous materials in the literature [6], [12]. The intuition is simple; the greater the porosity, the less material there is to give strength to the sample. However, in order to better parameterize the porous geometry of a two-dimensional sample beyond porosity, answering our second research question, we create additional features using the ideas from the following sub-sections.

5.2.1 Two-point Correlation Function

The two-point correlation function, also known as radially averaged correlation function, is a very popular statistical tool in astronomy and cosmology [30]. Torquato (1998) and, subsequently, Jiao (2013) demonstrated its application in modeling of random heterogeneous materials [31], [32]. Thus, it is a perfect feature for parameterization of porous geometry. The two-point correlation function (S_2) is defined, under the assumption of isotropicity, as:

$$S_2(r) = S_2(x_1, x_2) = \langle I(x_1), I(x_2) \rangle \tag{5.3}$$

where $r = \|x_1 - x_2\|$ and x are the distance between two points and the position vector, respectively.

We use function `autocorr2` by Benito (2021) to calculate the two-point correlation function up to the distance of 303 pixels as it is 30% of the maximum side length as recommended by the package author [33]. With the two-point correlation function, we can parameterize the input image and reduce the dimension of inputs, from 1,020,100 pixels to 303 entries of S_2 from $S_2(1)$ to $S_2(303)$. We can also further reduce the dimension of the feature by fitting it with an exponential function. We know, from theory, that $S_2(0) = \phi$ and $\lim_{r\to\infty} S_2(r) = \phi^2$. Therefore, we can fit the calculated two-point correlation in the form of

$$S_2(r) = (a - a^2) \exp\left(-\frac{r}{b}\right) + a^2 \tag{5.4}$$

and define the chord length

$$\texttt{cl} = -\frac{S_2(0)}{S_2'(0)} = \frac{a}{-\frac{1}{b}(a - a^2)} = \frac{b}{1 - a} \tag{5.5}$$

as a new parameter that summarizes the two-point correlation function as a single scalar variable.

In addition to the two-point correlation function and the chord length, we also believe that there is predictive value in the residual between the actual two-point correlation and the fitted curve, as shown in Fig 5.1. We can use this residual either as a vector of the same length as the two-point correlation or as two scalar variables: the location of the first extremum and its associated value.

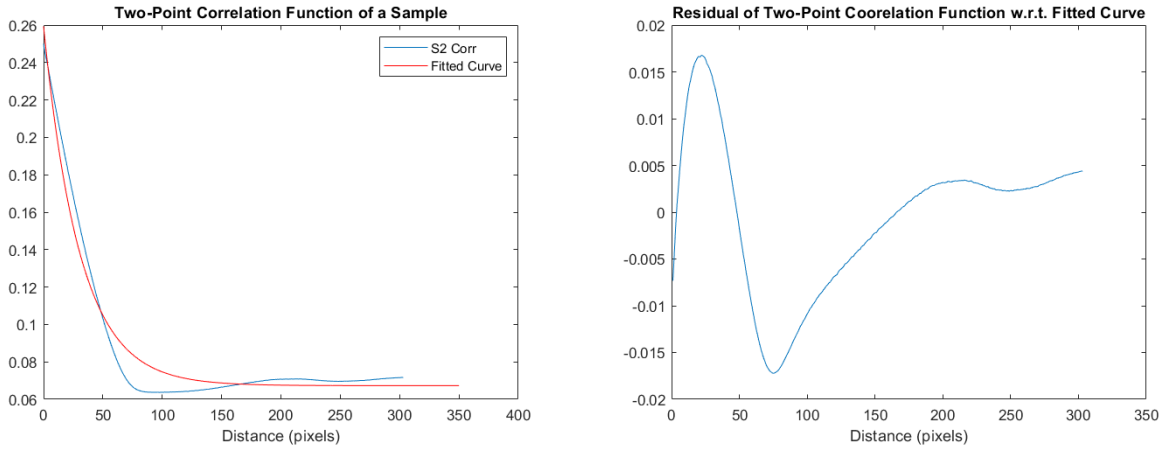

Figure 5.1: The two-point correlation, the fitted curve, and the residual of a sample.

5.2.2 Multi-scale Disorder Index

Since the porosity, which is the first statistical moment of the pores, is known to be an extremely important feature, we hypothesize that the variance of the pores should be an important feature as well. Laubie (2017) has demonstrated that this is true by defining the disorder index I_d as

$$I_d = (\langle \phi_i^2 \rangle - \phi^2)^{\frac{1}{2}} \tag{5.6}$$

where ϕ_i represents the local porosity and showing that, conditional on the porosity, the tensile strength decreases as the disorder index increases [10].

However, in contrast to Laubie's work, our pore radius is not a constant. Therefore, we refine the definition of the disorder index as a function of distance, similar to the two-point correlation, as

$$I_d(r) = (\langle \phi_i^2(r) \rangle - \phi^2)^{\frac{1}{2}} \tag{5.7}$$

where $\phi_i(r)$ represents the local porosity at the scale of r x r pixels.

Similar to the two-point correlation function, we can also fit the calculated multi-scale disorder index with a function. We know, from theory, that $I_d(1) = (\phi(1-\phi))^{\frac{1}{2}}$ and $\lim_{r\to\infty} I_d(r) = 0$. Therefore, we can fit the calculated multi-scale disorder index in the form of

$$I_d(r) = a \exp\left(-\frac{r}{b}\right) \tag{5.8}$$

and use b as the new scalar parameter that summarizes the multi-scale disorder index. Similar to the two-point correlation function, we also define the residual between the actual

multi-scale disorder index and the fitted curve, as well as the location and the associated value of the first extremum of the residual, as our features as well.

5.2.3 Pore Radius Distribution

We believe that the distribution of the pore radius is also an interesting feature. However, although we know the radius of every pore we placed on the sample in our sample generation process, it is an unobservable feature i.e., we do not have access to such information from our defined image input. Therefore, we have to recover this information using various hole detection algorithms. We decide use the Circle Hough Transform (CHT) algorithm as it is a well-known and robust method [34], [35], which is already implemented in MATLAB's image processing toolbox as `imfindcircle` [36]. From the distribution of the pore radius, we use the average pore radius and the 0-th, 5-th, 10-th, ..., 95-th, and 100-th percentile of the pore radius as our features. The choice to sample the pore radius every 5% is arbitrary. In addition to using the pore radius sampled above as our features, we also use those radius to sample our two-point correlation function and multi-scale disorder index, as well as their fit residual, to try and reduce their dimension.

5.2.4 Area Moment of Pores

The last set of features we created is the area moments of the pores. Inspired by the moment of inertia, which is the second moment of area, we define the following features based on this concept:

- $|\sum_i x_i I(x_i, y_i)|$ and $|\sum_i y_i I(x_i, y_i)|$, the absolute value of the first moment of area
- $\sum_i |x_i| I(x_i, y_i)$ and $\sum_i |y_i| I(x_i, y_i)$
- $\sum_i x_i^2 I(x_i, y_i)$ and $\sum_i y_i^2 I(x_i, y_i)$, the second moment of area

where the x-axis is parallel to the applied tensile load and passes through the center of the sample while the y-axis is perpendicular to the applied tensile load and passes through the center of the sample. The first feature helps us measure the asymmetry of the sample while the second and the third features help us measure the distribution of the pores if they are clustered towards the middle or the edge of the sample. Under the assumption of isotropicity, these features should be constant, or very clustered around a constant, for samples that are sufficiently large. In another word, these features are not texture parameters of porous geometry. However, they should have some predictive values and are worth exploring.

5.3 Data Exploration

With our features defined in Section 5.2, we process the simulation results into a Pandas dataframe [37] and conduct our exploratory data analysis to get a better understanding of our data. First, we plot the distribution of our scalar independent variables (x-variables) and target y-variable to identify if any variable needs additional transformation or if there are

outliers that must be processed, either by removal, winsorization, or other methods. Then, we plot the joint distribution of our x-variables and y-variable to examine their relationship. Example plots from our exploratory data analysis are shown in Fig 5.2.

