COMPUTATIONAL ANALYSIS OF FLUID FLOW IN 2-D & 3-D PORE GEOMETRY

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF GEOPHYSICS

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

NATTAVADEE SRISUTTHIYAKORN

JUNE 2018

Abstract

The motivation of this dissertation is to improve understanding of the geometrical parameters controlling flow properties and to develop toolkits to assist in this task in order to further advance Digital Rock Physics. With digital 2-D or 3-D images we can now investigate rock geometry thoroughly. Geometrical parameters such as cross-section geometry, tortuosity, pore size distribution, and grain size distribution will be discussed in detail in each chapter. Then, combining the knowledge of cross-section geometry and tortuosity, I show that the pore size distribution is the missing parameter crucial for accurately predicting permeability in porous media and I derive the revised Kozeny-Carman equation. Furthermore, the growing number of digital microstructures makes data analysis via machine learning possible. I will also discuss how to employ Machine Learning in Digital Rock Physics on permeability prediction from digital microstructures. These various approaches are designed to advance our fundamental understanding of rock geometry, and to determine the topological factors that are most relevant to the geophysical properties that we wish to simulate.

Acknowledgement

First and foremost, I would like to acknowledge my dissertation reading committee members – Professor Gary Mavko, Professor Tapan Mukerji, Dr. Jack Dvorkin, and Professor Dustin Schroeder. Additionally, I would like to thank Dr. Sander Hunter for serving on my defense oral committee, and Professor Anthony Kovscek for serving as my Ph.D. defense chair. I am grateful for their supports and encouragements on my Ph.D. path.

I am very fortunate to have been working with my advisor Professor Gary Mavko. My life in the past five years has been an invaluable and memorable experience due to him. He is the best example of the perfect advisor that any student can hope for. Every discussion with him is inspiring and enlightening. Whenever I faced any obstacle or encountered a life-changing event, Gary and his wife Barbara were always there, reaching out and supporting me. My heart felt thanks to both of them.

Professor Amos Nur, thank you so much for giving me the opportunity to join the SRB group. I still remember the first time we met in Bangkok discussing different research topics. This dissertation topic is also inspired by his curiosity regarding the prediction of physical properties from 2-D images.

I also would like to thank Professor Tapan Mukerji and Dr. Jack Dvorkin. Together with Professor Gary they are the three musketeers of the SRB group. Many of the creative ideas I employed in my dissertation stemmed from discussions with them. Thank you so much for helping me grow as a researcher.

Dr. Nishank Saxena, Dr. Ronny Hofmann, and Dr. Sander Hunter mentored me during the Summer of 2016, when I was an intern of the Petrophysics and Geomechanics team at the Shell Technology Center Houston. I always enjoy the time we spend exchanging ideas about the future of digital rock physics.

Dr. Ritu Sarker and Dr. Stephan Gelinsky mentored me during the Summer of 2017 when I was an intern of QI team at Shell International Exploration and Production. I have to thank them for taking such great care of me during the entire Summer. They were wonderful examples of great mentors.

Dr. Evelin Sullivan has been my writing tutor and my friend for the past three years. She patiently taught me the art of good writing. We enjoy talking about running and exercising, and she always amazes me with how she maintains an energetic lifestyle.

I thank my SRB and School of Earth Sciences colleagues for their friendship in the basement of Mitchell: Amrita, Danica, Adam, Nishank, Chisato, Yuki, Sabrina, Priyanka, Humberto, Pinar, Iris, Abdulla, Abrar, Salma, Vishal, Juan, Chen, Wei, Ankush, Krongrath, Dulcie, Beibei, Chao, Tianze, Jens, Meng, Mathias, Jason, Mustafa, Lei, Larry and many more.

My life at Stanford would have been lonelier without my friends from the Stanford Thai Student Association. We had a great time spending Friday nights together playing board games, going to Tahoe together skiing, and organizing Thai cultural events.

I was blessed with the friendship of some truly special people. I want to thank Nui Wiriya Thongsomboon, Amita Tiyaboonchai, and Kloy Kanteera Tepartimagorn. They are always there for me even at the hardest times.

To Jack Visutipol, thank you so much for your caring and encouragement. I cherish every moment we have spent together. Jack has made a tremendous effort in taking countless trips to Stanford and Houston in the past five years to support me.

Apart from Professor Gary, I would like to dedicate this thesis to my family, Somchai Srisutthiyakorn, Jiravadee Srisutthiyakorn, and Prin Srisutthiyakorn. Thank you for your unconditional support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter provides the background and addresses trends and challenges in digital rock physics. Then, it discusses the motivation of the dissertation and the overview of the dissertation.

1.1 BACKGROUND

As Mavko et al. states, Rock Physics is the study of “the relationships between geophysical observations and the underlying physical properties of rocks, such as composition, porosity, and pore fluid content”. The foundation of Rock Physics is to discover and to understand such physical relations through the combination of theoretical models and laboratory measurements.

Digital Rock Physics utilizes digital 2-D or 3-D images of the complex rock microstructures (i.e. porous media) to understand these physical relations, and it has emerged as a robust way to simulate physical processes and determine physical properties such as effective elastic and transport properties at microscale. Numerical simulations in Digital Rock Physics include, for example, finite element, finite difference, and Lattice Boltzmann flow simulation (Garboczi, 1998; Arns et al., 2002; Keehm, 2003; Andrä et al., 2013a).

Digital 3-D volumes of rock microstructures are acquired from 3-D micro x-ray computer tomography (μXCT), which requires several steps of image filtering and segmentation before the final 3-D binary-segmented volumes are obtained (Sezgin and Sankur, 2004; Andrä et al., 2013a). Andrä et al. (2013) showed that effective elastic and transport properties simulated from the 3-D binary volumes are in good quantitative agreement with laboratory measurements, even though the analyses are performed at a different scale (cm in laboratory measurements vs. μm to mm in numerical simulations).

Digital 2-D images of rock microstructures can be scanned from 2-D thin sections or obtained from slicing digital 3-D μXCT volumes. A thin section is an approximately 30 μm-thick slice of rock attached to a glass slide with epoxy. The segmentation of scanned thin sections is challenging since the 2-D thin sections are not truly 2-D as a result of the wedge areas being covered by epoxy. Also, we cannot directly predict same physical properties that require 3-D geometry such as permeability, without using some transforms. Despite these drawbacks of digital 2-D images, they serve a purpose: they are widely available and inexpensive and are offering higher resolution and larger field of view than the 3-D μXCT volumes. 2-D images from a Scanning Electron Microscope (SEM) is also useful to quantify microporosity (Flavio S. Anselmetti, 1998).

Numerical simulations have several advantages over laboratory measurements. For one, the numerical simulations take less time than laboratory measurements. For another, they can provide accurate measurements of physical properties without damaging the sample, which is inevitable in testing friable sands or oil sands (Dvorkin et al., 2008). They also allow us to obtain the "trend" of physical relations from subvolumes (i.e. porosity vs. permeability, porosity vs. electrical formation factor, and porosity vs. elastic moduli) (Dvorkin et al., 2011). The analysis of the 3-D binary segmented volumes can also be enhanced by various analytical techniques such as geometrical measurements, porous media construction and alteration, and machine learning. These techniques allow us to extract parameters from digital 3-D volumes that cannot be extracted through conventional laboratory experiments.

1.2 TRENDS AND CHALLENGES

- **Towards understanding the "geometry"**

Before the age of Digital Rock Physics, rock "geometry" often served as a fitting parameter in empirical or semi-theoretical relations. Many of the geometric parameters cannot be measured directly in the laboratory. For example, there is no practical way to measure tortuosity or the number of contacts per grain in the laboratory settings. Therefore, rock geometry became a mysterious parameter and was treated as such.

With digital 2-D or 3-D images we can now investigate rock geometry thoroughly. The previously mentioned parameters can now be calculated directly from the digital images, and we are moving towards a greater understanding of the role played by rock geometry in elastic and transport properties.

- **Towards higher resolution and larger scale digital images**

Digital Rock Physics has rapidly emerged as a robust way to simulate physical processes because of the availability of high resolution digital 3-D volumes. The main question regarding any research in Digital Rock Physics is whether the digital volumes in the study reach the representative elementary volume (REV), the volume necessary to ensure that simulated physical properties are accurate. The REV varies from property to property. For example, it is easier to reach REV for porosity than to reach it for permeability. The REV also varies from microstructure to microstructure. It is easier to reach REV for homogenous microstructure than for heterogenous microstructure.

Apart from the REV for the digital microstructure, it is also important to consider that the physical simulations themselves must have a certain minimum required number of voxels for the simulation result to be free from boundary condition effects. For example, if we want to measure the grains' diameters, measuring grain diameter at the boundary of the image will distort the distribution.

Improvements in computational power now enable researchers to simulate physical properties on higher resolution and larger scale digital samples. However, larger volumes require longer computational time. This raises the question of whether for cases in which it is not possible to simulate the physical properties on the REV volume, it might be possible to simulate the physical properties on small samples and then upscale those samples.

- **Towards Digital images analysis via machine learning**

Machine learning in Digital Rock Physics is another prominent trend. The growing number of digital microstructures makes data analysis via machine learning possible. For example, machine learning can be used for classification of minerals and rock types and prediction of geophysical properties. The machine can be trained to recognize specific patterns in 2-D and 3-D images to predict physical properties such as permeability.

- **Towards realistic numerical simulations imitating the geological processes.**

As the research in Digital Rock Physics grows, we expect to see more research with numerical simulations imitating dynamic geological processes such as deposition, compaction, and cementation. However, as the simulations get more complicated, computational power has to be increased as well to handle complex and dynamic simulations.

1.3 MOTIVATIONS

The main motivation of this dissertation is to improve understanding of the geometrical parameters controlling flow properties and to develop toolkits to assist in this task in order to further advance Digital Rock Physics. Geometrical parameters such as cross-section geometry, tortuosity and pore size distribution will be discussed in detail in each chapter. After extracting geometrical parameters, I also discuss how to employ Machine Learning in Digital Rock Physics on permeability prediction from digital microstructures. These various approaches are designed to advance our fundamental understanding of rock geometry, and to determine the topological factors that are most relevant to the geophysical properties that we wish to simulate. After presenting the understanding of geometrical parameters I gained from digital 3-D volumes, I apply this knowledge to predict permeability from digital 2-D images.

Conventionally, we use digital 2-D images only for obtaining porosity, mineralogy, and degree of cementation. To increase the use of digital 2-D images, important questions are (1) what geometrical features can we extract from the 2-D images, and (2) do they provide relevant information about 3-D effective elastic and transport properties? These effective properties largely depend on rock geometry. However, we still lack knowledge of the difference between 2-D and 3-D rock geometries, which keeps us from understanding how these differences are manifested in 2-D and 3-D simulated results of effective elastic and transport properties.

1.4 PREDICTING 3-D ROCK PROPERTIES FROM 2-D IMAGES

The use of high-resolution 2-D images for rock property prediction has long been a challenge in rock physics since they do not contain the complete rock geometry information, such as where the grains are in contact and how the pores are connected to form a network. For instance, a 2-D slice of closely packed equal spheres shows mostly isolated spheres unless the slice is cut right where the spheres are in contact. Rock properties such as bulk modulus, shear modulus, and permeability are therefore difficult to predict directly from 2-D images. Conventionally, these properties are obtained from core measurements, or alternatively, from numerical simulations on 3-D Micro X-ray Computed Tomography (μXCT) images. However, core measurements in the laboratory are time consuming and often inflict irreversible changes to the rock matrix. Acquiring 3-D μXCT images is also not part of the routine core analysis workflow.

There are, however, advantages to using 2-D thin sections. They are widely available and inexpensive, since they are often produced routinely as a part of the core analysis workflow. They also offer higher resolution images and a larger field of view compared to the 3-D μXCT images. For example, if the computing limitation is at x voxels, the square 2-D image will have a maximum size of $\sqrt{x}$ and the cube 3-D image will have a maximum size of $\sqrt[3]{x}$. Similarly, the resolution of microscope imaging is Abbe's limit – half the wavelength of the light source, whereas μXCT images are limited by the resolution of the sensor – currently ~1μm.

Various approaches can be employed to estimate 3-D rock properties. These approaches include (1) empirical or theoretical relations by simulating 2-D rock properties and then transforming them into 3-D rock properties, (2) reconstructing a 3-D binary image and then simulating 3-D rock properties, and (3) directly predicting 3-D rock properties by employing machine learning or using the Revised Kozeny-Carman equation (Figure 1-1).

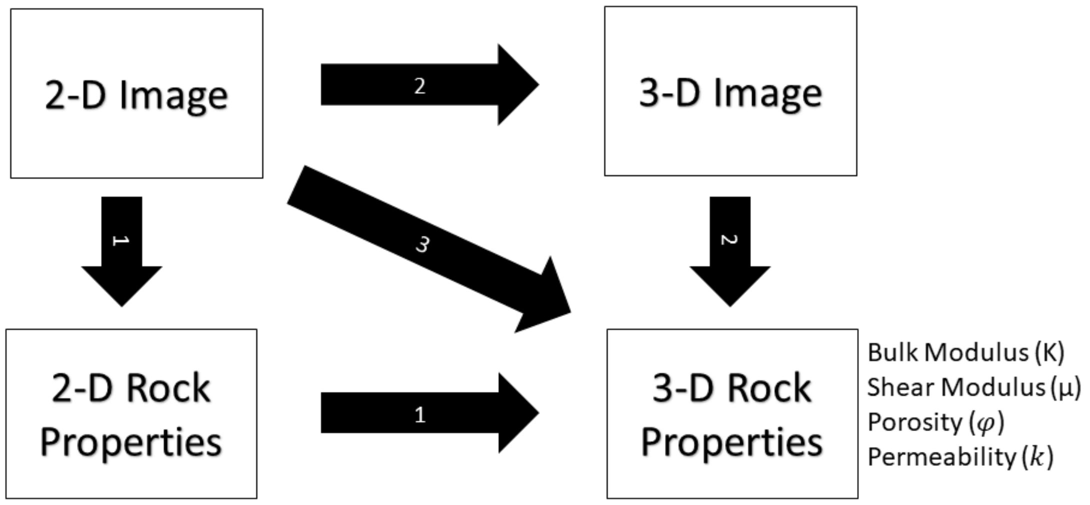


Figure 1-1: Multiple approaches to predict 3-D rock elastic and flow properties from 2-D images.

1.4.1 Empirical or theoretical relations

For the first approach, Berryman and Blair (1986) presented a method for estimating permeability from 2-D thin sections by combining electrical formation factor with a form of the Kozeny-Carman equation. Lock et al. (2002) proposed a method to predict permeability by measuring areas and perimeters of individual pores from 2-D thin sections. The Saxena et al. (2017) method involves two steps: (1) simulating permeability of the thin section for flow normal to the face using the Lattice Boltzmann Method (LBM) flow or the finite element method, and (2) applying 2D to 3D transform using calibration. For elastic properties, Saxena and Mavko (2016) predicted bulk and shear modulus by power law transform between the effective moduli obtained from 2-D and 3-D simulations.

1.4.2 Reconstructing a 3-D binary image

Previous attempts of 3-D images reconstruction can be classified into 2 categories: (1) using geostatistics-based method, and (2) using process-based method. Geostatistics-based reconstruction uses 2-D thin sections directly as inputs, whereas process-based reconstruction requires user to estimate input parameters such as porosity, the type and amount of cementation, grain size distribution, coefficient of friction, and visual estimation of compaction, all from 2-D thin sections.

Adler et al. (1990) reconstructed artificial porous media by using linear and non-linear filters of Gaussian random fields to match the porosity and correlation functions of Fontainebleau sandstone. Yeong and Torquato (1998) attempted to create 3-D reconstructed images of Fontainebleau sandstone by using a two-point probability function and a lineal-path function. Manwart et al. (2000) added pore size distribution to geostatistics-based reconstruction. Recently, a number of authors have proposed to use single normal equation simulation (SNESIM) and sequential indicator simulation (SISIM) for reconstructing 3-D images (Keehm, 2004; Okabe and Blunt, 2004, 2005; Kainourgiakis et al., 2005). The SNESIM technique was introduced in geostatistics to reconstruct field scale structures such as channels (Guardiano and Srivastava, 1993; Strebelle, 2002; Strebelle et al., 2003). SNESIM and SISIM simulations scan through the training images to build the search tree of data events, which can be used later to calculate conditional probabilities at each grid node. By storing data events in the tree structure, the size of memory required will grow exponentially. The limited size of patterns prevents SNESIM from capturing large scale features. Strebelle (2002) introduced multigrids to solve this problem, yet, there is still a problem with artifacts (Mariethoz et al., 2010). Direct sampling helps solve the memory issues by directly sampling the training images randomly, yet conditioned to the data event. Statistically, direct sampling can be recognized as a Markov chain with high number of neighborhood (Mariethoz and Caers, 2014). Similar to direct sampling, Wu et al. used a third-order Markov random field to generate 3-D images of the pore space from 2-D. (Wu et al., 2006).

For previous works on process-based reconstruction, Øren and Bakke (2002) included physical processes (sedimentation, compaction, and diagenesis) into their process-based 3-D reconstructions. From 2-D thin sections, they extract porosity, the type and amount of cementation, grain size distribution, and visual estimation of compaction. The limitation of this method is that it only considers spherical grains, and the physical process is limited to clastic sediments. Jin et al. developed physics-based reconstruction of sedimentary rocks based on grain size distribution, porosity, the type and amount of cementation, the coefficient of friction, the bound strength parameters, and the grain stiffness moduli (Jin et al., 2003, 2008). Sain (2011) implemented granular dynamics to construct granular packs and consolidated microstructures based on porosity, coordination number and stress state. The physics-based reconstructions can produce stress-strain relationship, but many parameters such as coefficient of friction and coordination number cannot be obtained from 2-D images.

1.4.3 Directly predicting 3-D rock properties

The third approach is an ongoing research and is a major part of my dissertation. Chapter 6 addresses the Revised Kozeny-Carman method and Chapter 7 employs machine learning to predict permeability from 2-D/3-D binary segmented images.

1.5 DISSERTATION OVERVIEW

Chapter 2 Tools and Techniques in Digital Rock Physics

This chapter focuses on the common tools and techniques used in Digital Rock Physics such as artificial microstructure construction, microstructure alternation, and microstructure geometric measurements.

Chapter 3 Review of the Kozeny-Carman Equation

This chapter presents the literature review of the Kozeny-Carman Equation, including the derivation and application of the equation. The Kozeny-Carman equation is a semi-empirical model relating permeability in single-phase flow to geometric measurements such as porosity, specific surface area, tortuosity, and a geometric factor. Through the Kozeny-Carman equation, we can better understand how the rock geometry relates to permeability.

Chapter 4 Cross-Section Geometry

The purpose of this chapter is to understand the effect of cross-section geometry on permeability. Therefore, I focus on the single-phase flow through pipes with various cross-sections including circular pipes, elliptical pipes, triangular pipes, square pipes, n-cusps hypotrochoidal pipes, and sinusoidal pipes. The sinusoidal pipes differ from the other pipes in having cross-section that varies sinusoidally along the flow direction. In this chapter, I also introduce the Apparent Radius, which can be used in place of the hydraulic radius in the Kozeny-Carman equation. By mathematical derivation, I show that this form is valid for pipes other than a circular pipe as well.

Chapter 5 Tortuosity

Chapter 5 presents a detailed study of hydraulic tortuosity. Hydraulic tortuosity is one of the most important parameters in characterizing fluid flow heterogeneity in porous media. The most basic definition of tortuosity is the ratio of average flow path length to sample length. Although this definition seems straightforward, the lack of understanding and of proper ways to measure tortuosity make tortuosity one of the most abused parameters in rock physics. Often, the tortuosity is obtained from laboratory measurements of porosity, permeability, and specific surface area by inverting the KC equation. This approach has a major pitfall, as it treats tortuosity as a fitting factor, and the inverted tortuosity is often un-physically high. In contrast, I obtained the tortuosity from 3-D segmented binary images of porous media using streamlines extracted from a local flux, the output from the Lattice Boltzmann flow simulation. After obtaining streamlines from each sample, I calculated the distribution of tortuosities and flux-weighted average tortuosity.

Chapter 6 The Revised Kozeny-Carman method

This chapter combines the knowledge of cross-section pore geometry and tortuosity, and I show that the pore size distribution is the missing parameter crucial for accurately predicting permeability in porous media. Based on this insight, I derive the revised Kozeny-Carman equation and show that it significantly improves the permeability estimation compared to the original equation for isotropic clastic rocks.

Chapter 7 Machine Learning in Digital Rock Physics

This chapter presents machine learning methods for predicting physical properties from binary segmented images. Instead of using conventional numerical simulations, I developed machine learning methods and showed that it is possible to predict 3-D transport properties by using geometrical features from both 2-D and 3-D μXCT binary segmented images. Both multilayer neural network (MNN) and convolutional neural network (CNN) algorithms were employed to predict permeability. Training was performed through both feed-forward and back-propagation with Bayesian Regularization by using a gradient descent algorithm. The inputs for MNN can be geometrical parameters such as Minkowski Functionals (porosity, specific surface area, integral of mean curvature (for 3-D), and Euler number). For CNN, the inputs can be either 2-D or 3-D binary images.

Chapter 8 Workflow for Grain Size Distribution

The aim of this chapter is to develop a robust code to digitally measure grain size distribution on a 2-D or 3-D image. I also aim to establish the workflow to estimate the grain size distribution from 2-D thin sections through Wicksell's corpuscle modeling on the μXCT images. Furthermore, I improved the precision of the method by incorporating principal component analysis to find the eigenvector of grain orientation. This method enables us to extract more information from the digital image about the grain size distribution such as the grain volume, grain surface area, grain principal axis inclination and azimuth, and the number of contacts of each grain.

Appendix A: Digital Microstructures

This section describes digital microstructures used in this dissertation including (1) pipes with various cross-sections, (2) artificial and physical sphere packs, and (3) natural rocks.

Appendix B: Numerical Simulations

This section describes different numerical simulations used in digital rock physics such as the Lattice Boltzmann flow simulations and finite element method. I will also discuss the effective of discretization in different numerical simulations.

Appendix C: Codes

This section compiles all the codes I developed for this dissertation.

Chapter 2

Tools and Techniques in Digital Rock Physics

This chapter describes various common tools and techniques developed for the Digital Rock Physics study in this dissertation. For specific tools such as streamlines extraction or grain size distribution measurements, the details will be given in the methodology section of their own chapters. For the complete list of the MATLAB codes, please refer to Appendix C. These digital tools can be divided into three main categories: (1) artificial microstructure construction, (2) microstructure alteration, and (3) microstructure geometric measurements.

These tools and techniques are applied on the binary segmented image of a rock. Each voxel in the image belongs to one of two phases: solid or pore space. The solid element has the value 1 and the pore element has the value 0.

2.1 ARTIFICIAL MICROSTRUCTURE CONSTRUCTION

The construction of artificial microstructures is one of the most important tools in Digital Rock Physics as it helps us to understand rock "geometry" better. Throughout this dissertation, various simple artificial microstructures are presented ranging from simple ones such as pipes with various cross-sections to more complex ones such as granular packs. Artificial microstructures serve as a bridge between traditional Rock Physics and Digital Rock Physics since most of the traditional Rock Physics theories are based on simplified rock geometry.

2.1.1 Pipes of various cross-sections

The following step allows us to create pipes with any cross-sections. First, we mesh the grid by specifying the sample size – defined as the number of pixels in the x, y, z directions (nx, ny, nz). Note that in order to create an image of size (nx, ny, nz), arguments to the meshgrid function in MATLAB must be given in the order ny, nx, nz.

```
[x, y, z] = meshgrid(1:ny, 1:nx, 1:nz);
```

Once the grid is created, we modify it by using the equation corresponding to the shape we want to create. To create a cylinder, for example, we need to provide a radius (r) and a center location [ctx, cty, ctz] for the pipe. The code below generates a solid cylindrical pipe.

```
tempImage   = sqrt((x-ctx).^2 + (z-ctz).^2) < r;
```

Then we can invert the solid, creating a pore space, using an absolute function.

```
Cylinder    = abs(1-tempImage);
```

We can create pipes of varying cross-section by replacing the constant r with a variable r. The following codes are available in Appendix C:

1) createCylinder.m

2) createEllipse.m

3) createEqTriangle.m

4) createCrescent.m

5) createRectangle.m

6) createHypotrochoid.m

7) createSinusoidalPipes.m

2.1.2 Artificial and physical sphere packs

The steps for creating a spherical pack are similar as those for creating a pipe. In this case, we need to know the center (x,y,z) and the radius of each sphere. This may require some scaling if the location and size of the spheres are different from those of the output image. MATLAB codes in the Appendix C include

1) createSCP.m

2) createFCP.m

3) createSphericalPack.m

A Finney pack is a physical random close packing of identical spheres (Finney, 1970). The Finney pack used here consists of 4021 spheres; the location of each sphere was digitally rendered in a 3-D Cartesian coordinate system. The Finney pack acts as a bridge between artificial packing models and natural rocks, and it is widely used in computational experiments (Jin et al., 2009; Richa, 2010; Sain, 2011; Dvorkin et al., 2012). The Finney pack can be generated by calling mkfinney2.exe within MATLAB using the following code:

```
nx = 200; ny = 200; nz = 200; LX = 12; x0 = 0; y0 = 0; z0 = 0; R = 1;
```

```
! mkfinney2.exe finneytest.dat nx ny nz LX x0 y0 z0 R
```

where nx, ny, and nz define the size of the 3-D binary image, LX is the field of view (the smaller the number, the larger the field of view; the maximum value accepted is 12), x0, y0, z0 describe the locations of the spheres, and R is the radius of the spheres (R=1 equals the original size of the spheres in the Finney Pack). The code mkfinney.exe is the in house code written in C++. This code also requires the input file Finney.dat, which consists of recorded x,y,z locations for 4021 identical spheres. Figure 2-1 shows different realizations of the Finney pack created using r = 0.5, 1, 1.5 and LX = 3, 6, 9, 12.

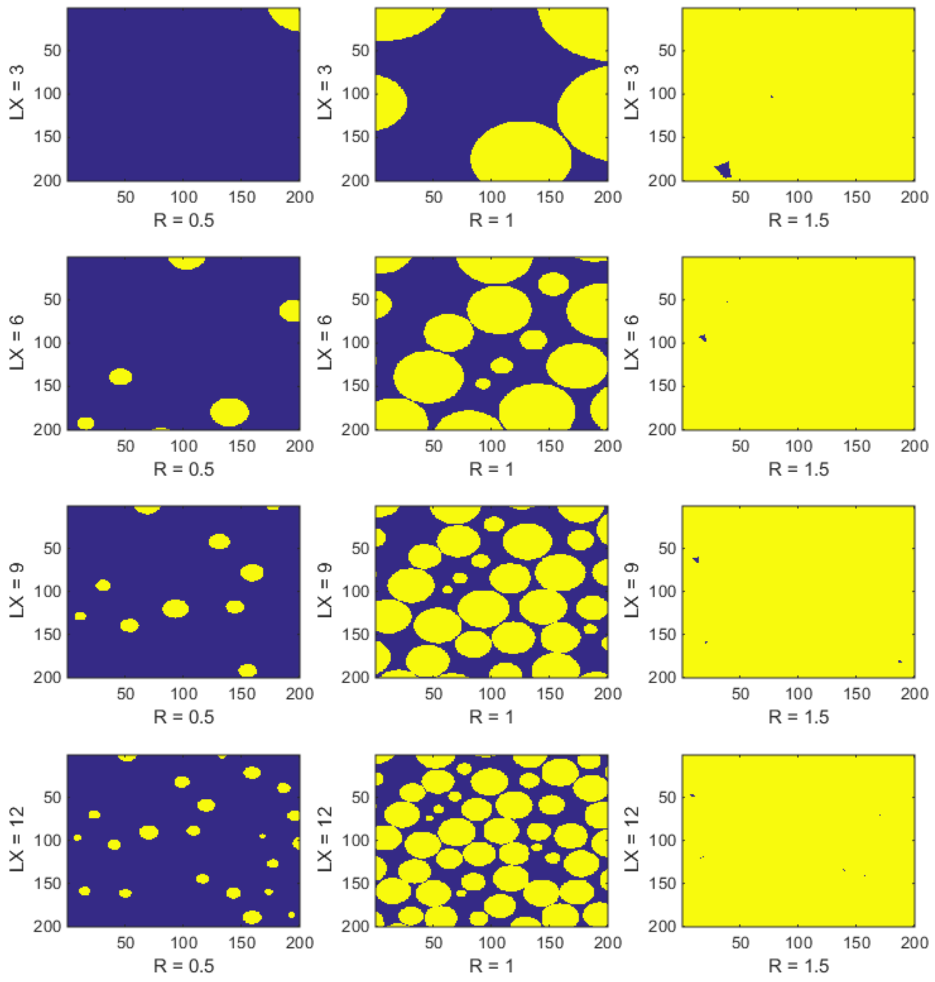


Figure 2-1:Multiple realizations of the Finney pack using r = 0.5, 1, 1.5 and LX = 3, 6, 9, 12.

2.2 MICROSTRUCTURE ALTERATION

The main objective for microstructure alteration is to study how a change in rock geometry would affect the rock's physical properties. Microstructure geometry can be altered in different ways. The simplest way is the dilation and erosion of the grain size.

We can also alter an image so that it contains only the connected pore space, which allows the computation of connected (effective) porosity. To find the connected pore space, we first label each group of connected pores by (1) selecting an unlabeled voxel, (2) labeling it and all the voxels connected to it, and (3) repeating steps 1 and 2 until all voxels in the pore space are labeled (Figure 2-2). To determine effective pore space, we find the labelled groups of connected pores that exist in all of the outermost 2-D slices from every face, which represent the connected pore space. This process is important for finding geometrical parameters used in transport properties because only connected pore space contributes to fluid and electrical flow. The MATLAB function createConnectedPorespace.m takes a digital 3-D binary image as an input.

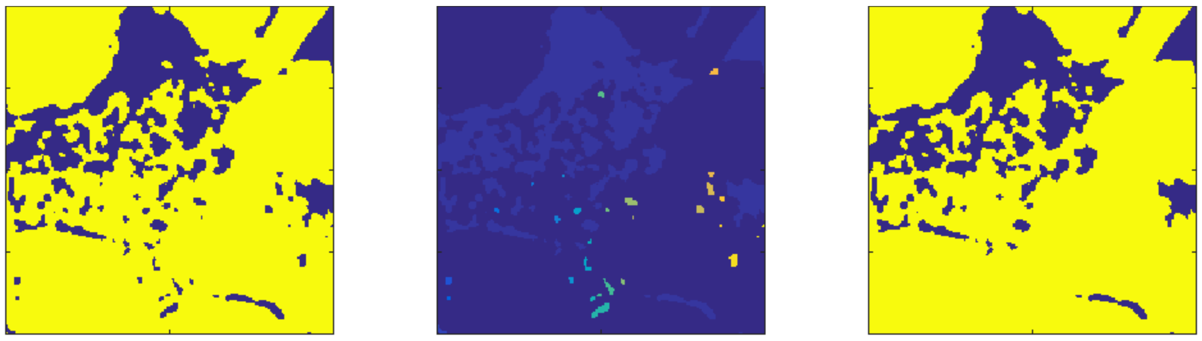


Figure 2-2:(a) Original image of a Berea sandstone, (b) labelled pore space – different colors represent separate components in the pore space, (c) connected pore space.

2.3 GEOMETRIC MEASUREMENTS

This section describes mathematical description of the pore space geometry used in this dissertation including (a) Minkowski Functionals, (b) 2-D/3-D shape extraction through convolution, (c) 2-D/3-D proximity, (d) 2-D/3-D pore size distribution along streamlines, and (e) 2-D/3-D grain size distribution.

2.3.1 Minkowski Functionals

Minkowski Functionals describe the standard geometric measurements for a binary image (Vogel et al., 2010). For a d-dimensional space, there are d+1 associated Minkowski measurements. For example, a 3-D geometry can be defined by 4 Minkowski measurements. For a 3-D model, the first functional M_0 is the total volume of pore space, with dimensions L^3. Porosity can be calculated by dividing M_0 by the total volume of solid and pore space. The second functional M_1 represents the total surface area between solid and pore spaces, with dimensions L^2. We define a specific surface area as M_1 divided by the total volume of solid and pore space. Therefore, a specific surface area has dimensions L^{-1}. The third functional M_2 is the integral of mean curvature (mean breadth) on the surface, with dimensions L. This can be defined as $M_2(\mathrm{X}) = \frac{1}{2}\int_{\delta \mathrm{x}} \left[\frac{1}{\mathrm{r}_1} + \frac{1}{\mathrm{r}_2}\right] \mathrm{ds}$, where r_1 and r_2 are the minimum and maximum radii of curvature on the surface element ds. The last Minkowski Functional, M_3, is the (dimensionless) Euler Characteristic, defined as (number of vertices) – (number of edges) + (number of faces) – (number of distinct objects). The computation of Minkowski Functionals on 2-D and 3-D binary images is based on Legland et al. (2007) MATLAB codes (Legland et al., 2007).

To understand each geometrical parameter in the Minkowski Functionals, we generated idealized shapes such as spheres, cylinders, and octahedron for the study. The geometric measurements on the idealized shapes allow for testing the accuracy of the Minkowski algorithm on 3-D binary images and help understanding whether the boundary of the image has any effect on the geometric measurements.

Table 2-1: The Minkowski measurements of idealized geometric shapes. The 3-D image size of each shape is 100^3 voxels. No. 2,4,6,8 are the inverse structures of No. 1,3,5,7, respectively. Examples of the basic shapes associated with this table are shown in Figure 2-3.

No.	Shape	Porosity (M_0)	Specific Surface Area (M_1)	Integral of Mean Curvature (M_2)	Euler Number (M_3)
1	A Sphere (r = 40 pixels)	0.2677	0.0207	79	1
2*	A cube with spherical hole (No.1 Inverse)	0.7323	0.0207	21	2
3	A Sphere (r = 50.5 pixels)	0.5396	0.0316	100	1
4*	A cube with spherical hole (No.3 Inverse)	0.4604	0.0316	80	-4
5	8 isolated spheres (r = 20 pixels)	0.267	0.0411	312	8
6*	A cube with 8 isolated spherical holes (No.5 Inverse)	0.733	0.0411	-212	9
7	8 isolated spheres (r = 25 pixels)	0.797	0.0485	136	-4
8*	A cube with 8 isolated spherical holes (No.7 Inverse)	0.203	0.0485	508	-27
9	Cylinder	0.1245	0.0107	59.3333	1
10	Three cylinders	0.2843	0.0208	100	1
11	Octahedron	0.0196	0.005	49	1
12	8 Octahedrons	0.0794	0.0251	312	8

The integral of mean curvature for a cube of size 100^3 is 100. When swapping grains with pores, the porosity becomes 1-porosity while the specific surface area remains the same. The integral of mean curvature changes sign from + to – but the algorithm also considers the integral of mean curvature for the outer surface of the cube. Therefore, the integral of mean curvature for a cube of size 100^3 voxels is 100. If the original structure has the integral of mean curvature for a cube of size X, then the inverse structure has the integral of mean curvature for a cube of size 100 – X (Table 2-1).

As the Euler number is a direct count of *vertices – edges + faces – solid*, it is totaled when there is more than one isolated object within the mesh grid. For example, if a sphere has an Euler number of 1, a cube with 8 spheres has an Euler number of 8. However, the Euler number becomes negative when the spheres start contacting each other. According to observation, the higher the negative number, the higher the number of holes in the pore space.

We change the radius of the sphere to examine the effect of grain size on Minkowski Functionals (Table 2-2). The mesh grid remains the same size, so when the radius = 50 (diameter = 100) the sphere touches the boundary of a mesh cube that has the size of 100^3 voxels. This is also the point with maximum specific surface area. The change in size of the grain does not affect the Euler number: it remains 1 as the radius increases. The algorithm approximates the integral of mean curvature to be very close to the theoretical value of 2r for the sphere.

Table 2-2: Minkowski Functionals on a sphere with the change in radius.

Radius of the Sphere	Porosity (M_0)	Specific Surface Area (M_1)	Integral of Mean Curvature (M_2)	Euler Number (M_3)
10	0.0041	0.0013	19	1
20	0.0334	0.0051	39	1
30	0.1129	0.0116	59	1
40	0.2677	0.0207	79	1
50	0.5232	0.0316	99	1
60	0.7974	0.0243	100	1
70	0.9589	0.0096	100	1
80	0.9979	0.0011	100	1
90	1	0	100	1
100	1	0	100	1

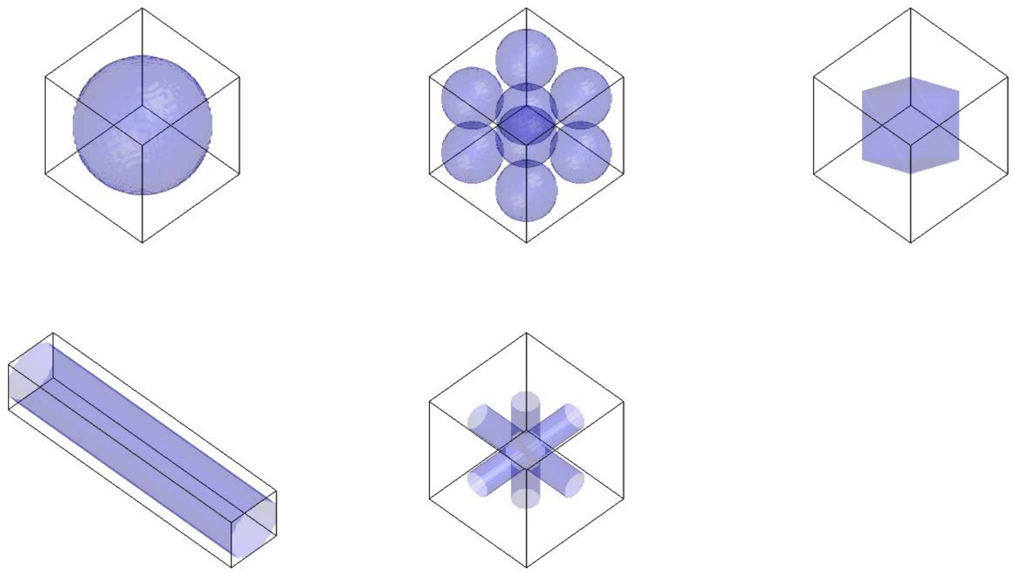


Figure 2-3: Idealized geometric shapes, from the left, sphere, simple cubic sphere pack, octahedron; cylinder, 3-axis cylinder. The properties of these shapes are summarized in Table 2-1.

We validated two of the Minkowski Functionals (porosity and specific surface area) by creating multiple 3-D straight pipes in a solid frame with circular, elliptical, square, and triangular cross-sections with porosities ranging from 5% to 40%. Since the porosity and specific surface are known exactly for these shapes, we can test the effect of discretization in Minkowski Functionals codes. Figure 2-4 shows that a Minkowski Functionals code can accurately calculate porosity for straight pipes with various cross-sections, even when porosity is very low. The largest difference between exact and calculated specific surface area (SSA) is for the square pipe. The codes employ 26-adjacency for polyhedral reconstructions of a structure (Legland et al., 2007), and it may be difficult to accurately reconstruct a square pipe in the form of polyhedrons having 26-adjacency. However, the SSA of a square pipe may be irrelevant for calculations involving realistic pore geometries.