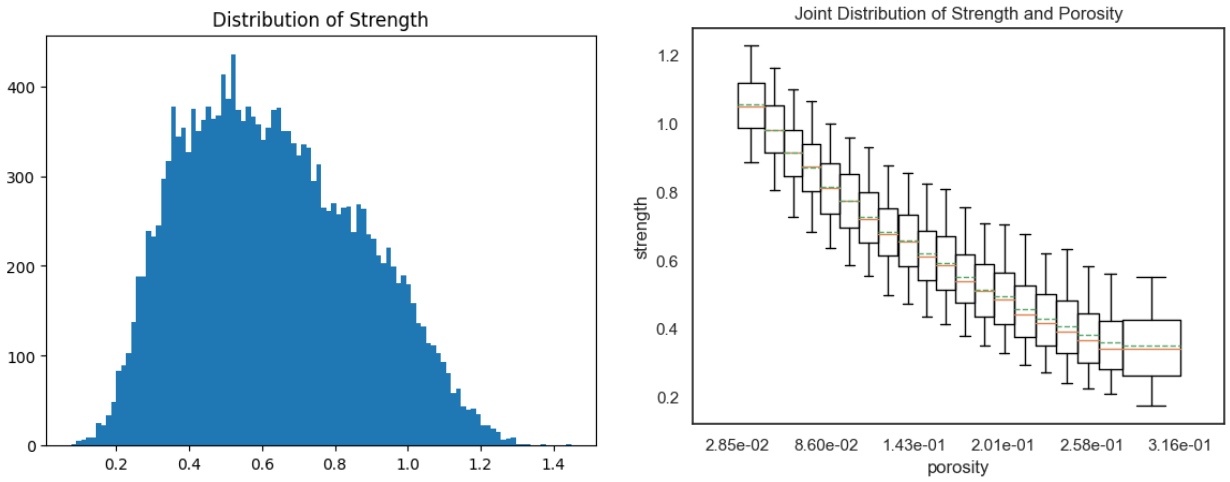

Figure 5.2: Sample variable distribution and joint distribution plots from exploratory data analysis.

The result of our exploratory data analysis reveals that our features do not need to be processed any further and that most of our features demonstrate some form of relationship with the target variable. However, there is one exception: the multi-scale disorder index curve fit parameter b. As shown in Fig 5.3, the values of b are exactly 100 for most of the entries. Therefore, this parameter b is not a good feature for our strength prediction and is thus removed.

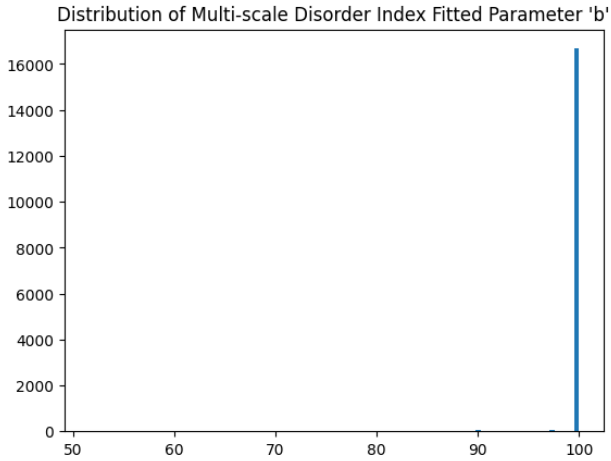

Figure 5.3: Distribution of the multi-scale disorder index curve fit parameter 'b'.

Finally, we plot the correlation matrix of our scalar features and target variable, as shown in Fig 5.4. Since some of our features are highly correlated, we must proceed with caution, especially when building linear regression models.

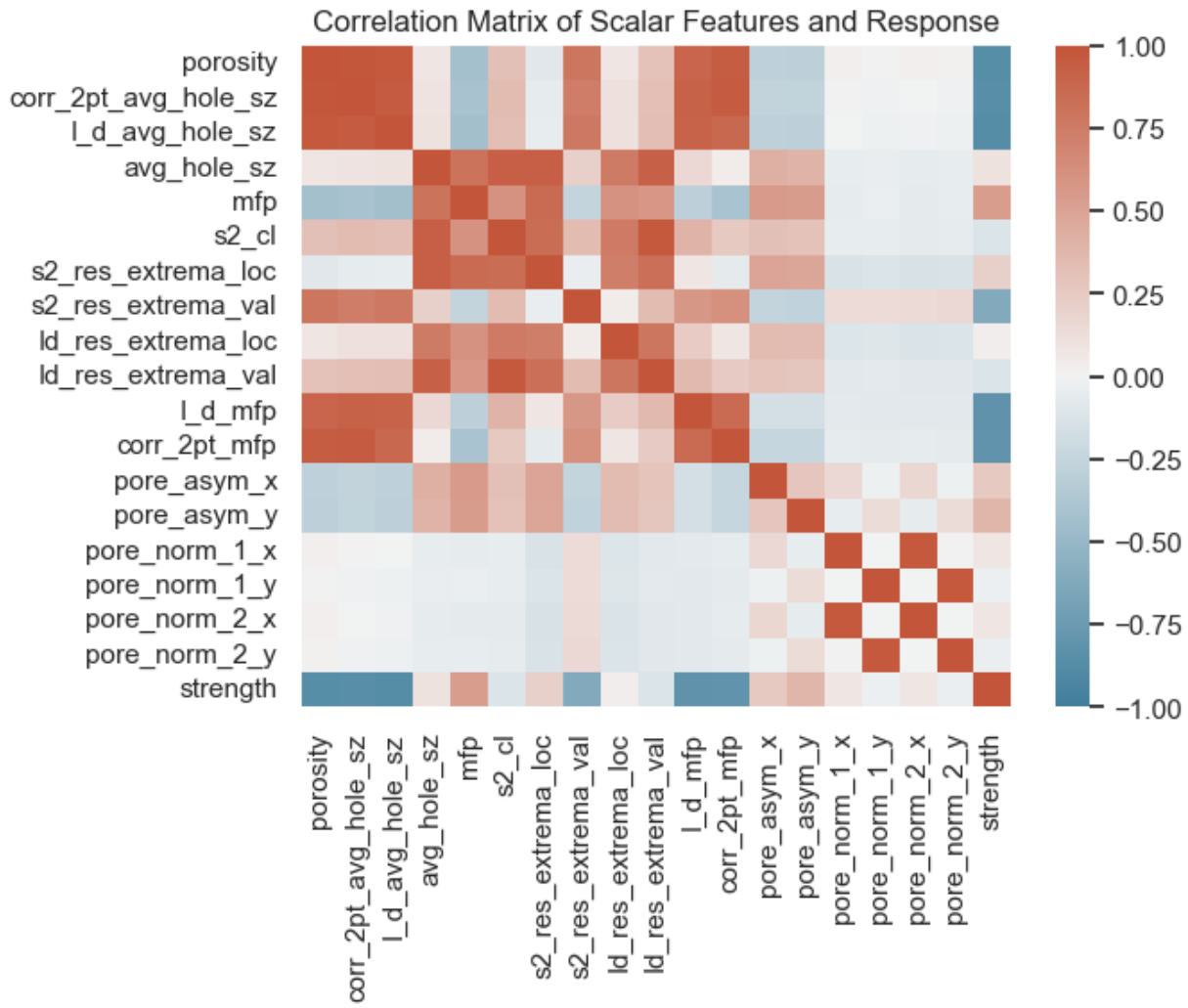

Figure 5.4: Correlation matrix of scalar features and strength.