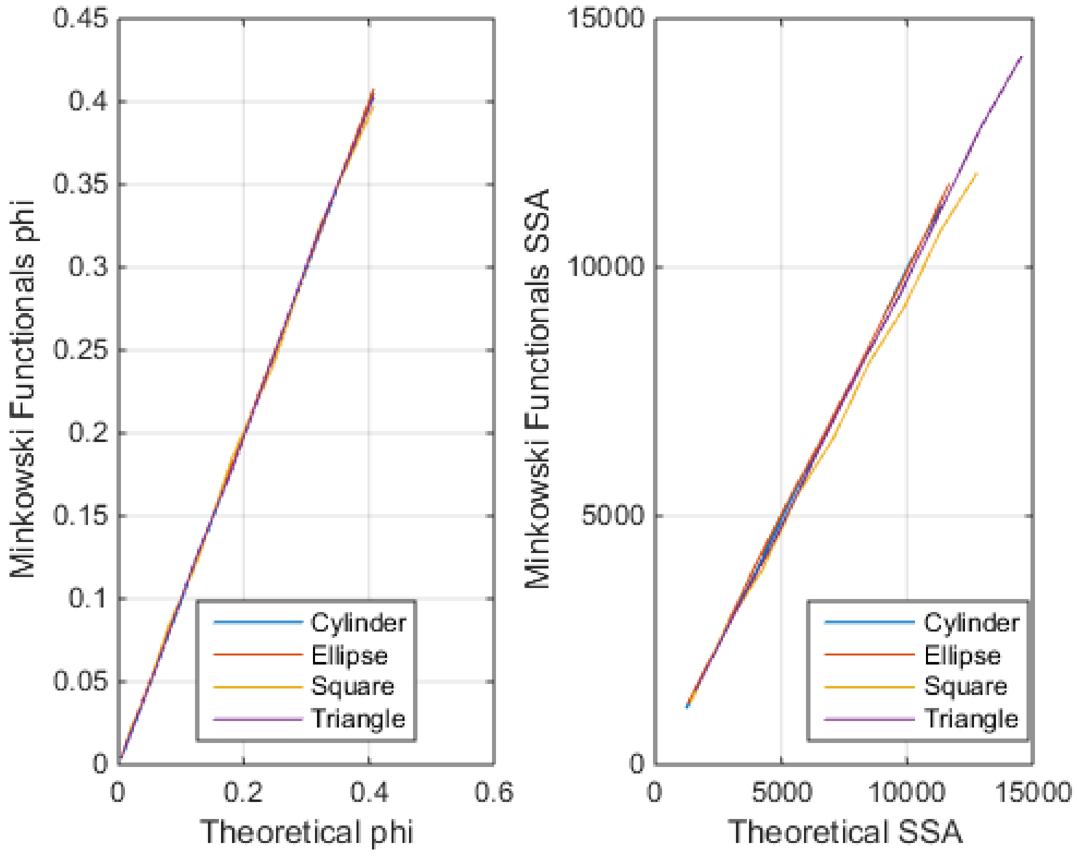


Figure 2-4: Comparison of Minkowski geometrical measurements (porosity -phi, specific surface area – SSA) to the theoretical values

The MATLAB code is available in the paper "Computation of Minkowski measures on 2D and 3D binary images," (Legland et al., 2007). We also provide a MATLAB wrapper code that accommodates a cell array input of 3-D images in Appendix C called computeMinkowski3D.m. The input can be either a 3-D matrix of a single 3-D porous image or a cell array containing 3-D matrices of multiple 3-D porous images. The program will run multiple simulations if a cell array containing 3-D matrices is an input.

2.3.2 2-D/3-D shape extraction through convolution

We can extract the location and the number of any specific shape in the 2-D/3-D binary image using convolution. For example, Figure 2-5 shows how to find the location of a cross shape within a sample image. After convolving, any pixel in the final image that has value equal to the number of pixels in the target shape indicates the location of the pattern found in the image. The MATLAB commands for this task are

```
[outputImage2D] = conv2(image2D, targetShape2D, 'same')
[outputImage3D] = convn(image3D, targetShape3D, 'same')
```

where image2D is the 2-D binary segmented image of size (nx*ny) and image3D is the 3-D binary segmented image of size (nx*ny*nz). The syntax 'same' returns the central part of the convolution that is the same size as image2D/image3D.

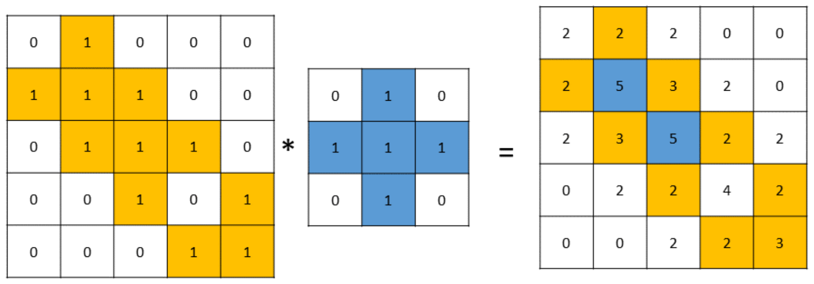


Figure 2-5: The convolution of a sample image and a pattern. After convolving, any pixel with value 5 (the number of pixels in the pattern) indicates the location of the pattern found in the image.

2.3.3 2-D/3-D proximity

Proximity is the Euclidean distance from each 0 pixel/voxel (pore) to the nearest 1 pixel/voxel (solid) (Figure 2-6). The size of the 2-D/3-D proximity output matrix is the same as the size of an input image.

For 2-D proximity, we slice a 3-D binary image into 2-D images perpendicular to the Z-direction. For each pore pixel, we find the Euclidean distance from that pixel to the nearest solid pixel in the plane. 2-D proximity is analogous to the radius of the largest inscribed circle centered at the pixel that will fit within the pore space. The 2-D Euclidean distance (D) between pixel (x_1, y_1) and pixel (x_2, y_2) is

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}.$$

In 3-D proximity, we find the Euclidean distance between each pore voxel and the nearest solid voxel. The Euclidean distance (D) between voxel (x_1, y_1, z_1) and voxel (x_2, y_2, z_2) is

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}.$$

According to this definition, the 3-D proximity for each solid voxel equals zero. Finding the 3-D proximity of each voxel is analogous to fitting the largest possible inscribed sphere centered at the voxel within the open pore space. The output matrix of 2-D/3-D proximity can then be used to further analyze pore throats along streamlines.

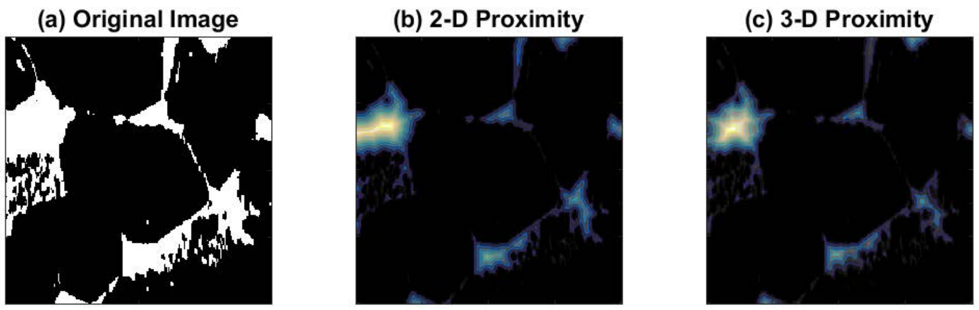


Figure 2-6: (a) original image of Berea sandstone, (b) contoured 2-D proximity, (c) contoured 3-D proximity.

The MATLAB function that calculates 2-D/3-D proximity is compute2D3DProximity.m. This code takes a 3-D image as an input.

2.3.4 2-D/3-D pore size distribution along streamlines

2-D/3-D Proximity can be combined with streamlines to extract 2-D/3-D pore size distribution along the streamlines. This process is achieved by using the xyz coordinates of streamlines to extract 2-D/3-D proximity along the streamlines. The 2-D/3-D proximity is equivalent to fitting a maximum inscribed circle/sphere, and hence some of the proximity along the streamlines can represent the pore throat.

The MATLAB function that calculates 2-D/3-D pore throat distribution along streamlines is compute2D3DProximity.m. This code takes a 3-D images as an input. The image3D input can be either a 3-D matrix of a single 3-D porous image or a cell array containing 3-D matrices of multiple 3-D porous images. The program will run multiple simulations if a cell array containing 3-D matrices is the input.

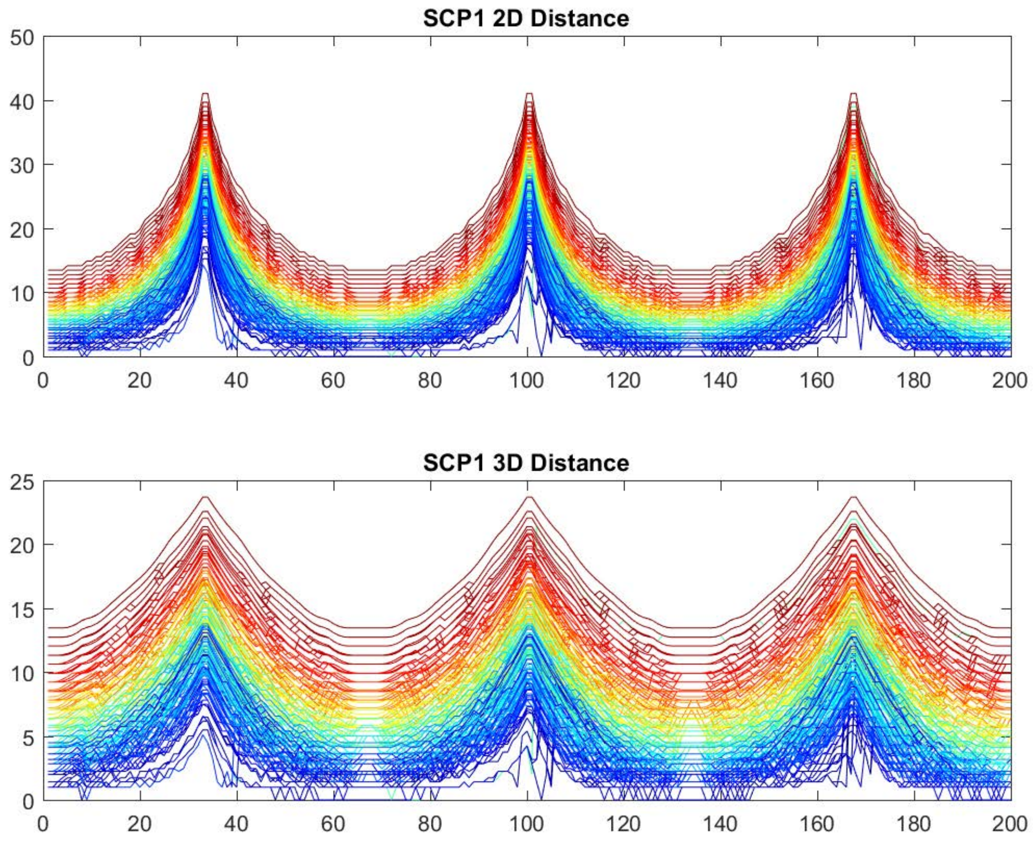


Figure 2-7: (Top) 2-D proximity along the streamlines in simple cubic pack (SCP) and (Bottom) 3-D proximity along the streamlines in simple cubic pack (SCP).

2.4 CONCLUSION

With digital 2D and 3-D images it is now possible to investigate rock geometry thoroughly. We can now measure geometric parameters that cannot be measured directly in the laboratory. The knowledge we gained from these algorithms will be discussed in the following chapter.

Chapter 3

Review of the Kozeny-Carman Equation

This chapter presents the literature review and the derivation of the Kozeny-Carman Equation. The Kozeny-Carman equation is a semi-empirical model relating permeability in single-phase flow to geometric measurements such as porosity, specific surface area, tortuosity, and a geometric factor. Through the Kozeny-Carman equation, we can better understand how the rock geometry relates to permeability, but using the equation can create problems, which are addressed below.

3.1 INTRODUCTION

The Kozeny-Carman (KC) equation is a well-known semi-empirical equation for permeability prediction. It relates single-phase flow permeability (κ) to other rock geometric properties such as porosity (ϕ), specific surface area (S), hydraulic tortuosity (τ), and a geometric factor (B). The original Kozeny-Carman equation was derived in 1956, but the equation is still being used nowadays in the literatures (Carrier, 2003; Gomez et al., 2010; Ozgumus et al., 2014).

The Kozeny-Carman equation is derived from the exact solution for laminar flow in a straight pipe with circular cross-section, the Hagen-Poiseuille equation, and Darcy's law. Based on the Hagen-Poiseuille equation, the exact solution of volumetric flow rate (q) in a round pipe of radius R, length l, pressure loss ΔP, and viscosity η is

$$q = -\frac{\pi R^4}{8\eta}\frac{\Delta P}{l}. \qquad (1)$$

The volumetric flow rate can also be calculated using Darcy's law, which describes volumetric flow rate (q) through a porous medium with permeability κ, cross-sectional area A, and length L:

$$q = -\kappa\frac{A}{\eta}\frac{\Delta P}{L} \quad . \qquad (2)$$

By equating the Hagen-Poiseuille equation (Equation 1) and Darcy's Law (Equation 2), we can derive the original Kozeny-Carman equation as

$$-\frac{\pi R^4}{8\eta}\frac{\Delta P}{l} = -\kappa\frac{A}{\eta}\frac{\Delta P}{L} \quad , \qquad (3)$$

$$\kappa = \frac{\pi R^4}{8A\tau} \quad , \qquad (4)$$

where tortuosity τ equals the ratio of a length of a round pipe l to the length of a permeable frame L. The Kozeny-Carman equation can be expressed in terms of porosity (ϕ) and specific surface area (*S*) by relating these two variables to radius R based on the assumption of a round pipe:

$$\phi = \frac{\pi R^2 l}{AL} = \frac{\pi R^2 \tau}{A} \quad , \qquad (5)$$

$$S = \frac{2\pi R l}{AL} = \frac{2\pi R \tau}{A} = \frac{2}{R}\frac{\pi R^2 \tau}{A} = \frac{2\phi}{R} \quad . \qquad (6)$$

By plugging Equations 5 and 6 into Equation 4, the Kozeny-Carman equation can be expressed in a more familiar form:

$$\kappa = \frac{1}{2}\frac{\phi^3}{S^2\tau^2} \quad or \quad \kappa = B\frac{\phi^3}{S^2\tau^2} \quad , \qquad (7)$$

where *B* is a geometric factor. For the estimation of permeability in a granular medium consisting of identical spheres with diameter *d*, we can express the specific surface area as

$$S = \frac{6(1-\phi)}{d} \quad . \qquad (8)$$

Plugging S into Equation 7, we get

$$\kappa = \frac{1}{72}\frac{\phi^3}{(1-\phi)^2\tau^2}d^2 \quad . \qquad (9)$$

3.2 THE KOZENY-CARMAN EQUATION IN DIGITAL ROCK PHYSICS

In the Kozeny-Carman equation, the porosity and specific surface area can either be measured directly from a laboratory experiment or calculated from a 3-D segmented binary image of the porous medium under investigation. Previously, the specific surface area was difficult to measure and very sensitive to the scale of the images (Keehm et al., 2001), but now the codes from Minkowski functionals discussed in Chapter 2 can yield more accurate specific surface areas (Legland et al., 2007). The tortuosity has often been treated merely as a fitting factor due to a lack of understanding of the parameter. From a recent study using a database of ordered sphere packs, random sphere packs, and natural rocks, we computed the flux-weighted tortuosity directly from streamlines extracted from a local flux velocity, which is the output from the Lattice Boltzmann (LB) flow simulation (Srisutthiyakorn and Mavko, 2017). The results show that the hydraulic tortuosities are mostly in the small range of 1.2-1.6, unlike the typical value used in the literature of 1.5 to 2.5 (Gomez et al., 2010). Another commonly used fitting factor is the geometric factor, B, which describes the effect of pipe cross-sectional shape, and typically also has a small range. For example, B is 0.5 in a round pipe, 0.562 in a square pipe, and 0.6 in an equilateral triangle pipe (Mavko et al., 2009a).

With known parameters and without a fitting factor, the Kozeny-Carman equation predicts permeability higher by one to two orders of magnitude than that predicted by the LB flow simulation. In a recent paper (Srisutthiyakorn and Mavko, 2017), we also searched for a missing parameter by exploring the computation of porosity and specific surface area using only connected pore space. We found that the connected pore space does not contribute to the large difference between the original Kozeny-Carman permeability and

LB permeability. Therefore, in order to use the original Kozeny-Carman equation effectively, one or two fitting parameters (i.e. tortuosity, geometric factor, and the exponent of porosity) are required for most data sets. For example, Bourbié et al. (1987) suggested making the exponent of porosity a variable. Mavko and Nur (1997) recommended modifying the porosity by subtracting the percolation threshold porosity from the total porosity, and using n values, from the derived value of 3 to values of 7-8 at very low porosity.

Chapter 4

Cross-Section Geometry

This chapter discusses the effect of cross-section pore geometry on permeability. I focus on single-phase flow through pipes with various cross-sections including circular pipes, elliptical pipes, triangular pipes, square pipes, n-cusps hypotrochoidal pipes, and sinusoidal pipes. The sinusoidal pipes differ from the other pipes in having circular cross-section with diameter that varies sinusoidally along the pipe axis. I also introduce the *Apparent Radius*, which can be used in place of the hydraulic radius in the Kozeny-Carman equation to better capture the pipe shape. I will be discussing the flow in complicated rock geometries in Chapter 6: The Revised Kozeny-Carman Equation.

4.1 INTRODUCTION

This chapter discusses single-phase flow through pipes with various cross-sections and shows that for the Kozeny-Carman form of $\frac{\phi^3}{S^2\tau^2}$, porosity (ϕ) and specific surface area (S) are not uniquely related to permeability. For pipes with various cross-sections, ϕ and S do not determine the radius of a pipe or the size of pore throats, which is critical for permeability. This fact can be illustrated by a pipe with a thin throat in the middle. While the porosity and the specific surface area of such a pipe are very similar to those of a pipe without a throat, the permeability of a pipe with a throat can be orders of magnitude smaller.

For previous works that studied the flow on pipe models, Mason and Morrow (1991) explained the capillary behavior of a perfectly wetting liquid in an irregular triangular cross-section pipe since he claimed it offers greater versatility than a circular pipe in studying capillary behavior in multiphase flow. Patzek and Kristensen (2001) explained the conduit geometry using a Mason-Morrow shape and measured the shape factor from corner half angle. Yet, using this method, it is not possible to determine the shape factor from a convex cross-section shape where the angles at the corners are not constant. White (1974) related the hydraulic radius approximation (the ratio of perimeter to area of 2-D cross-section of a pipe) to hydraulic conductivity. Sisavath et al. (2000) made a detailed study of the effect of laminar flow on irregular cross-sectional shape.

In this chapter, we focus on the original form of the Kozeny-Carman equation:

$$\kappa = \frac{\pi R^4}{8A\tau} \quad ,$$

where τ is tortuosity, which is the length of flow path (l) divided by the length of the permeable frame (L) . Because the original equation always assumes circular pipe geometry and uses hydraulic radius directly in R^4, it is guaranteed to underestimate the permeability. We solved this shortcoming by applying the Kozeny-Carman equation in the form of R^4 to pipes of any cross-sections, and replacing R, the radius of the pipe, with the "*apparent radius*" R_A (the geometric mean between the hydraulic radius R_H and the radius of a circular pipe R_{Circ} that has the same porosity as the pipe under consideration). This improved version of Kozeny-Carman equation will require less calibration when it is used to fit data.

Although results obtained using this method still need to be corrected using the geometric factor B, which is determined from the flux in the pipe, they give us a better estimation of permeability than the original Kozeny-Carman equation does. Furthermore, the geometric factor B has a minimal effect on permeability prediction; its value is typically assumed to be 0.5 for all complex geometries.

The Kozeny-Carman equation for non-circular pipes is derived in the same way, by relating specific surface area to porosity (Dvorkin, 2009). Table 4-1 summarizes analytical solutions for permeability for pipes of various cross-sections.

Table 4-1: Analytical solutions for permeability (κ) for pipes with various cross-sectional geometries. The parameter n is the number of pipes, l is the length of a pipe, ϕ is porosity, S is specific surface area, τ is tortuosity, R is the radius of a circular pipe, a is semi-major axis and b is semi-minor axis for an ellipse, t is the side of an equilateral triangle, and s is the side of a square. A is the cross sectional area and L is the length of permeable frame.

Cross-section	porosity (ϕ)	specific surface area (S)	Kozeny-Carman equation	B (Geometric Factor)
Circular	$\phi = n \cdot \frac{\pi R^2 l}{AL}$	$S = n \cdot \frac{2\pi R l}{AL}$	$\kappa = n \cdot \frac{1}{2}\frac{\phi^3}{S^2\tau^2}$	0.5
Elliptical	$\phi = n \cdot \frac{\pi a b l}{AL}$	$S = n \cdot \frac{\pi\sqrt{2(a^2+b^2)}l}{AL}$	$\kappa = n \cdot \frac{1}{2}\frac{\phi^3}{S^2\tau^2}$	0.5
Square	$\phi = n \cdot \frac{s^2 l}{AL}$	$S = n \cdot \frac{4sl}{AL}$	$\kappa = n \cdot 0.562\frac{\phi^3}{S^2\tau^2}$	0.562
Equilateral Triangular	$\phi = n \cdot \frac{t^2\frac{\sqrt{3}}{4}l}{AL}$	$S = n \cdot \frac{3tl}{AL}$	$\kappa = n \cdot 0.6\frac{\phi^3}{S^2\tau^2}$	0.6

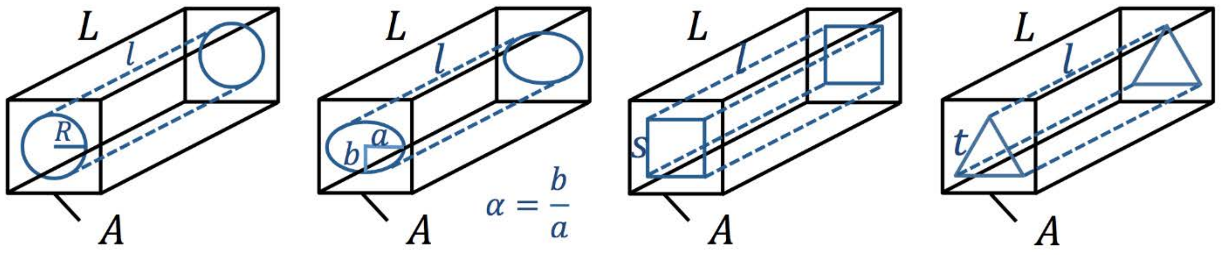


Figure 4-1: 3-D digital representations of a circular pipe, an elliptical pipe with aspect ratio 0.7, a square pipe, and an equilateral triangular pipe.

4.2 APPARENT RADIUS

For pipes with various cross-sections, we propose to use the original form of the Kozeny-Carman equation $\kappa = \frac{\pi R^4}{8A\tau}$ to estimate the effective permeability. The only change in the equation is that, R, the radius of a circular pipe, is replaced with R_A or *apparent radius* (defined below). The Kozeny-Carman can then be expressed as

$$\kappa = \frac{B}{B_{Circ}} \frac{\pi {R_A}^4}{8A\tau} = \frac{\pi {R_A}^4}{8A\tau} \ (assuming\ B = B_{Circ} = 0.5) \quad , \qquad (1)$$

where A is the cross-sectional area of the permeable frame and τ is tortuosity.

We define the apparent radius for any cross-section pipe as the geometric mean between hydraulic radius R_H and the radius of a circular pipe R_{Circ} that has the same porosity as the pipe under consideration.

$$R_A = \sigma \cdot R_{Circ} = \left(\sqrt{\frac{R_H}{R_{Circ}}} \right) R_{Circ} = \sqrt{R_H \cdot R_{Circ}} \quad . \quad (2)$$

The hydraulic radius R_H (Tchelepi, 2015) is defined as two times the porosity (ϕ) divided by the specific surface area (S):

$$R_H = \frac{2\phi}{S} \quad . \quad (3)$$

In equation 2, σ can be defined as a *shape constant* that is always less than or equal to 1. Using the *shape constant σ,* we can describe the permeability reduction resulting from changing a circular cross-section to other cross-sections. The only case where σ is equal to 1 is $R_H = R_{Circ}$, which is valid only in a circular pipe since, in that case, the hydraulic radius equals the radius of the pipe itself.

$$R_H = 2\frac{\pi R^2 \cdot l/AL}{2\pi R \cdot l/AL} = R, \qquad \sigma = \sqrt{\frac{R_H}{R_{Circ}}} = 1 \quad . \quad (4)$$

R_A for a circular pipe is the largest radius compared to the radii of all non-circular pipes with the same porosity. This can be proven using isoperimetric quotient theory. For any closed shape, the isoperimetric quotient Q relates area A and perimeter p of a closed area in a plane (Osserman, 1987) as follows:

$$Q = \frac{4\pi A}{p^2} \leq 1 \quad . \quad (5)$$

The isoperimetric quotient is 1 only if the closed shape is a circle, indicating that of all closed shapes with the same area, the circle has the shortest perimeter. In other words, a circular pipe has the least specific surface area and thus the highest porosity to specific surface area ratio. Since we are treating permeability as a function of ${R_A}^4$, for any cross-section pipe with the same porosity, a circular pipe will have the highest permeability. (Figure 4-2).

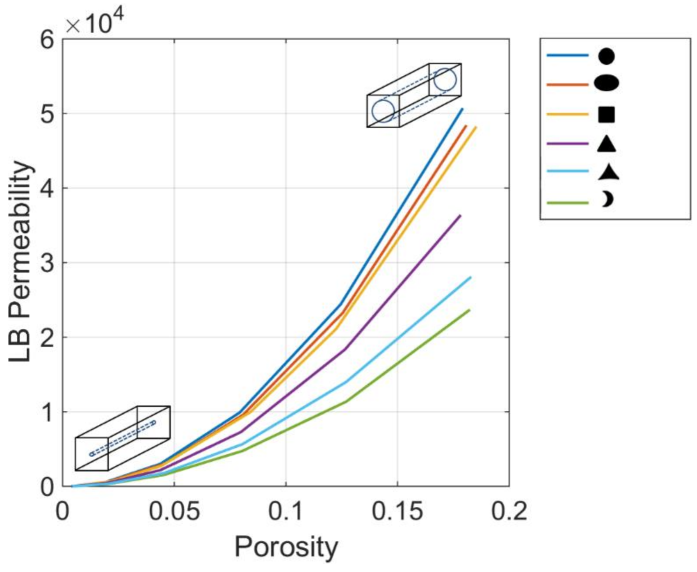


Figure 4-2: For all pipes with the same porosity, a circular pipe has the highest permeability. The ratio of the permeability of any cross-section to that of a circular cross-section can be found using the apparent radius concept. In this case, the permeability is calculated using Lattice-Boltzman flow simulation.

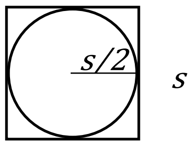


Figure 4-3: Cross-section of a square pipe of side s. The hydraulic radius is s/2 which is exactly the radius of a circle inscribed in the square.

Using hydraulic radius alone as the radius of a circular pipe in Equation (1) would underestimate the permeability even in a simple cross-section pipe. Take a square cross-section pipe, for example, which has the hydraulic radius:

$$R_H = 2\frac{s^2 \cdot l/AL}{4s \cdot l/AL} = s/2 \quad . \quad (6)$$

If we use the hydraulic radius of a square as the radius of a circular pipe to find the effective permeability, the permeability of a circular pipe (K_{circRH}) of radius s/2 will be less than the permeability of a square pipe (K_{sq}) of side s. This is because of porosity reduction by neglecting the flow in the corners of the square Figure 4-3. The proof is as follows:

The permeability of the square pipe is

$$\kappa_{sq} = 0.562\frac{\phi^3}{S^2\tau^2} = 0.562\frac{\left(s^2 \cdot \frac{l}{AL}\right)^3}{\left(4s \cdot \frac{l}{AL}\right)^2 \tau^2} = \frac{0.562}{16} \cdot s^4 \cdot \frac{l}{AL}, \qquad \tau^2 = 1. \tag{7}$$

The permeability of a circular pipe with hydraulic radius $R_H = \frac{s}{2}$ is

$$\kappa_{circRH} = 0.5\frac{\phi^3}{S^2\tau^2} = 0.5\frac{(\pi r^2 \cdot \frac{l}{AL})^3}{(2\pi r \cdot \frac{l}{AL})^2\tau^2} = 0.5\frac{(\pi(s/2)^2 \cdot \frac{l}{AL})^3}{(2\pi(s/2) \cdot \frac{l}{AL})^2\tau^2}$$
$$= \frac{0.5}{16} \cdot \frac{\pi}{4} \cdot s^4 \cdot \frac{l}{AL}, \qquad \tau^2 = 1. \tag{8}$$

Therefore, $\kappa_{circRH} = \frac{0.5}{0.562} \cdot \frac{\pi}{4}\kappa_{sq} = 0.6987 \cdot \kappa_{sq}$. As we have seen, finding the permeability using hydraulic radius instead of the apparent radius yields a prediction approximately 30% less than the theoretical value.

To demonstrate the utility of the concept of apparent radius, we compared analytical solutions developed by Dvorkin (2008) for flow in pipes with the analytical solution derived using the apparent radius. The ratio of the permeability of any pipe to the permeability of a circular pipe that has the same porosity is constant, based on how surface area changes compared to porosity. The permeability also depends on the geometric factor B in a form of the Kozeny-Carman equation that expresses the permeability as $\kappa = B\frac{\phi^3}{S^2\tau^2}$. The coefficient B is derived by equating the flux equation to Darcy's equation. For example, B is 0.5 in circular and elliptical pipes, 0.562 in a square pipe, and 0.6 in an equilateral triangular pipe. For natural rocks where B is unknown, it is typical to assume that B is equal to 0.5. We will demonstrate the utility of the apparent radius by looking at three specific examples – elliptical, square, and equilateral triangular pipes – in more detail.

4.2.1 An Elliptical Pipe

For a circular pipe of radius r and an elliptical pipe with semi axes a and b of the same cross-sectional area, the equation describing the porosity of both shapes is

$$\pi r^2 = \pi ab = \pi a^2\alpha \quad , \quad (9)$$

$$a = \frac{r}{\sqrt{\alpha}} \quad . \quad (10)$$

The aspect ratio (α) is the ratio of the semi-minor axis to the semi-major axis ($\alpha = \frac{b}{a}$). The specific surface area (S) of an elliptical pipe can be expressed in terms of the radius of a circular pipe of length l within a frame of cross-sectional area A and length L,

$$S = \frac{\pi\sqrt{2(a^2+b^2)}l}{AL} = \frac{\pi a\sqrt{2(1+\alpha^2)}l}{AL} = \frac{r}{\sqrt{\alpha}}\cdot\frac{\pi\sqrt{2(1+\alpha^2)}l}{AL} \quad (11)$$

$$= \frac{1}{\sqrt{2}}\left(\sqrt{\alpha+\frac{1}{\alpha}}\right)\cdot\frac{2\pi rl}{AL} \quad .$$

Given a constant porosity, the ratio of the permeability of an elliptical pipe to that of a circular pipe will be constant and is a function of the aspect ratio:

$$\frac{K_{ell}}{K_{circ}} = \frac{0.5\frac{\phi^3}{S^2\tau^2}}{0.5\frac{\phi^3}{S^2\tau^2}} = \frac{0.5\frac{1}{\left(\frac{1}{\sqrt{2}}\left(\sqrt{\alpha+\frac{1}{\alpha}}\right)\cdot\frac{2\pi rl}{AL}\right)^2}}{0.5\frac{1}{\left(\frac{2\pi rl}{AL}\right)^2}} = \frac{2}{\left(\alpha+\frac{1}{\alpha}\right)} \quad . \tag{12}$$

We can also use the apparent radius to find this ratio. For a circular pipe with the same porosity as an elliptical pipe,

$$\pi {R_{Circ}}^2 = \pi ab = \pi a^2\alpha = \frac{\pi b^2}{\alpha} \quad . \tag{13}$$

$$R_H = \frac{2\phi}{S} = \frac{2\left(\pi ab\cdot\frac{l}{AL}\right)}{\frac{\pi\sqrt{2(a^2+b^2)}l}{AL}} = \frac{2\left(\pi ab\cdot\frac{l}{AL}\right)}{\frac{\pi a\sqrt{2(1+\alpha^2)}l}{AL}} = b\sqrt{\frac{2}{(1+\alpha^2)}} \quad , \tag{14}$$

$$R_A = \sigma\cdot R_{Circ} = \sqrt{\frac{R_H}{R_{Circ}}}R_{Circ} = \sqrt{\frac{b\sqrt{\frac{2}{(1+\alpha^2)}}}{\frac{b}{\sqrt{\alpha}}}}R_{Circ} = \sqrt{\sqrt{\frac{2}{\left(\alpha+\frac{1}{\alpha}\right)}}}R_{Circ} \quad . \tag{15}$$

Since the effective permeability is proportional to R^4, and the geometric constant B of both an ellipse and a circular pipe is 0.5,

$$\frac{K_{ell}}{K_{circ}} = \frac{B_{ell}}{B_{circ}}\cdot\frac{2}{\left(\alpha+\frac{1}{\alpha}\right)} = \frac{2}{\left(\alpha+\frac{1}{\alpha}\right)} \quad . \tag{16}$$

Hence using the apparent radius yields the correct ratio.

4.2.2 A Square Pipe

For a circular pipe of radius r and a square pipe with side s with the same cross-sectional area, the equation describing the porosity of both shapes is,

$$\pi r^2 = s^2 \quad . \quad (17)$$

The specific surface area of a square pipe can be expressed in terms of the radius of a circular pipe of length l within a frame of cross-sectional area A and length L:

$$S = \frac{4sl}{AL} = \frac{4(\sqrt{\pi r^2})l}{AL} = \frac{2}{\sqrt{\pi}} \cdot \frac{2\pi rl}{AL} \quad . \quad (18)$$

The ratio between the permeability of a square pipe and that of a circular pipe is

$$\frac{K_{sq}}{K_{circ}} = \frac{0.562\frac{\phi^3}{S^2\tau^2}}{0.5\frac{\phi^3}{S^2\tau^2}} = \frac{0.562\frac{1}{\left(\frac{2}{\sqrt{\pi}} \cdot \frac{2\pi rl}{AL}\right)^2}}{0.5\frac{1}{\left(\frac{2\pi rl}{AL}\right)^2}} = \frac{0.562}{0.5} \cdot \frac{\pi}{4} \quad . \quad (19)$$

Alternatively, using the apparent radius concept,

$$\pi {R_{Circ}}^2 = s^2 \quad , \quad (20)$$

$$R_H = \frac{2\phi}{S} = \frac{2\left(s^2 \cdot \frac{l}{AL}\right)}{\frac{4sl}{AL}} = \frac{s}{2} = \frac{R_{Circ}\sqrt{\pi}}{2} \quad , \quad (21)$$

$$R_A = \sigma \cdot R_{Circ} = \sqrt{\frac{R_H}{R_{Circ}}} R_{Circ} = \sqrt{\frac{\sqrt{\pi}}{2}} R_{Circ} \quad . \quad (22)$$

Since effective permeability is proportional to R^4, and the geometric constant B for a square pipe is 0.562 and the geometric constant of a circular pipe is 0.5,

$$\frac{K_{sq}}{K_{circ}} = \frac{B_{sq}}{B_{circ}} \cdot \left(\sqrt{\frac{\sqrt{\pi}}{2}}\right)^4 = \frac{0.562}{0.5} \cdot \frac{\pi}{4} \quad . \qquad (23)$$

4.2.3 An Equilateral Triangular Pipe

For a circular pipe of radius r and an equilateral triangular pipe with side t of the same cross-sectional area, the equation describing the porosity of both shapes is

$$\pi r^2 = t^2 \frac{\sqrt{3}}{4} \quad . \qquad (24)$$

Hence,

$$t = \sqrt{\frac{4\pi r^2}{\sqrt{3}}} \quad . \qquad (25)$$

The specific surface area of an equilateral triangular pipe can then be expressed in terms of radius of a circular pipe of length l within a frame of cross-sectional area A and length L:

$$S = \frac{3tl}{AL} = \frac{3l}{AL}\sqrt{\frac{4\pi r^2}{\sqrt{3}}} = \left(\frac{3}{\sqrt{\pi\sqrt{3}}}\right)\frac{2\pi rl}{AL} \quad . \qquad (26)$$

The ratio between the permeability of a square pipe and that of a circular pipe is

$$\frac{K_{tri}}{K_{circ}} = \frac{0.6\frac{\phi^3}{S^2\tau^2}}{0.5\frac{\phi^3}{S^2\tau^2}} = \frac{0.6\frac{1}{\left(\left(\frac{3}{\sqrt{\pi\sqrt{3}}}\right)\frac{2\pi rl}{AL}\right)^2}}{0.5\frac{1}{\left(\frac{2\pi rl}{AL}\right)^2}} = \frac{0.6}{0.5} \cdot \frac{\sqrt{3}\pi}{9} \quad . \qquad (27)$$

Alternatively, using the apparent radius concept,

$$\pi {R_{Circ}}^2 = t^2 \frac{\sqrt{3}}{4} \quad , \tag{28}$$

$$R_H = \frac{2\phi}{S} = \frac{2\left(t^2 \frac{\sqrt{3}}{4} \cdot \frac{l}{AL}\right)}{\frac{3tl}{AL}} = \frac{t}{2\sqrt{3}} = \frac{R_{Circ}\sqrt{\pi}}{\sqrt{3\sqrt{3}}} \quad , \tag{29}$$

$$R_A = \sigma \cdot R_{Circ} = \sqrt{\frac{R_H}{R_{Circ}}} R_{Circ} = \sqrt{\frac{\sqrt{\pi}}{\sqrt{3\sqrt{3}}}} R_{Circ} \quad . \tag{30}$$

Since effective permeability is proportional to R^4, and the geometric constant B for an equilateral triangular pipe is 0.6 and the geometric constant of a circular pipe is 0.5:

$$\frac{K_{tri}}{K_{circ}} = \frac{B_{tri}}{B_{circ}} \cdot \frac{\sqrt{3}\pi}{9} = \frac{0.6}{0.5} \cdot \frac{\sqrt{3}\pi}{9} \quad . \tag{31}$$

4.3 APPARENT RADIUS IN A PIPE WITH A THROAT

Consider two circular cross-section pipes of radius R. The pipes are identical with the exception of the second one having a throat at the middle of the pipe (Figure 4-4). There are insignificant differences in porosity and specific surface areas between these two pipes. In addition, since both pipes are straight, both have a tortuosity of 1. Then we might expect both permeabilities to be more or less the same. In reality, the permeability obtained using Lattice-Boltzmann simulations are different by multiple orders of magnitude. Using the Lattice-Boltzmann simulations, we can numerically solve for an effective permeability. Assume that the circular pipe has the radius of 0.072 mm, the throat has the radius of 0.002 mm, and the frame has the size 0.4x0.2x0.2 mm. This means that in the digital representation, the circular pipe has the radius of 36 pixels, the throat has the radius of 4 pixels, and the frame has the size 200x100x100 voxel with dx = 0.002 mm. The permeability is 267293 mD for the pipe without a throat, and 1585 mD for the pipe containing the throat, a difference of more than two orders of magnitude.

This example also shows that permeability is not uniquely related to porosity and specific surface area. In actual rock geometries, we can think of the throat effect as analogous a reduction of the size of the pore throats, resulting, for example, from cementation.

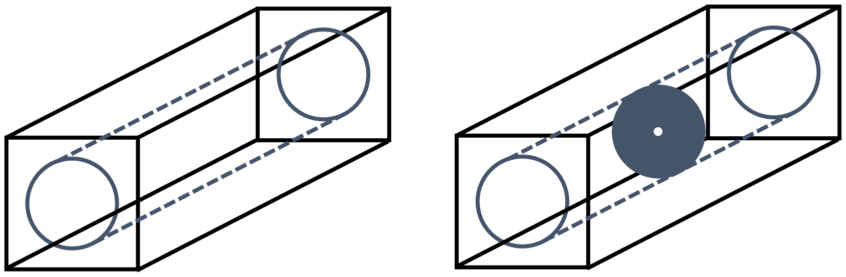


Figure 4-4: 3-D digital representation of a normal circular pipe and a circular pipe with a throat in the middle.