5.4 Statistical Modeling

We experiment with the following statistical models:

5.4.1 Baseline Model

We define the baseline model as a linear regression between the strength and the porosity of the sample. As expected, the porosity is highly predictive, but not perfectly predictive, of the strength, as shown in Fig 5.5. The in-sample R2 (i.e., the R2 evaluated on the training set) is 0.753 while the out-of-sample R2 (i.e., the R2 evaluated on the test set) is 0.752. The small gap between the in-sample and out-of-sample R2 shows that this baseline model generalizes very well.

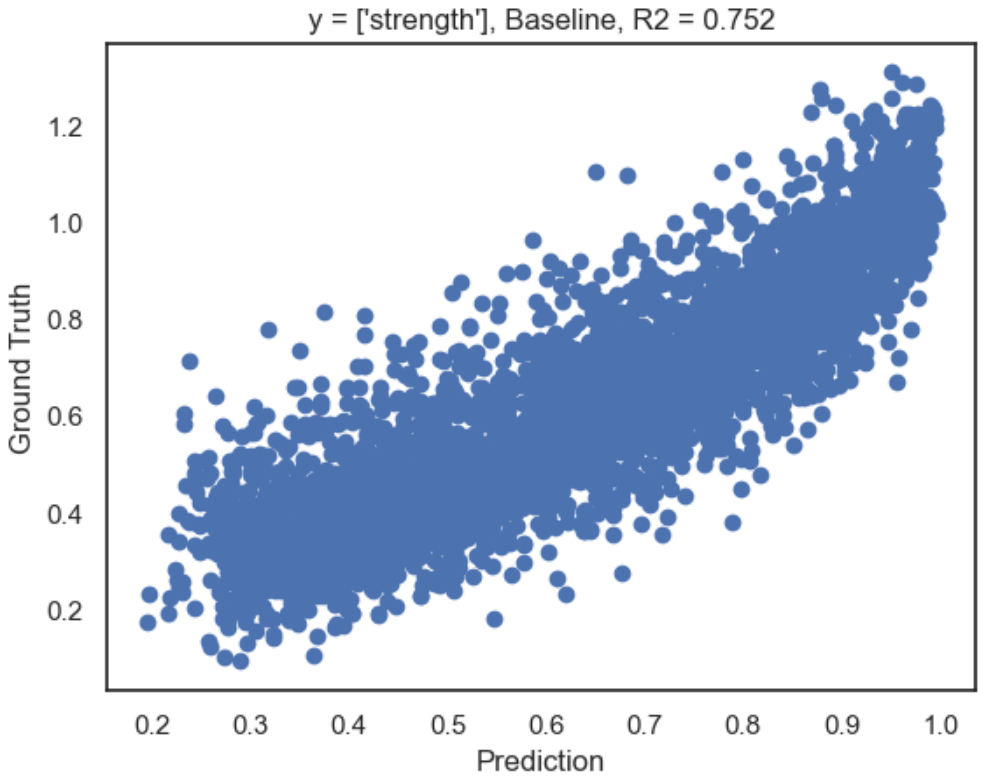

Figure 5.5: Comparison of the actual strength and the prediction of the baseline model.

For the ease of interpretation, we define an additional metrics as

$$\texttt{rel}\ \ R^2(y, \hat{y}_m, \hat{y}_b) = 1 - \frac{\sum_i^n (y_i - \hat{y}_{i,m})^2}{\sum_i^n (y_i - \hat{y}_{i,b})^2} = 1 - \frac{\text{MSE}(y, \hat{y}_m)}{\text{MSE}(y, \hat{y}_b)} \tag{5.9}$$

where $\hat{y}_m$ and $\hat{y}_b$ represent the prediction of the model and the prediction of the baseline model, respectively. Although this metrics is uncommon, it is quite intuitive. While the standard R2 measures the proportion of the variance that is explained by the model, this metrics measures the proportion of the remaining variance not explained by the baseline model that is explained by our new model. We call this metrics the relative R2.

5.4.2 Linear Regression

We build our first set of models using linear regression. Although linear regression is a simple model that can only capture linear relationships, it is a solid model that is easy to interpret and less likely to over-fit. However, since we have some highly correlated features, it is critical that we apply ridge regularization to our regression, especially when we include the vector features. We implement our linear regression with `scikit-learn` [38].

Table 5.1: The results of our linear regression models.

Feature Set	In-Sample R2	Out-of-Sample R2	Relative R2
Scalar Features	0.842	0.839	0.352
Scalar + Percentile	0.844	0.841	0.358
All Features	0.851	0.849	0.391

As shown in Table 5.1 and Fig 5.6, the features we have created are indeed predictive of the ultimate strength. The vector features refer to the two-point correlation function $S_2(r)$ and multi-scale disorder index $I_d(r)$, as well as their curve fit residuals. The percentile features refer to the 0-th, 5-th, 10-th, ..., 95-th, and 100-th percentile of the pore radius, as well as the two-point correlation function and multi-scale disorder index sampled at these distances. The scalar features refer to other features that do not fall into the category of vector features and percentile features. The scalar features, scalar + percentile features, and all (scalar + percentile + vector) features include 18, 81, and 1329 features, respectively.

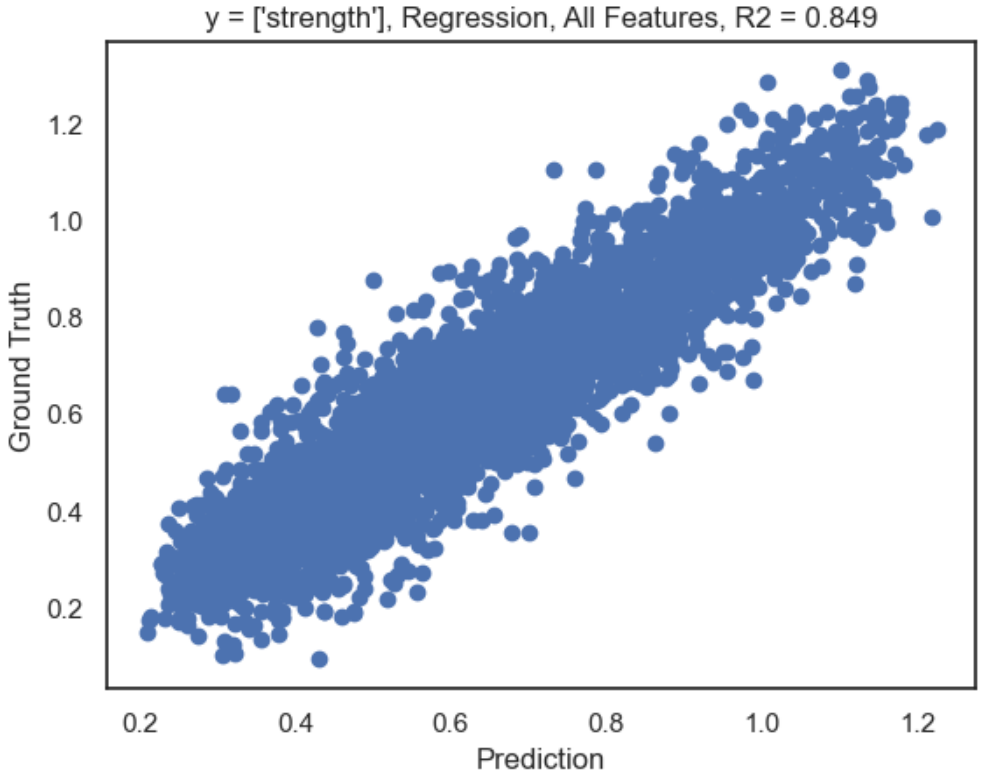

Figure 5.6: Comparison of the actual strength and the prediction of the linear regression model.

5.4.3 Linear Regression in Log-Scale

Similar to our first set of models, we build our second set of models using linear regression again. However, in this set of models, we apply logarithmic transformation to all of our variables before fitting the linear regression. The idea is to capture the non-linear relationship in the form of

$$Y = C \prod_{i=1}^{n} X_i^{\beta_i} \tag{5.10}$$

Following the same methods that we used in the previous sub-section, we obtain the following results as shown in Table 5.2 and Fig 5.7. Note that log-scale linear regression require strictly positive inputs. Therefore, features that contain zero or negative values, such as the fit residual of the two-point correlation function and the multi-scale disorder index cannot be used in these models. Interestingly, the accuracy of our linear regression models in log-scale is not as good as that of the simple linear regression. One potential reason is that the relationship in Equation 5.10 suggests that the strength is 0 when the porosity is 0, which is certainly not true.

Table 5.2: The results of our linear regression models with logarithmic transformation.

Feature Set	In-Sample R2	Out-of-Sample R2	Relative R2
Scalar Features	0.796	0.797	0.179
Scalar + Percentile	0.809	0.809	0.231
All Features	0.833	0.833	0.326

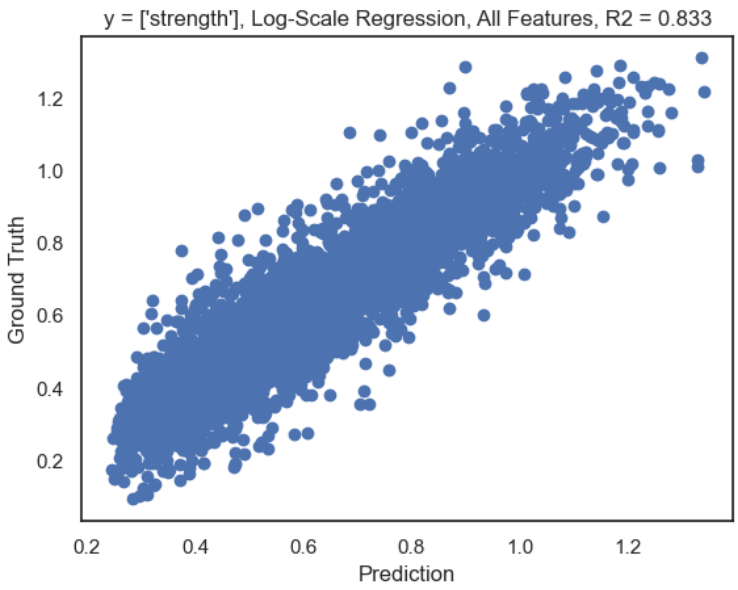

Figure 5.7: Comparison of the actual strength and the prediction of the linear regression model with logarithmic transformation.

5.4.4 Random Forest

We build the third set of models using random forest algorithm as implemented in scikit-learn `sklearn.ensemble.RandomForestRegressor`. Following the same methods that we used in the previous sub-sections, we obtain the following results as shown in Table 5.3 and Fig 5.8.

Table 5.3: The results of our random forest regression models.

Feature Set	In-Sample R2	Out-of-Sample R2	Relative R2
Scalar Features	0.986	0.902	0.604
Scalar + Percentile	0.985	0.902	0.603
All Features	0.985	0.900	0.598

The accuracy of our random forest regression models is noticeably higher than that of the previous linear regression models. However, contrary to the previous models, adding more features past the scalar features did not improve the accuracy of our random forest regression model. We also note the substantial gap between the in-sample R2 and out-of-sample R2, which suggests that there is some over-fitting in our model.

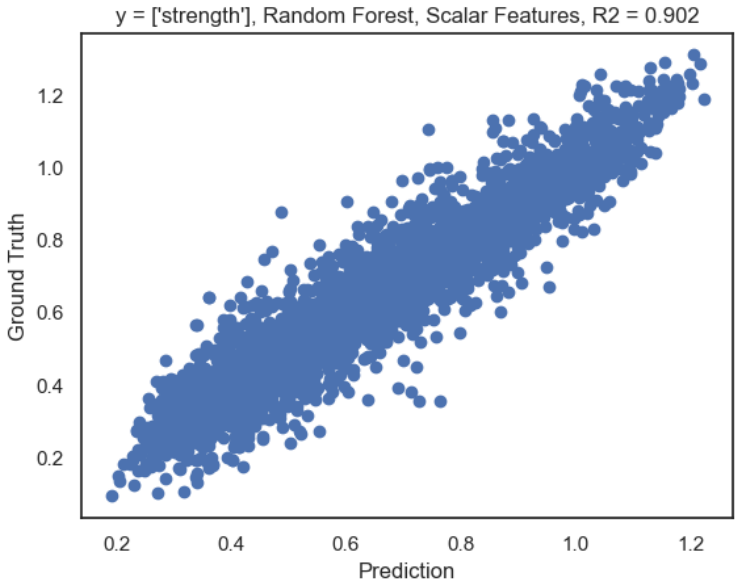

Figure 5.8: Comparison of the actual strength and the prediction of the random forest regression model.

5.4.5 Gradient Boosting Decision Trees

Finally, we build the fourth and last set of models using gradient boosting decision trees (GBDT) algorithm. We implement our GBDT models with `lightgbm` [39]. We follow the same methods that we used in the previous sub-sections with a minor modification. We further split the training set to obtain a small validation set to pass into the algorithm. With validation set, `lightgbm` will keep adding more trees, up to a specified limit, until the accuracy of the model on the validation set stops improving. The results of our GBDT models are shown in Table 5.4.

Table 5.4: The results of our gradient boosting decision trees models.

Feature Set	In-Sample R2	Out-of-Sample R2	Relative R2
Scalar Features	0.940	0.884	0.531
Scalar + Percentile	0.949	0.887	0.542
All Features	0.963	0.892	0.564

Once again, adding more features past the scalar features improves the accuracy of our GBDT models. Similar to the random forest models, our GBDT models have noticeably higher accuracy and over-fitting than the linear regression models. However, we should not be too worried about the over-fitting as our tree-based models are generalizing well enough, as evident from the out-of-sample R2.

Hyper-parameter Tuning

It is important to note that the results in Table 5.4 above are trained under the default hyper-parameters of `lightgbm`. We can further improve the accuracy of our model by tuning these hyper-parameters. We perform a grid search of the number of leaves `num_leaves` and minimum data points per leaf `min_data_in_leaf` to obtain our best hyper-parameters, which are 63 leaves and 20 minimum data points per leaf. These tuned hyper-parameters further improved the accuracy of our GBDT model, as shown in 5.5 and Fig 5.9. We select this model as our best model so far.

Table 5.5: The results of our best gradient boosting decision trees model with tuned hyper-parameters.

In-Sample R2	Out-of-Sample R2	Relative R2
0.983	0.904	0.611

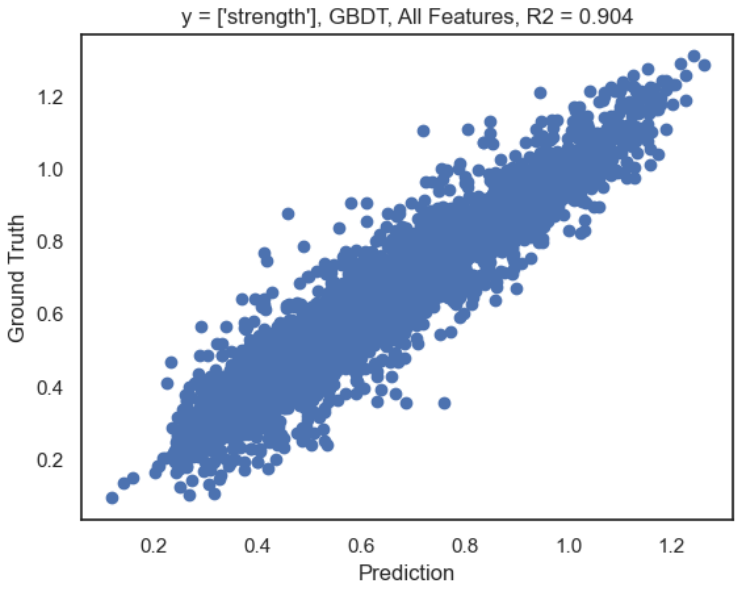

Figure 5.9: Comparison of the actual strength and the prediction of the gradient boosting decision trees model.