This problem can be solved by using the Kozeny-Carman equation with apparent radius. To get the approximate apparent radius, we run the permeability calculation using the Lattice-Boltzmann simulation and then use the Kozeny-Carman equation to solve for R. We can model the apparent radius in a pipe with a throat by using the harmonic mean of the radius of a throat R_B and the radius of the pipe R_{Circ}, in which the weights C_1 and C_2 are such that $C_1 + C_2 = 1$:

$$\frac{1}{R_A} = C_1 \cdot \frac{1}{R_B} + C_2 \cdot \frac{1}{R_{Circ}} \tag{41}$$

In our numerical simulations, we increase the size of the throat from 4 pixels to 36 pixels. The radius of the circular pipe is fixed at 36 pixels. The apparent radius model fits the result from numerical simulations if C_1= ¼ and C_2= ¾.

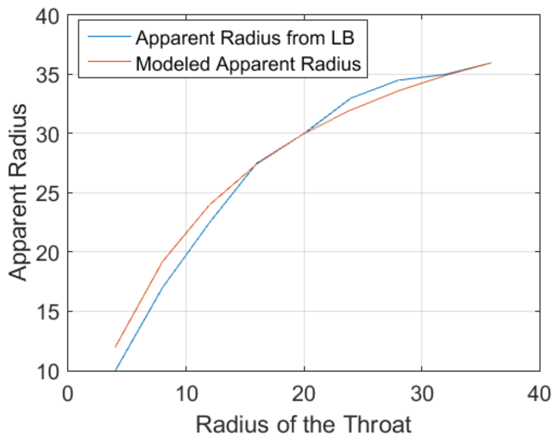


Figure 4-5: Comparison of the apparent radius calculated from Lattice Boltzmann simulations to that from the apparent radius model. The x axis represents the radius of the throat, which ranges from 4 to 36 pixels.

4.4 CONCLUSION

The concept of apparent radius helps us understand the effect of geometry on permeability for non-circular pipes. Of all the pipe shapes with the same porosity, the circular pipe has the highest permeability. This is because it has the highest ratio of porosity to specific surface area. Any deviation from a circular cross-section results in reduced permeability. Apparent radius also helps us use the Kozeny-Carman equation in a form that relates permeability to R^4. Although this method of using R_A still needs to be corrected using geometric factor B, B has minimal effect on permeability prediction, and its value is typically assumed to be 0.5 for all complex geometries. Our improved version of the Kozeny-Carman equation will require less calibration when it is used to fit the data.

Chapter 5

Tortuosity

Chapter 5 presents the detailed study of hydraulic tortuosity. The hydraulic tortuosity is one of the most important parameters in characterizing the heterogeneity of fluid flow in porous media. The most basic definition of tortuosity is the ratio of average flow path length to sample length. Although this definition seems straightforward, the lack of understanding and of proper ways to measure tortuosity make it one of the most abused parameters in rock physics. Often, the tortuosity is obtained from laboratory measurements of porosity, permeability, and specific surface area by inverting the Kozeny-Carman equation. This approach has a major pitfall, as it treats tortuosity as a fitting factor, and the inverted tortuosity is often un-physically high. In contrast, I obtained the tortuosity from 3-D segmented binary images of porous media using streamlines extracted from a local flux, the output from the Lattice Boltzmann flow simulation. After obtaining streamlines from each sample, I calculated the distribution of tortuosities and flux-weighted average tortuosity. With the tortuosity measurement from streamlines, every parameter in the KC equation can be measured accurately from 3-D segmented binary images. I found, however, that the KC equation is still missing some important geometric information needed to predict permeability. With known parameters and without a fitting factor, the KC equation predicts permeability higher by one to two orders of magnitude than that predicted by the LBM. I searched for a missing parameter by exploring various concepts such as connected pore space and pore throat distribution. I found that the connected pore space does not contribute to the difference between the KC permeability and LBM permeability, whereas, as I illustrate with sinusoidal pipe examples, the pore throat distribution captures what is missing from the Kozeny-Carman equation.

5.1 INTRODUCTION

Hydraulic tortuosity was first introduced by Carman in the Kozeny-Carman equation to account for the tortuous character of flow through a granular bed (Carman, 1937). The tortuosity (τ) is defined as l/L where l is the length of a sinuous track and L is the length of a sample. The Kozeny-Carman equation can be expressed in terms of porosity (ϕ), specific surface area (S), geometric factor (B), and tortuosity (τ):

$$\kappa = \frac{1}{2}\frac{\phi^3}{S^2\tau^2} \quad or \quad \kappa = B\frac{\phi^3}{S^2\tau^2} \quad . \tag{1}$$

The derivation of this equation is in chapter 3. This is the most frequently used and therefore most familiar form of the Kozeny-Carman equation, since porosity and specific surface area can be easily obtained in laboratory measurements. However, although the definition of tortuosity is straightforward, in practice, tortuosity is difficult to measure, especially in laboratory settings. The estimation of tortuosity requires visualization or an exact tracing of the flow paths, something that is difficult to achieve in natural rocks without destroying the samples. It is therefore a common practice to obtain the tortuosity by inverting the above equation with known porosity, specific surface area, and permeability from laboratory measurements:

$$\tau_{inv} = \sqrt{B\frac{\phi^3}{S^2\kappa}} \quad . \tag{2}$$

This practice often leads to an unphysically high value of the tortuosity. For example, Gomez (2010) reported a tortuosity of 2.5 on Fontainebleau sandstones, whereas the tortuosity from our streamlines approach is 1.46 (Gomez et al., 2010).

There are numerous literatures that focus on tortuosity. For example, Ghanbarian et al. (2013) provided an insightful review on tortuosity. Some researchers defined tortuosity as the shortest part through porous media (Arch and Maltman, 1990; Shepard, 1993; Clennell, 1997). Dullien (1991) described hydraulic tortuosity as the square of the flux-weighted average flow path to the sample length. Tortuosity is also often empirically related to porosity (Pech, 1984; Du Plessis and Masliyah, 1991; Koponen et al., 1997; Mauret and Renaud, 1997; Mota et al., 2001; Ahmadi et al., 2011)

We developed a new approach to finding tortuosity, conducting the numerical simulations of single phase flow using the Lattice Boltzmann (LB) simulation on 3-D segmented binary images (computeStreamlines.m). The LB simulation models the Navier-Stokes flow through the collisions of imaginary particles at a microscopic scale to solve for absolute permeability (Fredrich et al., 1999; Keehm, 2003; Andrä et al., 2013b). The output of LBM simulation is a local flux velocity field (ux, uy, uz), from which we can find the mean flux and permeability from Darcy’s equation. For each sample, we extracted streamlines from its local flux field and calculated the flux-weighted average tortuosity. The method is described in detail in the methodology section.

Once we acquired the flux-weighted average tortuosity, we could measure every variable in the Kozeny-Carman equation (Equation 1) except a geometric factor (B), which is typically assumed to be 0.5. As late as 2001, the specific surface area was difficult to measure and very sensitive to the scale of the images (Keehm et al., 2001), but now the codes from Minkowski functionals can yield accurate specific surface areas (Legland et al., 2007). We found, however, that the Kozeny-Carman equation still lacks a parameter needed to accurately predict permeability. In order to find this parameter, we are currently investigating multiple options, such as connected pore space and pore throat distribution from streamlines. We found that porosity and specific surface area from connected pore space images do not affect the permeability calculation from the Kozeny-Carman equation and that we can illustrate with sinusoidal pipe examples that the pore throat distribution captures what is missing from the Kozeny-Carman equation.

Our data consist of over 400 segmented binary 3-D image cubes containing (1) pipes of various cross-sections, (2) artificial and physical sphere packs, including simple cubic packs (SCP), face-centered cubic packs, and Finney packs, and (3) natural rocks including Fontainebleau sandstone, Bituminous sands, Berea sandstones, and Grosmont carbonates. The details of these microstructures are given in the Appendix A. We found that for most samples, the flux-weighted average tortuosity ranges from 1.2 to 1.6, except for straight pipes, for which the tortuosity is 1.

We subsampled all microstructures of natural rocks to the size 200x200x200 voxels to gain more samples and to test the variability of the tortuosity. Using subsamples may raise the concern that the sample volume is not at the representative elementary volume (REV). Generally, REV is defined as the smallest sample volume that can be representative of the whole rock for a given attribute (Bear, 1988). Therefore, REV is different for different physical properties. The Finney packs at large radii (LX = 3 and LX = 6) may not have reached the REV needed for simulating permeability since they have a small aspect ratio of cell size to grain diameter. However, we decided to include the results of these two volumes in the analysis since they have tortuosities similar to those of the Finney packs at small radii (LX = 9 and LX = 12). This also indicates that we can calculate the tortuosity from subsamples as well, and that the tortuosity requires even less REV than the permeability.

Due to our limitation in computational power at the time of the study, we mitigated this REV problem by also running the simulation on the entire volume of Berea sandstone with reduced resolution (from a resolution of 0.74 µm to one of 3.788 µm to reduce the voxels from 1024x1024x1024 to 200x200x200). The flux-weighted average tortuosity from the entire volume of Berea sandstone is 1.41, which is slightly higher than the average of flux-weighted average tortuosity of 1.38 from the subsamples of Berea sandstones.

5.2 DATA AND METHODOLOGY

We developed a method to extract streamline-based hydraulic tortuosity from local flux matrix output (Figure 5-1 and Figure 5-2). This method allows us to directly calculate tortuosity from its basic definition, the ratio of average flow path length to sample length.

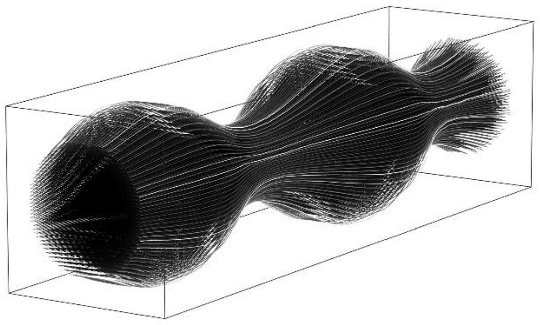


Figure 5-1: Streamlines in sinusoidal pore channels show laminar flow behavior (low Reynolds number).

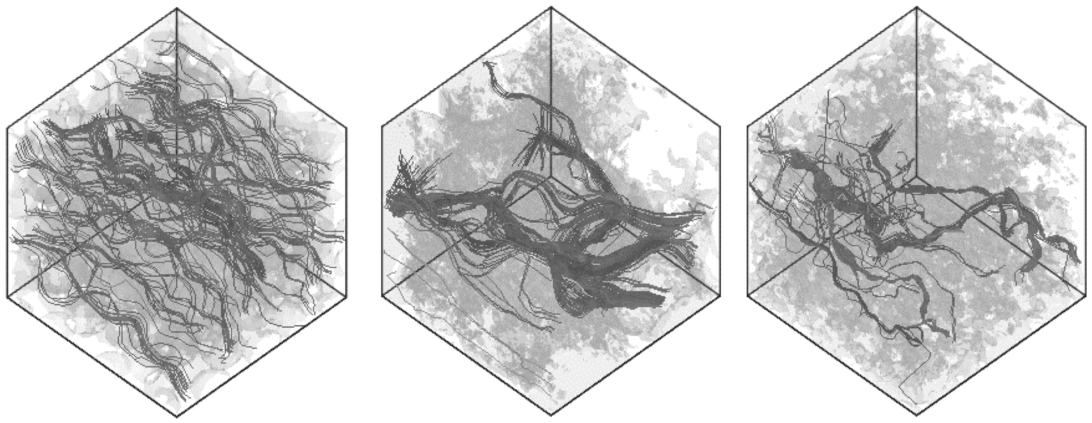


Figure 5-2: Streamlines with 3-D segmented binary images in light grey color in the background. From left to right: Finney pack, Berea sandstone, Grosmont carbonate.

For extracting streamlines, the first step is to run a Lattice Boltzmann (LB) simulation on a 3-D segmented binary image. The LB simulation approximates Navier-Stokes flow through the collisions of imaginary particles at a microscopic scale to solve for absolute permeability (Fredrich et al., 1999; Keehm, 2003; Andrä et al., 2013b). The output of the LB simulation is a local flux vector (ux, uy, uz) at each grid point, from which we can find the mean flux and permeability using Darcy's equation. More details of the LB simulation can be found in Appendix B.

For 3-D porous media, the local flux matrix comprises flux in the x, y, z directions for each voxel location. Since the mean flow is along the x-direction, we initialized a streamline for each pixel in a y-z cross-section. The streamlines step in the x-direction one pixel at a time, with the displacement vector determined from the ux, uy, uz local flux. Since after the first time step, the streamlines are unlikely to be on a grid, we interpolated the ux, uy, uz local flux using bilinear interpolation to obtain accurate velocity at a given location. We employed a no-flow boundary condition by eliminating any streamlines that touched the boundary of a sample. Only completed flow paths were used in the tortuosity calculation. After generating streamlines, we extracted the following information for each streamline:

1. XYZ coordinate

   A matrix (nx, 3) containing x,y,z coordinate locations along each streamline.

2. Absolute flux along streamlines

   A vector (nx, 1) containing absolute flux along each streamline $\left(A = \sqrt{u_x^2 + u_y^2 + u_z^2}\right)$.

3. Total distance

   A scalar containing total Euclidean distance along a streamline.

4. Total time

A scalar containing total Euclidean distance divided by Darcy's velocity.

5. Total flux

   A scalar containing total absolute flux along a streamline.

6. Individual streamline tortuosity

   A scalar containing total distance divided by sample length.

Since there are multiple streamlines for each sample, the tortuosity is a distribution. For each sample, we then calculated the tortuosity as follows:

1. Flux-weighted average tortuosity ($\tau_{FluxWeighted}$)

   The ratio of average streamline length, weighted by total flux of each streamline, to the length of a sample.

2. Mean tortuosity (τ_{Mean})

   The ratio of average streamline length to the length of a sample.

3. Minimum tortuosity (τ_{Min})

   The ratio of shortest streamline length to the length of a sample.

4. Maximum tortuosity (τ_{Max})

   The ratio of longest streamline length to the length of a sample.

5. Inverted tortuosity from Kozeny-Carman equation (τ_{KC})

   The inverted Kozeny-Carman equation given the known permeability, specific surface area, porosity, and a geometrical factor of 0.5 (Equation 2).

5.3 HYDRAULIC TORTUOSITY

Figure 5-3 shows examples for the distributions of individual streamline tortuosity. The black lines are flux-weighted average tortuosity and the red lines are inverted tortuosity from the Kozeny-Carman equation. We selected one sample of each of the following rocks: simple cubic pack (SCP), face-centered cubic pack (FCP), Finney pack, Fontainbleau sandstone, Bituminous sand, Berea sandstone, and Grosmont carbonate. From the distributions of individual streamline tortuosity, we calculated the minimum tortuosity, maximum tortuosity, mean tortuosity, and flux-weighted average tortuosity to represent the tortuosity of each sample. In this graph, the inverted tortuosities are all larger than the flux-weighted average tortuosities. We plot the inverted tortuosity vs. flux-weighted average tortuosity in Figure 5-4. The grey line represents a 1:1 reference line. Figure 5-4 helps to confirm that for most samples, the inverted tortuosity is larger than flux-weighted tortuosity, since the data lie to the right of the reference line. For all samples, excluding straight pipes, the flux-weighted tortuosity has quantiles P10 of 1.02, P50 of 1.34, and P90 of 1.49. In contrast, the inverted tortuosity has P10 of 1.63, P50 of 2.85, and P90 of 11.71. We advise against the latter method of calculating tortuosity since it treats tortuosity as a fitting factor in the Kozeny-Carman equation, and the Kozeny-Carman equation almost always yields unphysically high tortuosity compared to the flux-weighted tortuosity, even in the case of artificial sphere packs.

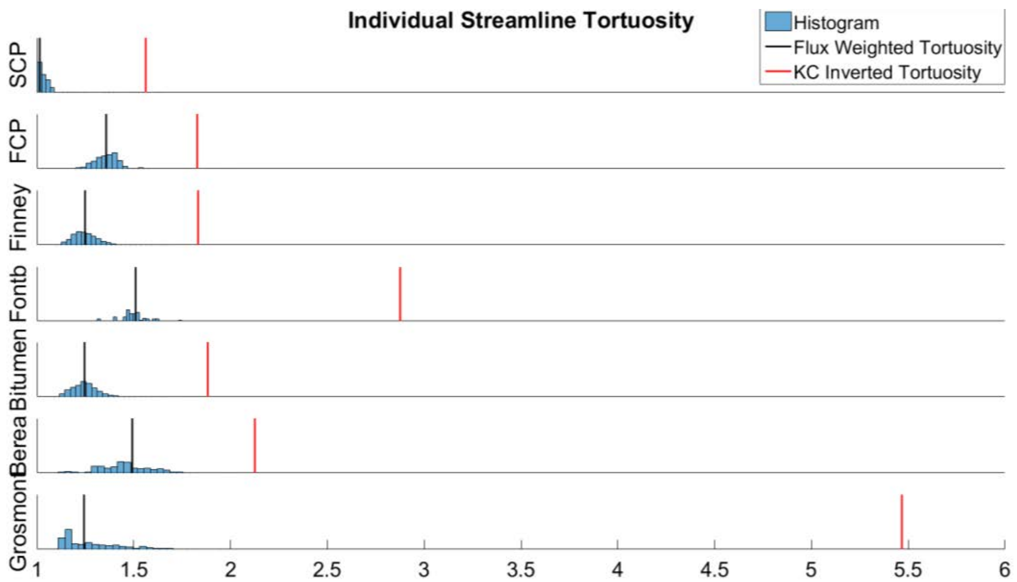


Figure 5-3: Individual streamline tortuosity from simple cubic pack (SCP), face-centered cubic pack (FCP), Finney pack, Fontainebleau sandstone, Bituminous sand, Berea sandstone, and Grosmont carbonate. The black lines show flux-weighted tortuosity and the grey lines show the inverted tortuosity from the Kozeny-Carman equation.

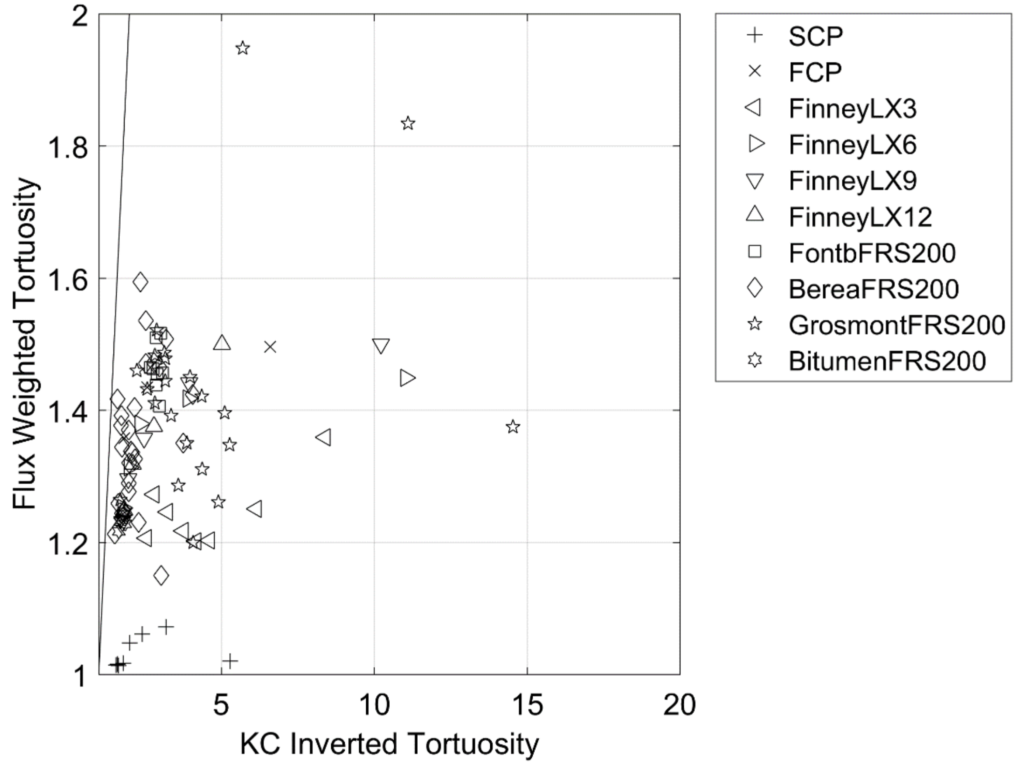


Figure 5-4: Inverted tortuosity from the Kozeny-Carman equation (assuming that the geometric factor B is 0.5) vs. flux-weighted tortuosity in sphere packs and natural rocks. For the Berea sandstone and Grosmont carbonate, we selected the subsets of the result from one of every five samples for the representation. The black line is a 1:1 reference line. This graph shows that the inverted tortuosity is higher than the flux-weighted tortuosity in most cases.