5.5 Conclusion

In this chapter, we have addressed our second and third research questions with the features we created and the statistical models we trained. We can parameterize the porous geometry of a two-dimensional sample with our defined features, such as the multi-scale disorder index, which are shown to be predictive of the sample's ultimate strength. We have also shown that we can use statistical models such as Gradient Boosting Decision Trees (GBDT) regression to create a structure function that bypasses the simulation process, making it more accessible and cost-effective at the cost of accuracy. The summary of our models are shown in Table 5.6.

Table 5.6: The summary of the results of our statistical models.

Model	Out-of-Sample R2
Baseline (Linear Regression with Porosity)	0.753
Linear Regression with Our Features	0.849
Random Forest with Our Features	0.902
GBDT with Our Features	0.904

Chapter 6

Conclusion and Discussions

In order to address the need for accurate, accessible, and cost-effective structure function relation of concrete, we have, in this thesis, formulated, implemented, and optimized the Semi-Grand Canonical Monte Carlo (SGCMC) simulation, which is a stochastic computational method to simulate both the fracture initiation and fracture propagation in a medium. We address the computational cost of the SGCMC simulation, which is one of its limitations, by optimizing its performance using various techniques, as detailed in Chapter 2. The first key optimization is the exploitation of the sparsity of the stiffness matrix, which reduces the space complexity of the algorithm from $O(n^2)$ to $O(n)$ and the time complexity of the algorithm from $O(n^{2.38})$ to $O(n^{1.24})$. The second key optimization is the scalable deployment of the simulation on super-computing infrastructures, such as MIT SuperCloud, in order to run multiple simulations simultaneously, up to the limit of our available resources.

Additionally, in order to create an even more accessible and cost-effective structure function relation of concrete, at the cost of some accuracy, we have demonstrated a pipeline to create statistical models that predict the strength of a two-dimensional porous material and bypass the simulation process with good accuracy. Our pipeline includes the sample generation, simulation, and result post-processing in Chapter 4 and the feature creation. data exploration, model training, model evaluation, and hyper-parameter tuning in Chapter 5. We have also defined some features, namely the two-point correlation function, the multi-scale disorder index, the pore size distribution as extracted by Circle Hough Transformation (CHT), the area moment of pores, and the derivatives of these features, that are predictive of a sample's ultimate strength and can be used to parameterize the two-dimensional porous geometry of the sample. Our best model so far is an ensemble of Gradient Boosting Decision Trees (GBDT) with our defined features and tuned hyper-parameters. This model has out-of-sample R2 of 0.904, explaining 61.1% of the variance not already explained by the baseline model, which is the linear regression between the strength and the porosity of the sample.

With our proposed accurate, accessible, and cost-effective ways to achieve structure function relation of concrete, we hope to help address the reduction of the environmental impact of concrete and the resilience of concrete structures.

6.1 Contributions

The contributions we make in this thesis can be divided into two parts. The first part is in performance optimization and scalable deployment of SGCMC simulations. Although SGCMC simulation has been previously formulated in the context of fracture mechanics in Ulm (2022) and Mulla (2020), our implementation focuses less on the formulation and more on the performance optimization, which is a necessary step to make SGCMC accessible and cost-effective. Not only is our optimized version of the algorithm 16 times faster in our use case, its time complexity has also reduced . Therefore, the speed-up we have gained will be even greater on larger inputs. In addition, our deployment of the simulation on super-computing infrastructures makes it scalable.

The second part of our contribution is in the statistical modeling of the strength of two-dimensional porous material. From a larger perspective, we have demonstrated a pipeline that utilizes the SGCMC simulation to create a data set and trains various statistical models with the data set. From the perspective of parameterization of the two-dimensional porous geometry, we have identified a few features that are useful and predictive of the strength of material. While we did not modify the definition of the two-point correlation function from the original works, we have refined the definition of the disorder index from Laubie (2017) to a multi-scale disorder index that better fits the general case where the pore radius is not a constant. We have also shown that we can further parameterize our vector features by fitting them with a functional form and take both the parameters of the fitted curve and the residual between the actual vector features and its fitted curve as features to predict the strength of the sample. Finally, we have created a gradient boosting decision trees (GBDT) model that utilizes our defined features to predict the strength of a porous geometry. Our model has out-of-sample R2 of 0.904, explaining up to 61.1% of the remaining variance that is not already explained by the baseline model.

6.2 Limitations

Although we are quite satisfied with our contributions, we acknowledge the following limitations in our work:

6.2.1 Limitations in the Implemented SGCMC Simulation

The first limitation in our implementation of the SGCMC simulation is the fact that our code only handles two-dimensional case. Although we made the decision to focus on two-dimensional case for the simplicity and the speed of computation, our performance optimizations should generalize well into the three-dimensional case with some extra work on the implementation.

The second limitation in our implementation is our implementation of the batch MC trials. Although the MC trials only take up a small fraction of computation time in the original code, it is now a larger fraction of the computation time after our optimization of

the NVT updates. In our implementation of the batch MC trials, the introduced overhead to ensure the serial correctness of the MC trials is significant and offsets most of the benefits from the batching of the MC trials. In retrospect, we should be able to reduce the overhead with better implementation and further improve the performance of our SGCMC code, especially for the smaller samples.

The third and final limitation in our implementation is the constant load increment during the NVT updates. In order to achieve satisfactory accuracy, we need sufficient NVT updates, resulting in smaller load increments between the NVT updates. However, as shown in Fig 4.2, the results of the early part of the simulations are the same across the different numbers of NVT updates. In fact, we can also see from Fig 3.2 that there is little spread across different instances of the simulations during this region where the Hooke's law holds true. We should be able to make our simulation more cost-effective without affecting the accuracy by spending less NVT updates in this region. Therefore, a dynamic way to determine the discretization of load increment per NVT step is definitely something worth looking into.

6.2.2 Limitations in the Data Set Generation

One limitation in our data set generation is, without a doubt, the fact that we did not have access to actual images of concrete under a microscope. Although we managed to generate our data set with artificially generated samples, we have made several assumptions, such as the perfectly circular pores and the uniform and independent distribution of pore center locations, that may not be true. Therefore, we need actual images data in order to accurately capture the mechanism of pore placements and the clustering of the pores in reality. In addition, we do not know the prior distribution of the underlying parameters (the prescribed porosity, the mean pore radius, and the pore radii standard deviation) of the sample generation. It would be an improvement over the uniform distribution over a prescribed range if we can obtain the actual distribution of these underlying parameters of the sample generation.

The other limitation in our data set generation is, in fact, the number of samples in our data set. As previously discussed in Section 4.4, it turns out that we don't have enough data to fully train our Gradient Boosting Decision Trees (GBDT) model. As shown in Fig 6.1, the accuracy of our model is still increasing with the number of training data points. Therefore, we can still further improve our model if we feed it with more training data.