For representing the tortuosity of a sample, we chose flux-weighted tortuosity, which assigns higher weight to streamlines with higher flux value since streamlines with high flux are the ones that govern most of the flow. There is no significant difference between flux-weighted average tortuosity and mean tortuosity as observed from the cross-plot between these two quantities (Table 5-1).

In Figure 5-5, the flux-weighted average tortuosity is plotted against porosity. The face-centered cubic packs and Finney packs show a trend of increasing flux-weighted tortuosity when porosity decreases. The porosity reduction caused by grain dilation results in closing some of the flow paths and therefore in increased tortuosity. For SCP, the grain dilation barely affects tortuosity since the streamlines in SCP are mostly straight. For subsamples of natural rocks including Fontainebleau sandstone, Bituminous sand, Berea sandstone and Grosmont carbonate, we do not observe any significant relationship between porosity and tortuosity.

In Table 5-1, we report averages of flux-weighted average tortuosity, mean tortuosity, minimum tortuosity, and maximum tortuosity for artificial packs and natural rocks. In determining the average of the different types of tortuosity, we excluded 0, Inf, and NaN values. For artificial packs (labeled with an asterisk), we report only the tortuosity from an original image without any grain dilation. Note that the inverted tortuosity from the Kozeny-Carman equation is larger than the flux-weighted tortuosity in every case.

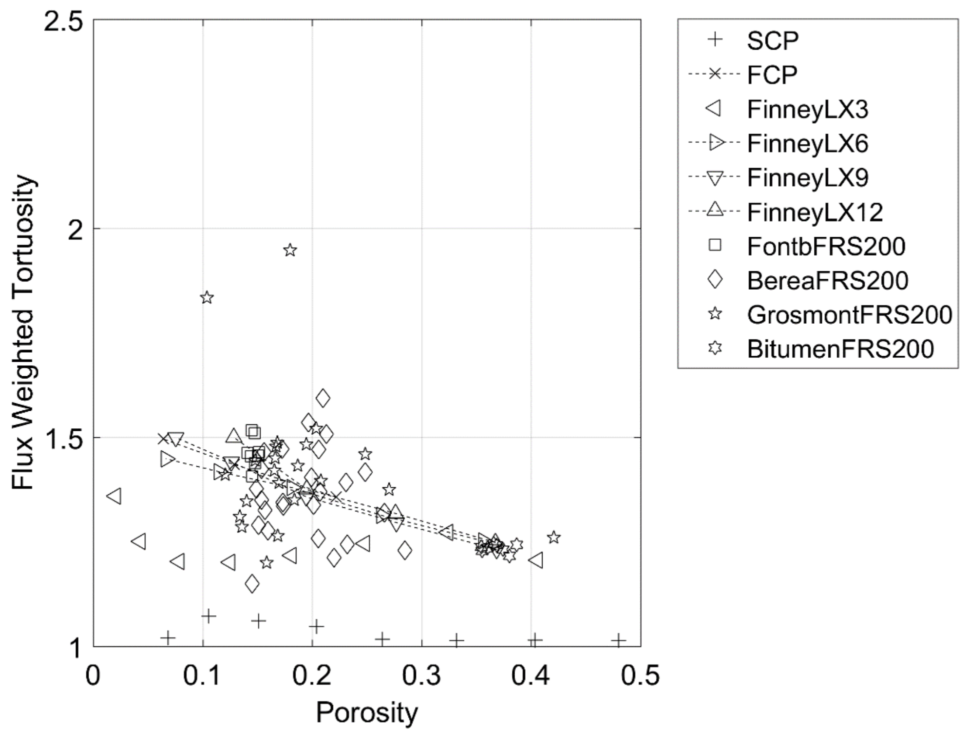


Figure 5-5: Porosity vs. flux-weighted tortuosity in sphere packs and natural rocks. For the Berea sandstone and Grosmont carbonate, we selected the subsets of result from one of every five samples for the representation. Finney packs and face-centered cubic packs with dashed lines show a clear trend of increasing flux-weighted tortuosity for an increase in grain/sphere boundary.

Table 5-1: The average of flux-weighted average tortuosity, mean tortuosity, minimum tortuosity and maximum tortuosity, and inverted tortuosity from the Kozeny-Carman equation. For artificial packs (with an asterisk), we report only tortuosities from an original image without any grain dilation.

	Simple cubic pack (SCP)*	Face-centered cubic pack (FCP)*	Finney packs (LX = 3)*	Finney packs (LX = 6)*	Finney packs (LX = 9)*	Finney packs (LX = 12)*	Fontainebleau sandstone	Bituminous sand	Berea sandstone	Grosmont carbonate
$\tau_{FluxWeighted}$	1.01	1.35	1.20	1.25	1.23	1.24	1.46	1.23	1.38	1.38
τ_{Mean}	1.02	1.35	1.21	1.26	1.23	1.25	1.46	1.24	1.37	1.37
τ_{Min}	1.00	1.20	1.06	1.07	1.08	1.10	1.27	1.09	1.13	1.17
τ_{Max}	1.63	2.00	1.75	1.96	1.85	1.86	1.89	2.03	2.25	2.20
τ_{KC}	1.56	1.82	2.54	1.78	1.77	1.83	2.90	1.78	2.55	7.85

5.4 APPLYING LB TORTUOSITY TO THE KOZENY-CARMAN EQUATION

Once we acquire the flux-weighted average tortuosity, we can measure every variable in the Kozeny-Carman equation (Equation 1) except a geometric factor (B), which is typically assumed to be 0.5 for natural rocks. Figure 5-6 shows the comparison between Lattice Boltzmann (LB) permeability and the Kozeny-Carman (KC) predicted permeability using flux-weighted average tortuosity and a geometric factor (B) of 0.5. Note that the permeability on the y-axis is on the log10 scale, and, therefore, these two permeabilities differ by one to two orders of magnitude. The graph also shows that the Kozeny-Carman equation can be misleading especially in the case of sinusoidal pipes (i.e., pipes whose radius varies along the direction of flow). As porosity increases and pore throat size decreases in sinusoidal pipes, LB permeability decreases while KC permeability increases (Figure 5-7). LB permeability vs. KC permeability is plotted in Figure 5-8, which clearly shows that for most samples, KC permeability is always equal to or greater, even by orders of magnitude, than LB permeability. Our ultimate goal is to find a physical parameter to shift data to a 1:1 reference line. Figure 5-8 also confirms that the Kozeny-Carman equation works well in the sphere packs. Since the differences between LB and KC permeability are mostly parallel to the 1:1 reference line, we require for sphere packs only a constant shift to fit the Kozeny-Carman equation. However, this is not the case for natural rocks since the differences between LB and KC permeability are highly scattered.

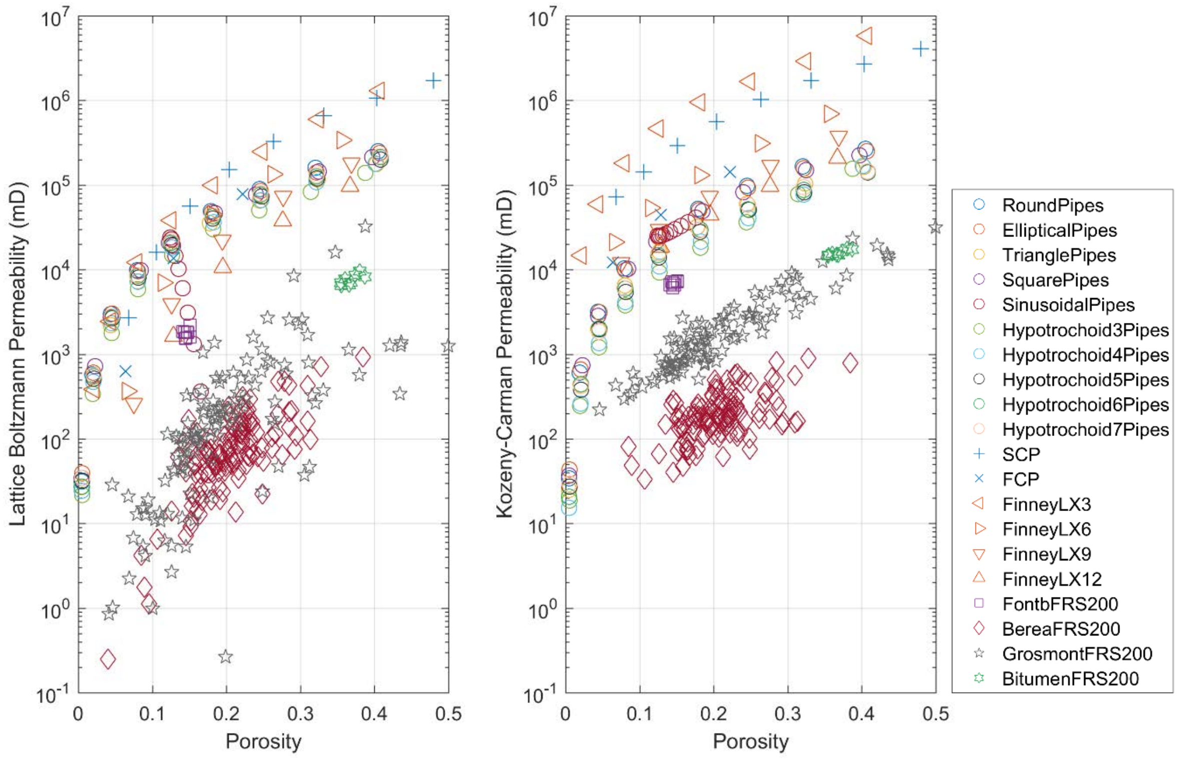


Figure 5-6: Porosity vs. Lattice Boltzmann (LB) permeability and Kozeny-Carman (KC) predicted permeability using flux-weighted average tortuosity and a geometric factor (B) of 0.5. The permeability is on the log10 scale. The sinusoidal pipes (red circles) show that the Kozeny-Carman equation can be misleading. As porosity increases and pore throat size decreases in sinusoidal pipes, LB permeability decreases, whereas KC permeability increases.

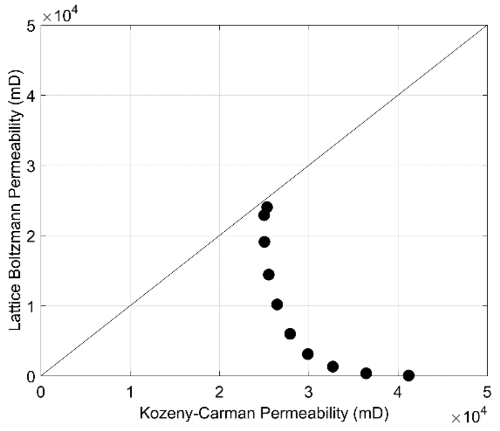


Figure 5-7: Lattice Boltzmann (LB) permeability vs. Kozeny-Carman (KC) predicted permeability using flux-weighted average tortuosity and a geometric factor (B) of 0.5 for sinusoidal pipes. Note that both axes are on a linear scale. The black line is a 1:1 reference line.

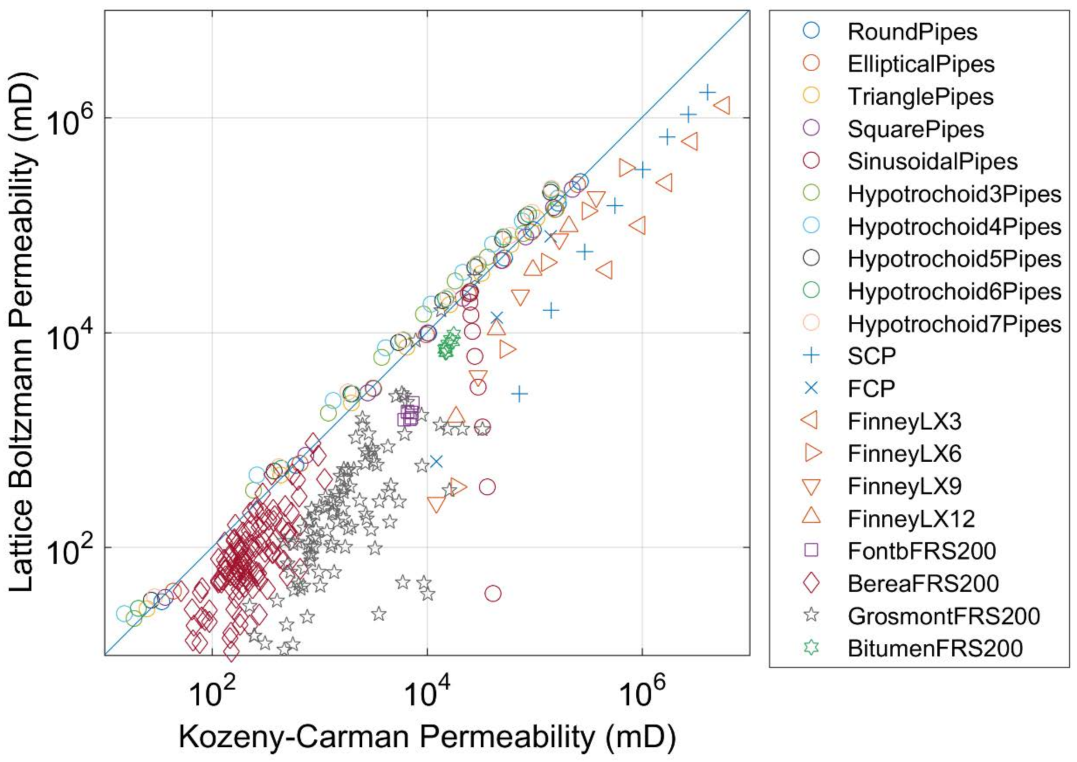


Figure 5-8: Lattice Boltzmann (LB) permeability vs. the Kozeny-Carman (KC) predicted permeability using flux-weighted average tortuosity and a geometric factor (B) of 0.5. For most samples, KC permeability overpredicts permeability by orders of magnitude. Note that both axes are on the log10 scale.

5.5 SEARCHING FOR THE MISSING PARAMETER

The difference between LB and KC permeability varies by orders of magnitude for different rocks and within the same rock. For example, Figure 5-8 shows that for Grosmont carbonate the difference can be anywhere from 0.3 order of magnitude to two orders of magnitude. If the difference were from the geometric factor (B), we would expect the geometric factor to be more or less the same within the same rock. B accounts for the variation in the cross-sectional shape of the pipes and typically has a small range. For example, B is 0.5 in a round pipe, 0.562 in a square pipe, and 0.6 in an equilateral triangle pipe (Mavko et al., 2009b). Since the difference between LB and KC permeability is large, the difference must result not from a geometric factor but from either the computation of parameters from 3-D segmented binary images or from the missing parameter in the Kozeny-Carman equation.

We first investigated the computation of porosity and specific surface area. For the flow within porous media, only the connected pore space contributes to fluid flow. However, when we calculate porosity and specific surface area, the calculation includes isolated pores (pores not connected to other pores) and therefore yields total porosity and total specific surface area. We, therefore, developed a workflow to obtain 3-D segmented binary images that have only the connected pore space. In order to find the connected pore space, we first label each group of connected pores by (1) selecting an unlabeled voxel, (2) labeling it and all the voxels connected to it, and (3) repeating steps 1 and 2 until all voxels in the pore space are labeled. To determine effective pore space, we find the labeled groups of connected pores that exist in all of the outermost 2-D slices from every face, which represent the connected pore space. The connected porosity and the connected specific surface area are always less than or equal to the total porosity and the total specific surface area. But, when we used both the connected porosity and the connected specific surface area in the calculation, we did not observe any significant change in the KC permeability (Figure 5-9). Therefore, this is not the cause of the difference between the KC and LB permeabilities.

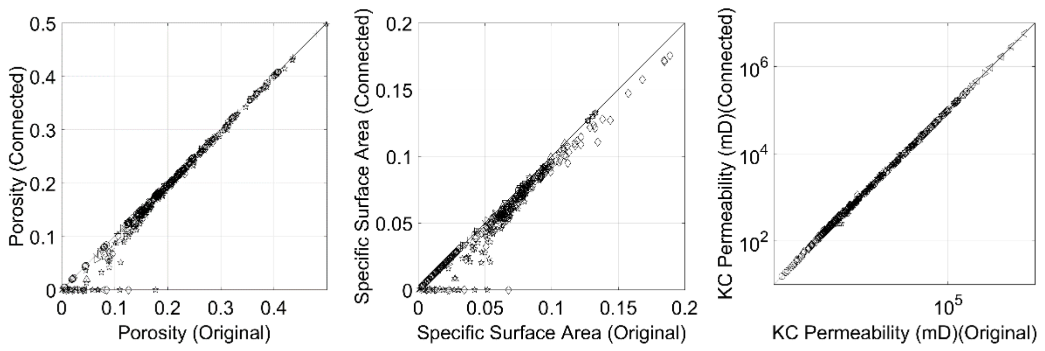


Figure 5-9: Comparison between porosity, specific surface area, and KC permeability of original images vs those of connected pore space images. The right plot shows that even though the connected porosity and the connected specific surface area are always less than the original ones, their effects cancel out when we calculate permeability from the KC equation.

Another possible cause of the difference is that the Kozeny-Carman equation lacks an important parameter governing the flow, and this can be illustrated with sinusoidal pipe examples. To do so, we started with the KC equation (Equation 1). However this equation lacks a parameter to capture variation in radius or pore throat along the sinusoidal pipes.

To account for this lack, we can approximate the analytic solution for the permeability of sinusoidal pipes by pipes of various radii in series. This will be explained in detail in the next chapter. We will show how to incorporate pore throat distribution to the Kozeny-Carman equation as a correction to permeability prediction.

5.6 CONCLUSION

In this chapter, we obtained the flux-weighted tortuosity using streamlines extracted from a local flux field of the Lattice Boltzmann simulation. This practice allowed us to calculate the tortuosity using its original definition, the ratio of average flow path length to sample length. We showed that the flux-weighted tortuosity of artificial packs, physical packs, and natural rocks are in a small range from 1.2 to 1.6. By comparison, the inverted tortuosity from the Kozeny-Carman equation is unphysically high. For example, the inverted tortuosity has a mean of 2.55 for Berea sandstone and a mean of 7.85 for Grosmont carbonate. We recommend against using this approach for finding tortuosity since it simply uses tortuosity as a fitting factor in the Kozeny-Carman equation.

As one acquires the flux-weighted average tortuosity, one can measure every variable in the Kozeny-Carman equation except a geometric factor, which is typically assumed to be a constant of 0.5 for natural rocks. For most samples, the Kozeny-Carman permeability is greater than or equal to the Lattice Boltzmann permeability, and the difference between these two permeabilities can vary by orders of magnitude even within the same rock. Therefore, the difference does not arise from a geometric factor but from either the computation of parameters from 3-D segmented binary images or from the missing parameter in the Kozeny-Carman equation. We explored (1) the computation of porosity and specific surface area using connected pore space and (2) the pore throat distribution. We found that the Kozeny-Carman equation lacks a parameter to describe pore throat distribution, which we illustrated with sinusoidal pipe examples.

Chapter 6

The Revised Kozeny-Carman

This chapter combines the knowledge of cross-section pore geometry and tortuosity, and I show that the pore size distribution is the missing parameter crucial for predicting permeability in porous media accurately. Based on this insight, I derive the revised Kozeny-Carman equation and show that it significantly improves the permeability estimation compared to the original equation for isotropic clastic rocks.