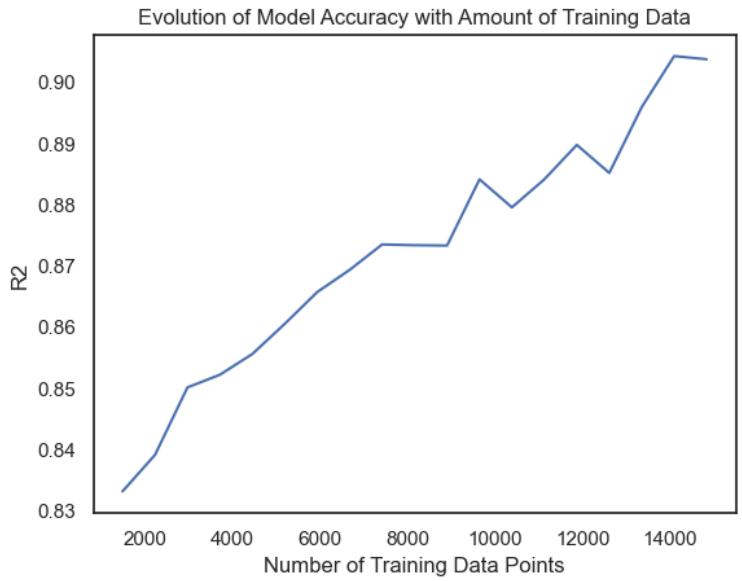

Figure 6.1: The evolution of our gradient boosting decision trees model accuracy with increasing number of training data points.

6.2.3 Limitations in the Statistical Modeling

The first limitation in our statistical modeling is the fact that we did not have time to properly train Neural Network (NN) models, especially the Convolutional Neural Networks (CNN). Since we are working with an image input, using the CNN is a very natural choice that we couldn't employ due to our time constraints, as well as the limited size of our data set, as evident from Fig 6.1. Since Convolutional Neural Network models are more complex than GBDT models, they will certainly require more training data. We hope that the future research can address this limitation.

The second limitation in our statistical modeling is our lack of features. It is important to note that the statistical modeling is an iterative process. Although we have defined several features that successfully parameterize the two-dimensional porous geometry of a sample and are predictive of the sample's strength, we can most certainly further improve our models with additional features. However, similar to our first limitation, we cannot do so due to our time constraints and hope to address this in the future research.

The third limitation in our statistical modeling is the lack of y-intercept in our linear regression in log-scale. As formulated in Equation 5.10, the linear regression in log-scale implies that the strength is 0 when the porosity (or other features) is 0, which is not true. However, we still think that there is value in capturing the non-linear relationship with some transformation of linear regression. Ideally, we would like to capture the non-linear relationship in the form of

$$Y = C_0 + \sum_{j=1}^{m} C_j \prod_{i=1}^{n} X_i^{\beta_{i,j}} \tag{6.1}$$

which would require a more complex implementation. We hope to further expand on this idea in the future research.

The fourth and final limitation in our statistical modeling is the curve fit parameter b of the multi-scale disorder index. As shown in 5.3, the value of this parameter is exactly 100 for most of our data points. While it is an interesting and important discovery, we do not have a conclusive explanation for it yet. We should certainly address the reason behind this discovery in the future research, either analytically or empirically with different sample sizes, sample generation methods, and ideally actual images data set.

6.3 Future Research

On top of addressing the previously discussed limitations, we propose a few future research directions as follows:

6.3.1 Quantifying Feature Importance

Although we have defined a set of features and shown that they are predictive of the sample's strength, we have not directly quantified the impact of each feature on our prediction, which is important to the interpretation of our model. There are multiple ways to quantify the feature importance. For example, we can use the regression coefficients in the linear regression model to interpret the importance of their associated features. In addition to the benefits in model interpretation, feature importance can also help us eliminate unimportant or redundant features in order to reduce over-fitting and improve the generalization of the model.

6.3.2 Multi-task Learning

While the ultimate strength is one of the most important property of material, there are other properties that are useful and need to be addressed in structure function relations, such as the ductility. Therefore, in order to achieve a comprehensive structure function relation, we need to extract these properties from our simulation results and include them as target y-variables in our statistical modeling. In addition, there is also a benefit in training our statistical models on multiple target y-variables simultaneously, as the domain information contained in the target y-variables can help improve the generalization of the Neural Network models [40].

Appendix A

Sparse Stiffness Matrix Assembly and NVT Solver

```
classdef NVT_utils
    methods (Static)

    %=========================================================
    function k_2pt = Two_PointK(nx,ny,Db,Lbx)
    %---------------------------------------------------------
    % Stiffness matrix of 2pt interaction ij
    %    INPUT: nx,ny = cos, sin
    %           Db    = bond energy
    %           Lbx   = bond length
    %    OUTPUT: 4 x 4 Matrix for (u_i,v_i,u_j,v_j)
    %=========================================================

        k_2pt(1:4,1:4) = [nx^2      nx*ny     -nx^2     -nx*ny
                          nx*ny     ny^2      -nx*ny    -ny^2
                         -nx^2     -nx*ny      nx^2      nx*ny
                         -nx*ny    -ny^2       nx*ny     ny^2];
        k_2pt = Db*Lbx^(-2) * k_2pt;

        return
    end

    %=========================================================
    function [i,j,v] = Two_PointK_diagonal(idx,HC,nx,ny,Db,Lbx)
    %---------------------------------------------------------
    % Stiffness matrix of 2pt interaction ij
    %    INPUT: idx   = particle index
    %           HC    = bond activation
    %           nx,ny = cos, sin
    %           Db    = bond energy
    %           Lbx   = bond length
```

```
%    OUTPUT: [i,j,v] of (u_i,v_i,u_j,v_j) for sparse matrix creation
%=============================================================
    NcT = length(nx); % number of connected particles
    k_2pt = zeros(2,2,NcT);
    k_2pt(1,1,:) = nx .^ 2;
    k_2pt(1,2,:) = nx .* ny;
    k_2pt(2,1,:) = nx .* ny;
    k_2pt(2,2,:) = ny .^ 2;
    k_2pt = k_2pt .* reshape(HC .* Db .* Lbx .^ (-2), 1, 1, []);
    k_2pt = sum(k_2pt, 3);
    v = reshape(k_2pt, [], 1);
    i = [2*idx-1; 2*idx; 2*idx-1; 2*idx];
    j = [2*idx-1; 2*idx-1; 2*idx; 2*idx];
    return
end

%=============================================================
function [i,j,v] = Two_PointK_offdiag(DhT,HC,nx,ny,Db,Lbx)
%-------------------------------------------------------------
% Stiffness matrix of 2pt interaction ij
%    INPUT: DhT   = bond connectivity
%           HC    = bond activation
%           nx,ny = cos, sin
%           Db    = bond energy
%           Lbx   = bond length
%    OUTPUT: [i,j,v] of (u_i,v_i,u_j,v_j) for sparse matrix creation
%=============================================================

    % not ideal but it works
    % there should be a way to vectorize this
    nx2 = - HC.* Db .* (Lbx .^ (-2)) .* (nx .^  2);
    nxy = - HC.* Db .* (Lbx .^ (-2)) .* (nx .* ny);
    ny2 = - HC.* Db .* (Lbx .^ (-2)) .* (ny .^  2);

    i1 = 2 .* DhT(:,1) - 1;
    j1 = 2 .* DhT(:,2) - 1;
    v1 = nx2;

    i2 = 2 .* DhT(:,1) - 1;
    j2 = 2 .* DhT(:,2) - 0;
    v2 = nxy;

    i3 = 2 .* DhT(:,1) - 0;
    j3 = 2 .* DhT(:,2) - 1;
    v3 = nxy;
```

```
77
78          i4 = 2 .* DhT(:,1) - 0;
79          j4 = 2 .* DhT(:,2) - 0;
80          v4 = ny2;
81
82          i5 = 2 .* DhT(:,2) - 1;
83          j5 = 2 .* DhT(:,1) - 1;
84          v5 = nx2;
85
86          i6 = 2 .* DhT(:,2) - 1;
87          j6 = 2 .* DhT(:,1) - 0;
88          v6 = nxy;
89
90          i7 = 2 .* DhT(:,2) - 0;
91          j7 = 2 .* DhT(:,1) - 1;
92          v7 = nxy;
93
94          i8 = 2 .* DhT(:,2) - 0;
95          j8 = 2 .* DhT(:,1) - 0;
96          v8 = ny2;
97
98
99          i = [i1;i2;i3;i4;i5;i6;i7;i8];
100         j = [j1;j2;j3;j4;j5;j6;j7;j8];
101         v = [v1;v2;v3;v4;v5;v6;v7;v8];
102         return
103     end
104
105     %============================================================
106     function f_2pt = Inc_Force(nx,ny,Db,Lb,lambda)
107     %------------------------------------------------------------
108     % Linear term for quadratic optimization term
109     %------------------------------------------------------------
110
111         f_2pt = Db*lambda/Lb * [-nx,-ny,nx,ny];
112         f_2pt = f_2pt';
113
114         return
115     end
116
117     %============================================================
118     function f_2pt = Inc_Force_vectorized(HC,nx,ny,Db,Lb,lambda)
119     %------------------------------------------------------------
120     % Linear term for quadratic optimization term
121     %------------------------------------------------------------
122         if ~isempty(HC)
123             f_2pt_x = sum(HC .* Db .* lambda ./ Lb .* nx);
```

```
            f_2pt_y = sum(HC .* Db .* lambda ./ Lb .* ny);
            f_2pt = [f_2pt_x;f_2pt_y];
        else
            f_2pt = [0;0];
        end
        return
    end
    end
end

classdef NVT_solver
    methods (Static)

    %=========================================================
    function Xi = sparse_solver(DhT,BhT,BhTL,BhTR,NC,BN,WWd,Db,x,y,nx
       ,ny,Lb,lambda,lambda_p)
    %---------------------------------------------------------
    % incremental NVT Solver
    %     INPUT: DhT  = 2-pt Bond connectivity
    %            ChT  = 2-pt connected particle array
    %            BhT  = 2-pt associated bond index array
    %            NC   = Bond activation (1 if bond is active, -1 if
       bond
    %            deactivated)
    %            BN   = Boundary Nodes
    %            Db   = Bond stiffnesses
    %            x,y  = coordinates
    %            nx,ny= cos, sin of all bonds
    %            Lb   = length of all bonds
    %            lambda = bond stretch
    %            lambda_p = plastic bond stretch
    %     OUTPUT: Xi  = incremental displacement of particles (du,dv)
    %
    %---------------------------------------------------------

        %% --- 1. Inititialization

        Ntot = length(x);
        NhT  = length(DhT(:,1));

        % k_2pt = zeros(4,4);
        % K_2pt = sparse(2*Ntot,2*Ntot);
        % f_2pt = zeros(4,1);       % for "incremental" minimization
        fff   = zeros(2*Ntot,1);    % for "incremental" minimization
        % FFF   = zeros(2*Ntot,1);

        i_arr_diag = zeros(4*Ntot,1);
```

```
        j_arr_diag = zeros(4*Ntot,1);
        v_arr_diag = zeros(4*Ntot,1);

        for idx = 1:Ntot % start loop over all particles
            nxi = nx(BhT{idx});
            nyi = ny(BhT{idx});
            Dbi = Db(BhT{idx});
            Lbi = Lb(BhT{idx});
            % lambdai = lambda(BhT{idx})-lambda_p(BhT{idx}); %---
               elastic stretch
            iHC = 1/2*(1+NC(BhT{idx})); % --- (1 if bond active, 0 if
                bond broken)
            [i_arr_diag(4*idx-3:4*idx),j_arr_diag(4*idx-3:4*idx),
               v_arr_diag(4*idx-3:4*idx)] = NVT_utils.
               Two_PointK_diagonal(idx,iHC,nxi,nyi,Dbi,Lbi);

            nxi = nx(BhTL{idx});
            nyi = ny(BhTL{idx});
            Dbi = Db(BhTL{idx});
            Lbi = Lb(BhTL{idx});
            lambdai = lambda(BhTL{idx})-lambda_p(BhTL{idx}); %---
               elastic stretch
            iHC = 1/2*(1+NC(BhTL{idx})); % --- (1 if bond active, 0
               if bond broken)
            f_2pt_L = NVT_utils.Inc_Force_vectorized(iHC,nxi,nyi,Dbi,
               Lbi,lambdai);

            nxi = nx(BhTR{idx});
            nyi = ny(BhTR{idx});
            Dbi = Db(BhTR{idx});
            Lbi = Lb(BhTR{idx});
            lambdai = lambda(BhTR{idx})-lambda_p(BhTR{idx}); %---
               elastic stretch
            iHC = 1/2*(1+NC(BhTR{idx})); % --- (1 if bond active, 0
               if bond broken)
            f_2pt_R = NVT_utils.Inc_Force_vectorized(iHC,nxi,nyi,Dbi,
               Lbi,lambdai);

            f_2pt = f_2pt_L - f_2pt_R;
            fff(2*idx-1) = f_2pt(1);
            fff(2*idx-0) = f_2pt(2);
        end

        HC = (NC+1)./2; % --- (1 if bond active, 0 if bond broken)
        [i_arr_offdiag,j_arr_offdiag,v_arr_offdiag] = NVT_utils.
           Two_PointK_offdiag(DhT,HC,nx,ny,Db,Lb);
        i_arr = [i_arr_diag;i_arr_offdiag];
```

```
205         j_arr = [j_arr_diag;j_arr_offdiag];
206         v_arr = [v_arr_diag;v_arr_offdiag];
207         K_2pt = sparse(i_arr,j_arr,v_arr);
208 
209         %% --- 4. Consideration of DISPLACEMENT Boundary Conditions
210 
211             NB  = length(BN);
212             K1 = K_2pt;
213             FFF = -K_2pt*WWd;
214 
215             traceK = trace(K_2pt)/(2*Ntot); %conditioning
216             Lambda = traceK * 10^5;
217 
218             for k = 1:NB
219                 K1(2*BN(k)-1,2*BN(k)-1) = K1(2*BN(k)-1,2*BN(k)-1) +
                        Lambda;
220                 K1(2*BN(k)-0,2*BN(k)-0) = K1(2*BN(k)-0,2*BN(k)-0) +
                        Lambda;
221             end
222 
223         %% --- 5. Minimization Problem
224             % quick and dirty bounds - expecting 200 x 200 size
                    lattice
225             % restrict incremental displacement to 10% of the lattice
                    size
226             lb = -20 * ones(length(x) + length(y), 1);
227             ub =  20 * ones(length(x) + length(y), 1);
228             if condest(K1) < 1e9
229                 Xi = K1\(FFF-fff) + WWd;
230             else
231                 %--- quadratic minimization
232                 Xi = quadprog(K_2pt,fff,[],[],K1,FFF-fff,[],[],[],
                        optimset('Display', 'off')) + WWd;
233             end
234 
235             % re-run with bounded quadprog if necessary
236             if any(isnan(Xi)) || any(Xi < lb) || any(Xi > ub)
237                 % bound quadprog is slow, only use when Xi is
                        unrealistically large
238                 Xi = quadprog(K_2pt,fff,[],[],K1,FFF-fff,lb,ub,[],
                        optimset('Display', 'off')) + WWd;
239             end
240 
241         return
242     end
243     end
244 end
```

References

[1] Y. Guo, "Analysis on concrete construction technology in civil engineering construction," *Journal of Physics: Conference Series*, vol. 2011, p. 012 023, Sep. 2021. DOI: 10.1088/1742-6596/2011/1/012023.

[2] K. Neupane, D. Chalmers, and P. Kidd, "High-strength geopolymer concrete-properties, advantages and challenges," *Advances in Materials*, vol. 7, Jun. 2018. DOI: 10.11648/j.am.20180702.11.

[3] K. P. Mehta, "Reducing the environmental impact of concrete," *Concrete international*, vol. 23, no. 10, pp. 61–66, 2001.