6.1 INTRODUCTION

In previous chapters, I showed that the well-known Kozeny-Carman (KC) equation cannot accurately estimate permeability, as is illustrated by sinusoidal pipe examples because pore size distribution is not included the equation. In this chapter, I derive the revised KC equation to include pore size distribution in the equation. The analysis was performed on sinusoidal pipes, simple cubic packs, face-centered cubic packs, Finney packs, Fontainebleau sandstones, and bituminous sands. The results show that the revised equation significantly improves the permeability estimation compared to the original equation for isotropic clastic rocks, without the help of a fitting parameter.

Furthermore, I show that it is possible to estimate permeability, which is a 3-D physical property, from 2-D images through the revised KC equation. The computational time for this method is also minimal compared to the time for typical permeability simulation such as Lattice Boltzmann (LB) flow simulations.

To arrive at these results, I thoroughly analyzed the representative pore size distribution by conducting the LB simulations on 3-D binary segmented images to obtain permeability and output local flux. I then used the velocity vector from the output local flux to trace out the streamlines. For each streamline I computed the pore size distribution along the streamline using the distance map in the binary images, as described in Chapter 2. The analysis from streamlines shows that the important variables in the representative pore size distribution are the maximum pore throat size and the representative pore body size. Another finding is that the pore size distribution, sorted or unsorted, yields similar estimated permeability. This is a positive result that allows researchers to use 2-D thin sections to predict permeability by modeling the sorted pore size distribution using various functions such as the linear function, the sinusoidal function, and Gauss error function (from cumulative distribution of Gaussian distribution). I then created the model of representative pore size distribution using these functions, with the starting point being the maximum of the minima of pore morphology (maximum pore throat) and the ending point being the minimum of the maxima of pore morphology (representative pore body). I showed that using these models in the revised KC equation enhances the estimation of permeability even though the inputs are from 2-D thin sections.

6.2 THE REVISED KOZENY-CARMAN EQUATION

Our revised equation assumes that the flow in porous media is similar to the flow in several pipes with any cross-section in series instead of the flow in a straight pipe with circular cross-section. First, we write the Hagen-Poiseuille equation, the exact solution of volumetric flow rate (q) in a round pipe of radius R, length l, pressure loss ΔP, and viscosity η. It is given by

$$q = -\frac{\pi R^4}{8\eta}\frac{\Delta P}{l} \quad . \qquad (1)$$

Then, the pressure loss along a pipe i in the series is

$$\Delta P_i = -\frac{8q\eta l_i}{\pi R_i^4} \quad , \qquad (2)$$

where l_i is the length and R_i is the radius of pipe i. Summing up Equation 2 gives the total pressure loss along pipes in series:

$$\Delta P_{total} = -\frac{8q\eta}{\pi}\sum_i \frac{l_i}{R_i^4} \quad . \qquad (3)$$

The total volumetric flow rate is then

$$q = -\frac{\pi \Delta P_{total}}{8\eta \sum_i \frac{l_i}{R_i^4}} \quad . \qquad (4)$$

The volumetric flow rate can be equated to Darcy's law, which describes volumetric flow rate (q) through a porous medium with permeability κ, cross-sectional area A, and length L:

$$q = -\kappa \frac{A}{\eta} \frac{\Delta P}{L} \quad . \qquad (5)$$

Hence, the permeability of several pipes in series is

$$\kappa = \frac{\pi L}{8A \sum_i \frac{l_i}{R_i^4}} \quad . \qquad (6)$$

If we know the flow of the pipes in series, we can meticulously calculate the porosity and specific surface area based on the shape of connected conical frustums (Figure 6-1). If we were instead to calculate the specific surface area directly from several pipes in series without assuming the conical frustum shape, the specific surface area would be overestimated.

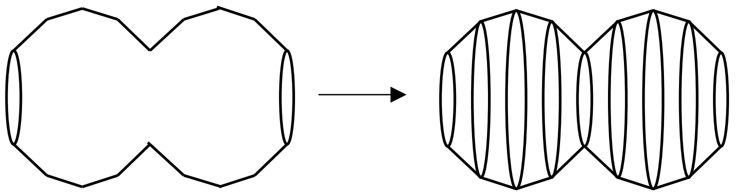


Figure 6-1: An example of connected conical frustums derived from discretizing a sinusoidal pipe.

The porosity of this connected conical frustum is approximately

$$\phi_f = \frac{\pi \sum_{i=1}^{nz} R_i^2 l_i}{AL} \quad . \qquad (7)$$

The specific surface area of the connected conical frustum is

$$S_f = \frac{\sum_{i=1}^{nz-1} (\pi \cdot (R_i + R_{i+1}) \cdot \sqrt{(R_i - R_{i+1})^2 + h_i^2})}{AL} \quad . \qquad (8)$$

The revised equation is derived by assuming that the primary flow path has the shape of segmented conical frustums in series to compute a correction factor γ. This correction factor is the ratio of the permeability of pipes in series (Equation 6) to the Kozeny-Carman permeability in its well-known form $\kappa = B\frac{\phi^3}{S^2\tau^2}$ (Equation 7 in Chapter 3), computed based on the shape of segmented conical frustums in series. Therefore, the parameters we need to estimate the permeability are the radius along the pipes in series R_i, the length of each pipe segment l_i, the total length of flow path L, the porosity of frustum shape $\phi_f$the specific surface area S_f, and the tortuosity τ_f. The tortuosity can be obtained from the streamlines, or we can assume that it is within the range of 1.2-1.6 (Srisutthiyakorn and Mavko, 2017). This correction factor can then be used as a correction to any original form of the Kozeny-Carman equation.

$$\gamma = \frac{\kappa_{exact(pipes)}}{\kappa_{KC(pipes)}} = \frac{\dfrac{\pi L}{8A\sum_i \dfrac{l_i}{R_i^4}}}{0.5\dfrac{\phi_f^3}{S_f^2\tau_f^2}} \tag{9}$$

The revised Kozeny-Carman equation is then

$$\kappa'_{KC} = \gamma \cdot \kappa_{KC} = \gamma \cdot \frac{1}{2}\frac{\phi^3}{S^2\tau^2} = \gamma \cdot \frac{1}{72}\frac{\phi^3}{(1-\phi)^2\tau^2}d^2 \quad . \tag{10}$$

6.3 IMPLEMENTATION

The main task for implementing the revised Kozeny-Carman equation is to find the most representative pore size distribution (R_i). We present two approaches we employ to find the pore size distribution: (1) streamlines on a distance map and (2) multiple 2-D thin sections.

6.3.1 Streamlines on distance map approach

The first approach to compute the pore size distribution is through a combination of streamlines and a distance map (Figure 6-2). This approach allows us to find the pore size distribution that corresponds to the major flow path and to understand the flow in porous media better. The first step is to solve the absolute permeability using the Lattice-Boltzmann (LB) method. We employed a Lattice-Boltzmann simulation with a time-dependent fixed flow rate, which handles the complex geometry of the pore space well (Keehm and Bosl, 2003). The fixed flow rate scheme simulates the pressure gradient along the flow path (Fredrich et al., 1999). It does not require mirroring of the pore space; instead, it adds a 15-pixel-wide buffer zone at the inlet and outlet faces (Keehm, 2003). Our calculation assumed no-flow boundary conditions on the side walls. The Lattice Boltzmann code is implemented in Windows C++ and wrapped in MATLAB; the inputs are a 3-D image and the size of a voxel.

After conducting the numerical simulations, the output of LB simulation is a local flux velocity field (ux, uy, uz), from which we were able to find the mean flux and permeability using Darcy's law. For each sample, we extracted streamlines from its local flux field. For each streamline, we followed the path along the Z-direction and calculated the distance from the streamline to the nearest solid pixel in the plane. This distance is the radius of the largest circle that will fit within the pore space. The final steps were to sort this pore size distribution in order to ensure that the pore size changes gradually along the primary flow path and then to calculate the correction γ by using Equation 7 to Equation 9, assuming that each pore size distribution has the shape of the connected conical frustum. We selected the pore size distribution from the streamline that yields the highest correction γ factor to be the representative pore size distribution.

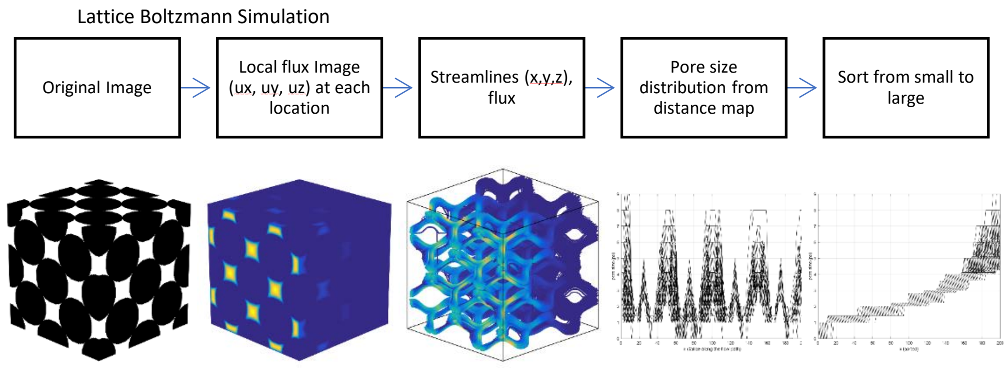

Lattice Boltzmann Simulation
Original Image
Local flux Image (ux, uy, uz) at each location
Streamlines (x,y,z), flux
Pore size distribution from distance map
Sort from small to large

Figure 6-2: An example workflow on extracting the representative pore size distribution using streamlines in a face-centered cubic pack.

For most samples in this isotropic clastic rock data set, these streamlines with maximum correction γ possess similar properties. They have a starting point, which is the maximum of the minima of pore morphology (the maximum pore throat), and an ending point, which is a point between the minimum to median of the minima of pore morphology (representative of the pore body). After defining these two points, we can model the pore size distribution using various equations as explained below. Figure 6-3 shows an example of how to model the pore size distribution using a sinusoidal reverse equation.

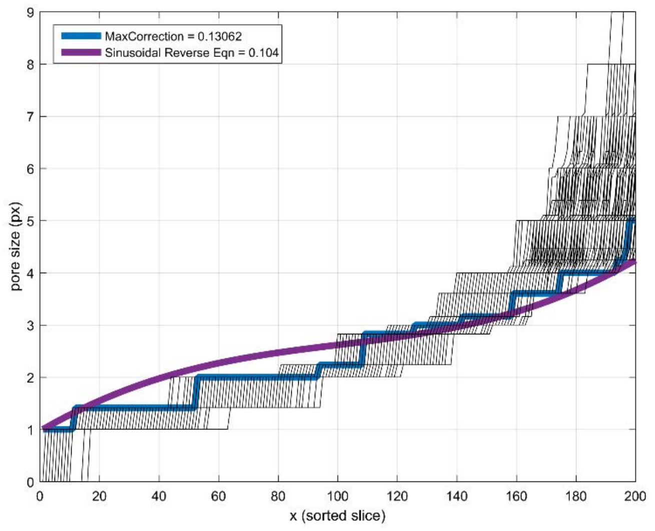


Figure 6-3: The comparison between the pore size distribution from streamlines and the modeled pore size distribution.

6.3.2 Multiple 2-D thin section approach

The second approach is to extract the pore size distribution from multiple 2-D thin sections by modeling the pore size distribution. The approach still has room for improvement, and we plan to invest time on this in the future to reduce the number of slices used. Currently we use 50 slices for each sample to obtain the pore size distribution model.

We first process the images by computing distance images individually for each thin section. After processing, we calculate the local minima and find the center location of each local minimum. We then use these locations to extract pore sizes from the distance image (Figure 6-4). Then for each thin section, we extract the minimum pore size and the maximum pore size. After combing the data for all the minima and all the maxima, we can find the maximum of the minima pore morphology and the minimum of the maxima pore morphology. After finding these two points, we model the sorted pore size distribution using various equations such as

- A linear equation: $y(x) = mx + c$
- A sinusoidal equation: $y(x) = \sin(x)$

- A Gauss error equation: $y(x) = \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$
- A sinusoidal reverse equation (the mirror image of the sinusoidal equation on a 45 degree line)
- A Gauss error reverse equation (the mirror image of the Gauss error equation on a 45 degree line)

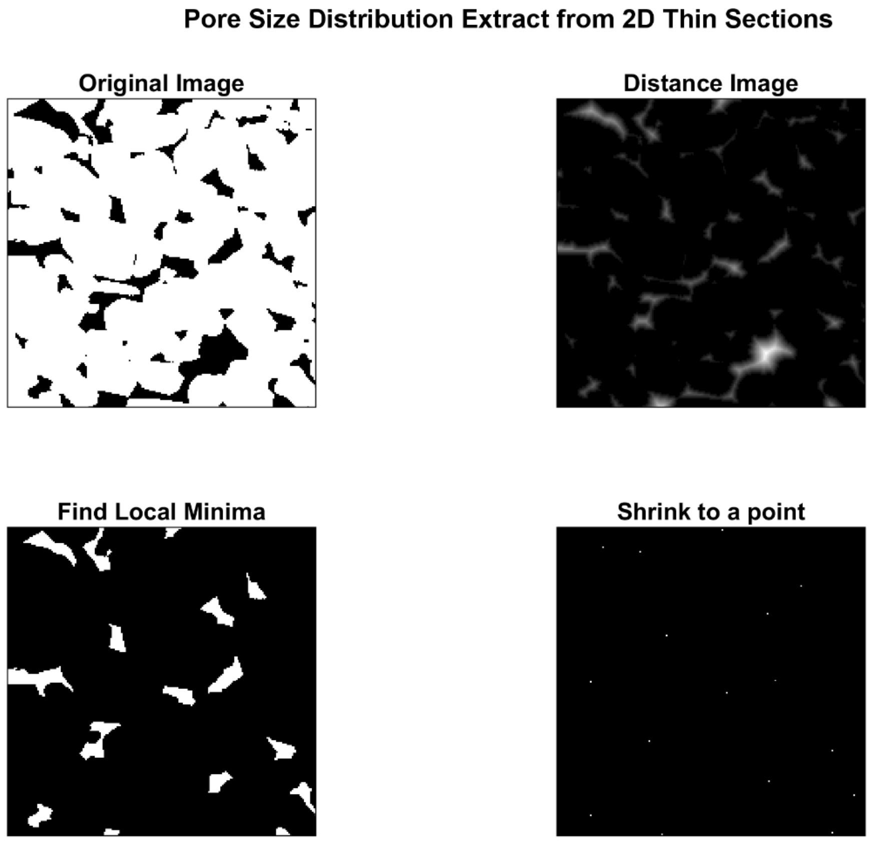


Figure 6-4: An example workflow of extracting the representative pore size distribution from 2-D thin sections.

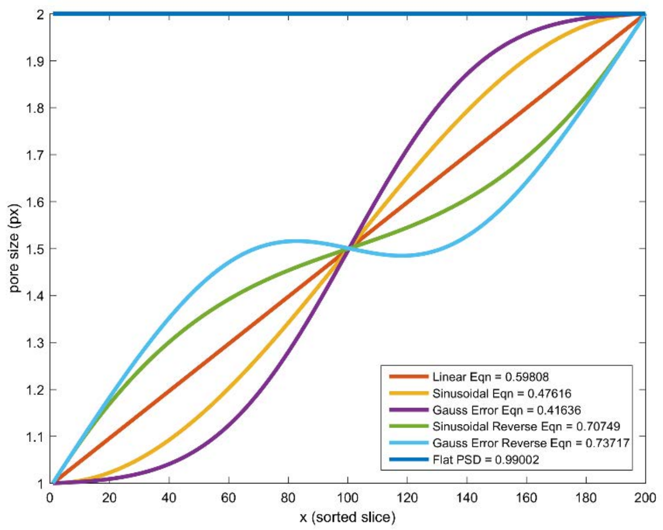


Figure 6-5: The representative pore size distribution model using various equations such as linear, sinusoidal, Gauss error, sinusoidal reverse, and Gauss error reverse.

Figure 6-5 shows the sensitivity of the correction γ calculated from the equations above for the ratio of pore body to pore throat of 2. The correction γ ranges from 0.4205 using the Gauss error equation to 0.7473 using the Gauss reverse equation. Further sensitivity of the correction is shown in Figure 6-6, where we plotted the correction for the different ratios of pore body to pore throat.

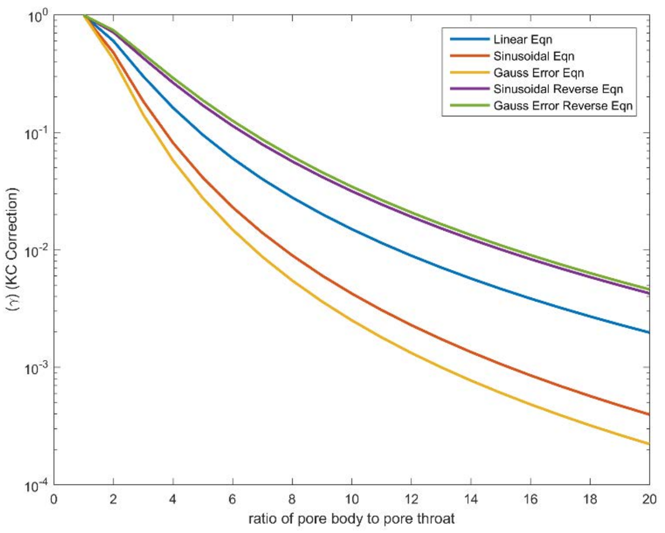


Figure 6-6: The sensitivity of the correction factor γ for various ratios of pore body to pore throat.

Table 6-1: Correction γ for different ratio of pore body to pore throat.

rBody/rThroat	Linear Equation	Sinusoidal Equation	Gauss Error Equation	Sinusoidal Reverse Equation	Gauss Error Reverse Equation
1	0.9980	0.9980	0.9980	0.9980	0.9980
2	0.6054	0.4813	0.4205	0.7170	0.7473
3	0.3049	0.1860	0.1428	0.4382	0.4744
4	0.1656	0.0825	0.0583	0.2728	0.3005
5	0.0979	0.0418	0.0279	0.1771	0.1962
6	0.0621	0.0234	0.0150	0.1200	0.1329
7	0.0417	0.0142	0.0088	0.0844	0.0933
8	0.0292	0.0091	0.0056	0.0613	0.0676
9	0.0212	0.0062	0.0037	0.0458	0.0503
10	0.0159	0.0043	0.0025	0.0350	0.0383
11	0.0122	0.0031	0.0018	0.0273	0.0298
12	0.0096	0.0023	0.0013	0.0217	0.0236
13	0.0076	0.0018	0.0010	0.0175	0.0190
14	0.0062	0.0014	0.0008	0.0143	0.0155
15	0.0051	0.0011	0.0006	0.0118	0.0128
16	0.0042	0.0009	0.0005	0.0099	0.0107
17	0.0035	0.0007	0.0004	0.0083	0.0090
18	0.0030	0.0006	0.0003	0.0071	0.0077
19	0.0026	0.0005	0.0003	0.0061	0.0066
20	0.0022	0.0004	0.0002	0.0053	0.0057

6.4 RESULTS

Figure 6-7 shows the comparison between Lattice Boltzmann (LB) permeability, and the Kozeny-Carman (KC) predicted permeability – calculated using flux-weighted average tortuosity and a geometric factor (B) of 0.5 – before the correction. Note that the permeability on the y-axis is on the log10 scale, and, therefore, LB and KC permeabilities differ by one to two orders of magnitude. This is the reason that the KC equation often requires a fitting parameter. The figure also shows that, prior to the correction, the Kozeny-Carman equation is invalid particularly in the sinusoidal pipe examples. As porosity increases and pore throat size decreases in sinusoidal pipes, LB permeability decreases while KC permeability increases. For most samples, the original KC permeability is always greater than or equal to the LB permeability by orders of magnitude.