[4] J. J. Biernacki, J. W. Bullard, G. Sant, *et al.*, "Cements in the 21(st) century: Challenges, perspectives, and opportunities," en, *J Am Ceram Soc*, vol. 100, no. 7, pp. 2746–2773, May 2017.

[5] K. Keremidis, "Kinetic temperature of structures for resilience, instability and failure analysis of building systems," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, 2022.

[6] A. H. Nilson, D. Darwin, and 1. Dolan Charles W., *Design of concrete structures*, English, 13th ed. Boston: McGraw-Hill Higher Education, 2004, ISBN: 9780072483055.

[7] X. Chen and S. Wu, "Influence of water-to-cement ratio and curing period on pore structure of cement mortar," *Construction and Building Materials*, vol. 38, pp. 804–812, 2013, 25th Anniversary Session for ACI 228 – Building on the Past for the Future of NDT of Concrete, ISSN: 0950-0618. DOI: https://doi.org/10.1016/j.conbuildmat.2012.09.058.

[8] A. T. Zehnder, "Griffith theory of fracture," in *Encyclopedia of Tribology*, Q. J. Wang and Y.-W. Chung, Eds. Boston, MA: Springer US, 2013, pp. 1570–1573, ISBN: 978-0-387-92897-5. DOI: 10.1007/978-0-387-92897-5_259.

[9] T. Mulla, R. J.-M. Pellenq, and F.-J. Ulm, "Fluctuation-based fracture mechanics of heterogeneous materials," *Phys. Rev. E*, vol. 106, p. 065 003, 6 Dec. 2022. DOI: 10.1103/PhysRevE.106.065003.

[10] H. Laubie, F. Radjai, R. Pellenq, and F.-J. Ulm, "Stress transmission and failure in disordered porous media," *Phys. Rev. Lett.*, vol. 119, p. 075 501, 7 Aug. 2017. DOI: 10.1103/PhysRevLett.119.075501.

[11] C. R. Gagg, "Cement and concrete as an engineering material: An historic appraisal and case study analysis," *Engineering Failure Analysis*, vol. 40, pp. 114–140, 2014, ISSN: 1350-6307. DOI: https://doi.org/10.1016/j.engfailanal.2014.02.004.

[12] F.-J. Ulm, "1.035: Mechanics of materials," Massachusetts Institute of Technology, Cambridge, MA, Lecture Notes, 2018.

[13] P. Yu, Z. Ren, Z. Chen, and S. P. A. Bordas, "A multiscale finite element model for prediction of tensile strength of concrete," *Finite Elements in Analysis and Design*, vol. 215, p. 103 877, 2023, ISSN: 0168-874X. DOI: https://doi.org/10.1016/j.finel.2022.103877.

[14] W. McGuire, R. H. Gallagher, and R. D. Ziemian, *Matrix Structural Analysis*, 2nd ed. John Wiley & Sons, Inc., 2000.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd. The MIT Press, 2001, ISBN: 0262032937.

[16] B. Efron and T. Hastie, *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science* (Institute of Mathematical Statistics Monographs). Cambridge University Press, 2016.

[17] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics). New York, NY, USA: Springer New York Inc., 2001.

[18] J. Gareth, W. Daniela, H. Trevor, and T. Robert, *An introduction to statistical learning : with applications in R*. New York : Springer, 2013.

[19] T. Mulla, S. Moeini, K. Ioannidou, R. J.-M. Pellenq, and F.-J. Ulm, "Phase diagram of brittle fracture in the semi-grand-canonical ensemble," *Phys. Rev. E*, vol. 103, p. 013 003, 1 Jan. 2021. DOI: 10.1103/PhysRevE.103.013003.

[20] T. M. Inc., *Matlab version: 9.13.0 (r2022b)*, Natick, Massachusetts, United States, 2022.

[21] T. M. Inc., *Matlab documentation*, Natick, Massachusetts, United States, 2022. URL: https://www.mathworks.com/help/matlab/index.html.

[22] T. M. Inc., *Optimization toolbox*, Natick, Massachusetts, United States, 2022.

[23] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions," *Journal of Symbolic Computation*, vol. 9, no. 3, pp. 251–280, 1990, Computational algebraic complexity editorial, ISSN: 0747-7171. DOI: https://doi.org/10.1016/S0747-7171(08)80013-2.

[24] A. Reuther, J. Kepner, C. Byun, *et al.*, "Interactive supercomputing on 40,000 cores for machine learning and data analysis," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, IEEE, 2018, pp. 1–6.

[25] K. Ioannidou, M. Kanduč, L. Li, D. Frenkel, J. Dobnikar, and E. Del Gado, "The crucial effect of early-stage gelation on the mechanical properties of cement hydrates," *Nature Communications*, vol. 7, no. 1, p. 12 106, Jun. 2016, ISSN: 2041-1723. DOI: 10.1038/ncomms12106.

[26] J. Zhu, R. Zhang, Y. Zhang, and F. He, "The fractal characteristics of pore size distribution in cement-based materials and its effect on gas permeability," *Scientific Reports*, vol. 9, no. 1, p. 17 191, Nov. 2019, ISSN: 2045-2322. DOI: 10.1038/s41598-019-53828-5.

[27] J. Zhang and P. C. Taylor, "Pore size distribution in cement pastes in relation to freeze-thaw distress," *Journal of Materials in Civil Engineering*, vol. 27, no. 3, p. 04 014 123, 2015. DOI: 10.1061/(ASCE)MT.1943-5533.0001053.

[28] G. Fuyuan, Z. Dawei, S. Evdon, and U. Tamon, "Empirical estimation of pore size distribution in cement, mortar, and concrete," *Journal of Materials in Civil Engineering*, vol. 26, no. 7, p. 04 014 023, Jul. 2014. DOI: 10.1061/(ASCE)MT.1943-5533.0000945.

[29] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[30] M. Kerscher, I. Szapudi, and A. S. Szalay, "A comparison of estimators for the two-point correlation function," *The Astrophysical Journal*, vol. 535, no. 1, p. L13, May 2000. DOI: 10.1086/312702.

[31] C. L. Y. Yeong and S. Torquato, "Reconstructing random media," *Phys. Rev. E*, vol. 57, pp. 495–506, 1 Jan. 1998. DOI: 10.1103/PhysRevE.57.495.

[32] Y. Jiao, F. H. Stillinger, and S. Torquato, "Modeling heterogeneous materials via two-point correlation functions: Basic principles," *Phys. Rev. E*, vol. 76, p. 031 110, 3 Sep. 2007. DOI: 10.1103/PhysRevE.76.031110.

[33] S. Benito, *Twopointprobability - two-point probability functions*, https://www.mathworks.com/matlabcentral/fileexchange/86852-twopointprobability-two-point-probability-functions, [Online; accessed October, 2022], 2021.

[34] H. Yuen, J. Princen, J. Illingworth, and J. Kittler, "Comparative study of hough transform methods for circle finding," *Image and Vision Computing*, vol. 8, no. 1, pp. 71–77, 1990, ISSN: 0262-8856. DOI: https://doi.org/10.1016/0262-8856(90)90059-E.

[35] T. Atherton and D. Kerbyson, "Size invariant circle detection," *Image and Vision Computing*, vol. 17, no. 11, pp. 795–803, 1999, ISSN: 0262-8856. DOI: https://doi.org/10.1016/S0262-8856(98)00160-7.

[36] T. M. Inc., *Image processing toolbox*, Natick, Massachusetts, United States, 2022.

[37] T. pandas development team, *Pandas-dev/pandas: Pandas*, version latest, Feb. 2020. DOI: 10.5281/zenodo.3509134.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, pp. 3146–3154, 2017.

[40] R. Caruana, "Multitask learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, Jul. 1997, ISSN: 1573-0565. DOI: 10.1023/A:1007379606734.