When we applied the correction (Figure 6-8), the permeability prediction from the KC equation improves significantly. For this figure, we used unsorted pore size distribution along the streamline using the distance map. For simple cubic packs (SCP) and Face-centered cubic packs (FCP), the inaccurate permeability predictions are mostly for the realizations with high value of grain dilation (the realization with low porosity and, therefore, low permeability).

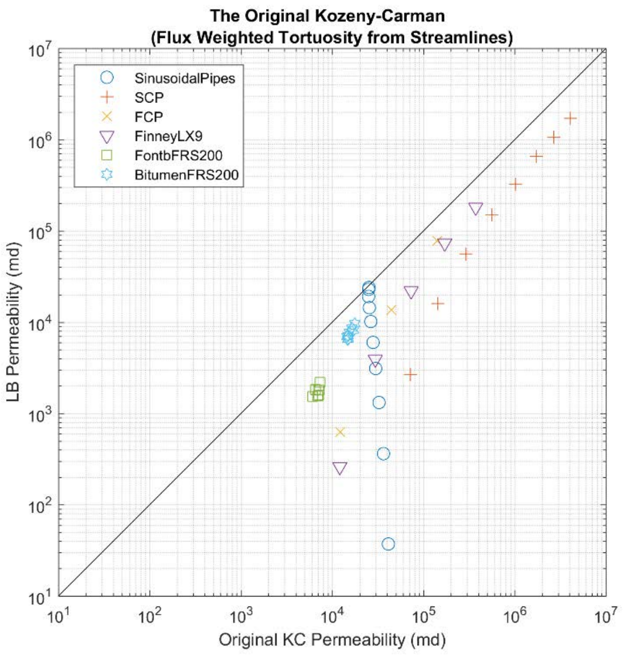


Figure 6-7: Lattice Boltzmann (LB) permeability vs. the Kozeny-Carman (KC) predicted permeability using flux-weighted average tortuosity and a geometric factor (B) of 0.5. For most samples, KC permeability overpredicts permeability by orders of magnitude. Note that both axes are on the log10 scale. The inaccuracy of KC permeability is prominent in sinusoidal pipes.

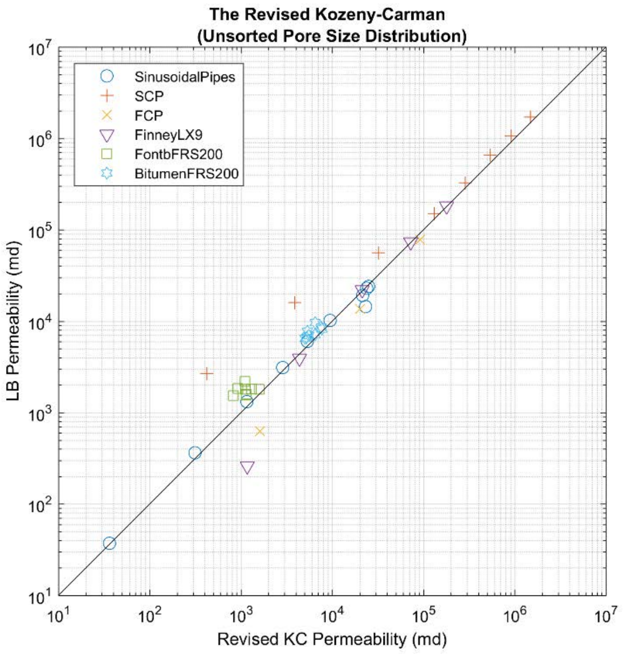


Figure 6-8: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with unsorted pore size distribution along the streamlines. Note that both axes are on the log10 scale.

Figure 6-9 shows a comparison between Lattice Boltzmann (LB) permeability and the revised Kozeny-Carman (KC) predicted permeability with sorted pore size distribution. We found that the pore size distribution, sorted or unsorted, yields similar predicted permeability. This confirms that knowing every detail of pore size distribution has minimal impact on predicting permeability and allows us to model sorted pore size distribution using various functions as mentioned before.

Figure 6-10 to Figure 6-14 show the permeability predicted using pore size distribution extracted from multiple 2-D thin sections in various models. The results are plotted without any empirical fitting. Although the prediction using only 2-D thin sections is not as good as that using streamlines, compared to the original KC equation the plots still show that the permeability prediction has improved significantly.

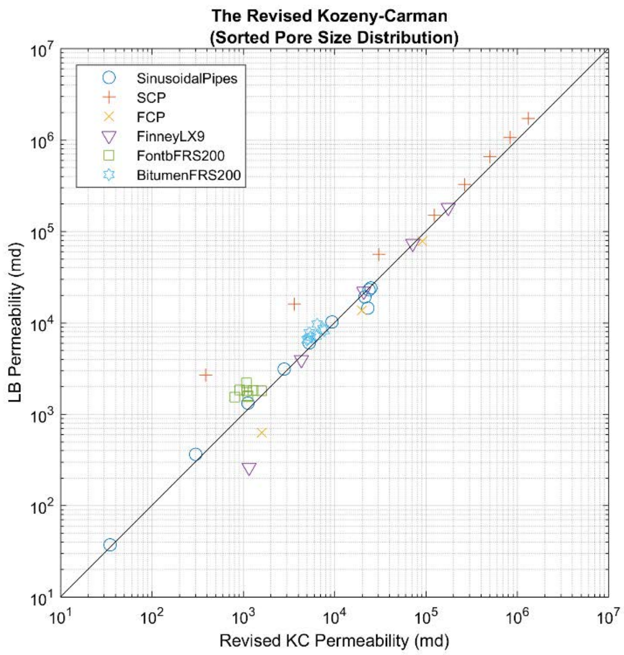


Figure 6-9: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with sorted pore size distribution along the streamlines. Note that both axes are on the log10 scale.

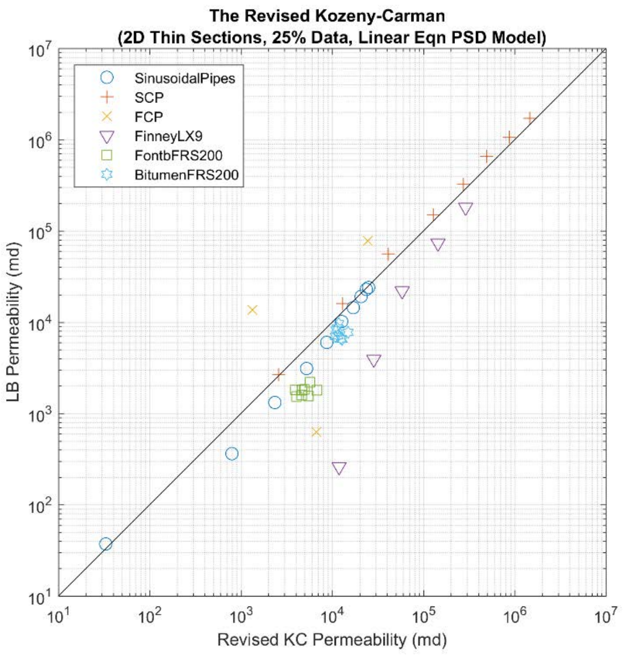


Figure 6-10: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with linear equation pore size distribution model.

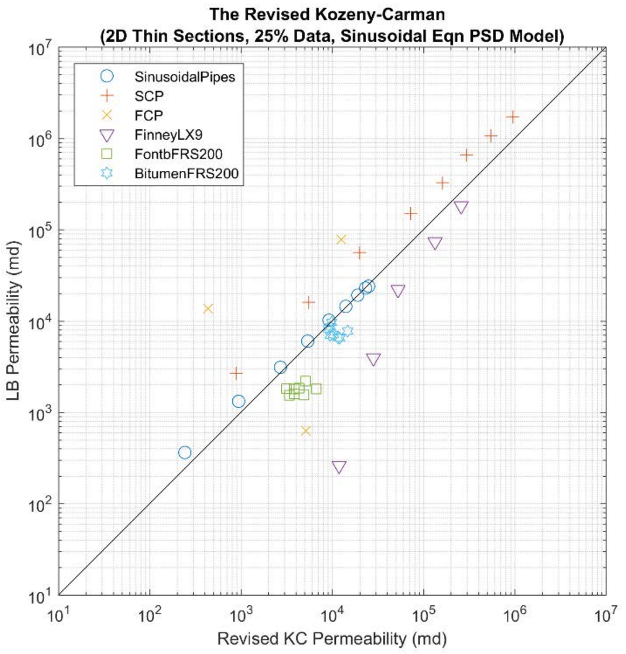


Figure 6-11: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with sinusoidal equation pore size distribution model.

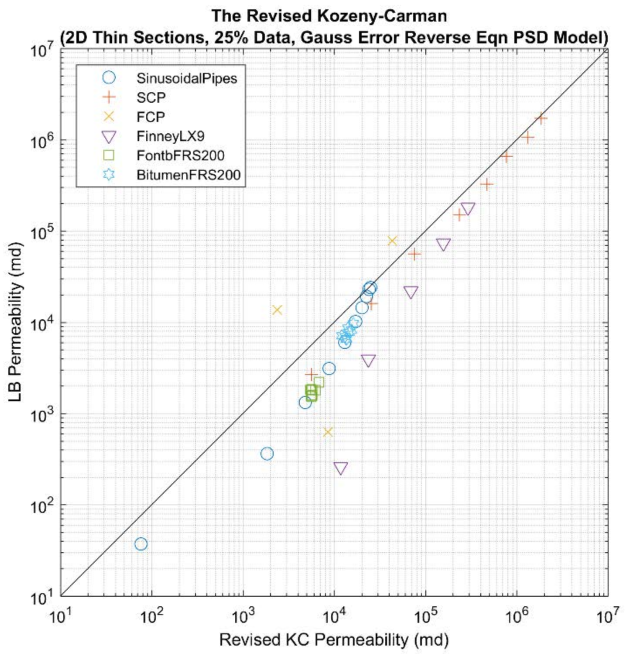


Figure 6-12: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with Gauss error equation pore size distribution model.

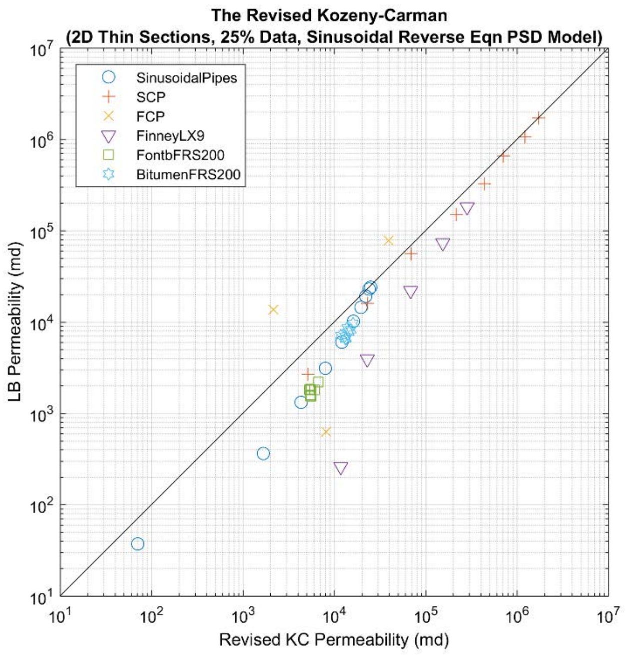


Figure 6-13: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with sinusoidal reverse equation pore size distribution model.

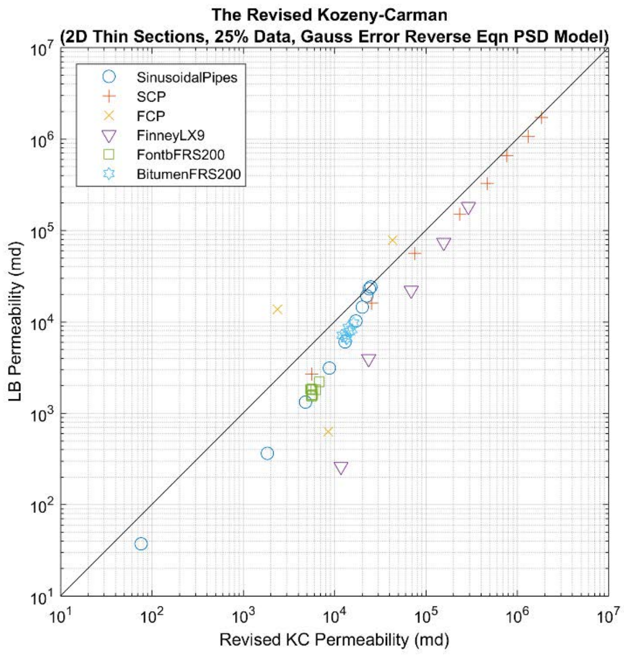


Figure 6-14: Lattice Boltzmann (LB) permeability vs. the revised Kozeny-Carman (KC) predicted permeability with Gauss error reverse equation pore size distribution.

6.5 CONCLUSION

We proposed a revised Kozeny-Carman (KC) equation, which is based on samples' pore size distribution and apparent radius. This revised KC equation shows that the pore size distribution is an essential parameter in the permeability estimation. It also solves the problem of misusing either tortuosity or geometric factor as a fitting parameter in the KC equation. The pore size distribution in the revised equation can be obtained from either 3-D μXCT segmented binary images or 2-D thin sections. We showed that using these models in the revised KC equation enhances the estimation of permeability without introduce empirical fitting even though the inputs are from 2-D thin sections.

Chapter 7

Machine Learning in Digital Rock Physics

This chapter presents machine learning methods for predicting physical properties from binary segmented images. Instead of using these conventional numerical simulations, I developed machine learning methods and show that it is possible to predict 3-D transport properties, by using geometrical features from both 2-D and 3-D μXCT binary segmented images. Both multilayer neural network (MNN) and convolutional neural network (CNN) algorithms are employed to predict permeability. Training is performed through both feed-forward and back-propagation with Bayesian Regularization by using a gradient descent algorithm. The inputs for MNN can be geometrical parameters such as Minkowski Functionals (porosity, specific surface area, integral of mean curvature (for 3-D), and Euler number). For CNN, the inputs are either 2-D or 3-D binary images.

7.1 INTRODUCTION

Permeability is one of the keys to understanding the nature of a hydrocarbon reservoir and estimate its production capability. For single-phase fluid flow, the Lattice-Boltzmann (LB) simulation is the established method for solving for absolute permeability. The LB simulation approximates the Navier-Stokes equations at the pore scale, but the calculation can be computationally expensive for large digital rock images (Keehm, 2003).

In contrast, geometric measurements and 2-D/3-D patterns are computationally inexpensive even at larger scales, and they can provide insights into the structure of the pores and perhaps into how the structure relates to the flow properties. In this project, machine learning methods were applied to understand the relationship between geometry (extracted features of rock images) and permeability, in the hope to improve accuracy and reduce computational time of permeability calculation.

7.2 DATA PROCESSING

Data includes 3-D binary segmented images of Berea sandstone and Fontainebleau sandstone. We subsampled the binary images to obtain more samples for machine learning (64 images of size 50 voxels from Fontainebleau Sandstone and 1000 images of size 100 voxels from Berea Sandstone). Permeability was computed from the LB simulation for each image to be used as the target in supervised learning. Raw inputs are binary segmented 3-D images, where at each voxel, the value 1 represents solid and 0 represents pore. Each type of feature was extracted from 3 multi-scales: (a) original image, (b) 5x-upscaled image (Figure 7-1), and (c) 10x-upscaled image. The upscaling was done by averaging. Upscaling ensures that the convolution of the 2-D and 3-D patterns capture the global patterns in addition to the local ones.

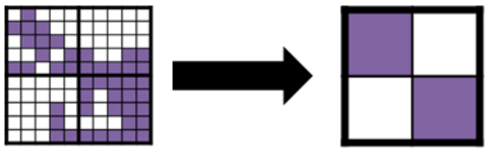


Figure 7-1: Example for the 5 times upscale in 2D images.

7.3 FEATURE EXTRACTION

7.3.1 Minkowski Functionals

Minkowski Functionals encompass standard geometric measurements for a binary segmented image. For a d-dimensional space, there are d+1 associated Minkowski measurements (Vogel et al., 2010). For example, a 2-D slice can be defined by 3 Minkowski measurements (area, perimeter, Euler characteristic), and a 3-D solid can be defined by 4 Minkowski measurements (M_0 - volume, M_1 - surface area, M_2 - integral of mean curvature (mean breadth), and M_3 - Euler characteristic). The units of Minkowski measurements are $\mathrm{L}^3, \mathrm{L}^2, \mathrm{L}^1$, for M_0, M_1, M_2, respectively, while M_3 is dimensionless.

M_0 - Volume (L^3)

$$\mathrm{M}_0(\mathrm{X}) = \mathrm{V}(\mathrm{X})$$

M_1 - Surface Area (L^2)

$$\mathrm{M}_1(\mathrm{X}) = \int_{\delta \mathrm{x}} \mathrm{ds} = \mathrm{S}(\mathrm{X})$$

M_2 - Integral of Mean Curvature (L)

$$\mathrm{M}_2(\mathrm{X}) = \frac{1}{2}\int_{\delta \mathrm{x}} \left[\frac{1}{\mathrm{r}_1} + \frac{1}{\mathrm{r}_2}\right] \mathrm{ds} = \mathrm{C}(\mathrm{X})$$

M_3 - Euler Characteristic (Unitless)

$\mathrm{M}_3(\mathrm{X})$ = vertices - edges + faces – solid

7.3.2 2-D Pattern Distribution

2-D pattern distribution can be derived from the convolution between the pattern and the image. For 2-D, patterns derived from a cross shape template such as that shown in Figure 7-2 has been employed. Four pixels adjacent to the center form the template. Thus, there are 2^4 = 16 combinations of the pattern (Figure 7-2). After convolution, the number of times that the pattern appears in the image can be obtained directly by counting the pixels that have the value that equals the number of pixels in the pattern (Figure 7-3). If the inputs are from 3 multi-scales (original, 5x upscale, 10x upscale) then the total number of 2-D patterns is 16*3 = 48 features.

For 2-D patterns, if 3-D images are 50x50x50 pixels, then the 3-D images can be sliced into 50 2-D images, and the pattern can be added from every 2-D image to form the pattern distribution.

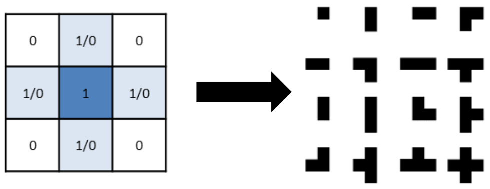


Figure 7-2: 16 2-D patterns derived from the cross-shape template.

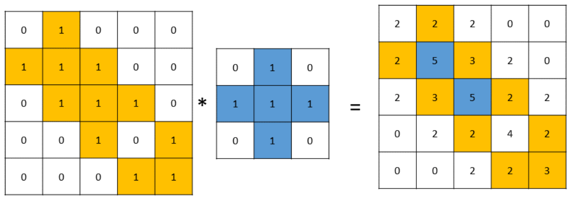


Figure 7-3: The convolution of a sample image and a pattern. After convolving, the pixel that has value 5 (the number of total pixel in the pattern) indicates the location of the pattern found in the image.

7.3.3 3-D Pattern Distribution

For 3-D patterns, there are 6 pixels adjacent to the center of the 3-D cross-shape template, resulting in 2^6 = 64 combinations of pattern. If inputs are from 3 multi-scales then the total number of 3-D pattern is 64*3 = 192 Features.

7.4 METHODOLOGY

To train the network, both multilayer neural network and convolutional neural network can be used. I divided the data into a 3:1:1 ratio for the training, testing and validation set. For 5 groups of data, if one group is selected to be the test set and the rest of the data are training sets, this results in a 5-fold calculation. However, in both methodologies, one validation data set is required to stop the training early, in order to prevent over-fitting. Hence, there are a total of 20-fold combinations of cross-validation. After 20-fold calculations, the mean square errors are calculated from the average of 20 cases. For each type of network, I varied the number of nodes (5,10,20) and hidden layers (1 to 5).

7.4.1 Multilayer Neural Network (MNN)

There are four steps for neural network design: (1) create a network, (2) configure the network, (3) train the network, and (4) validate the network. Questions pertaining to network configuration include how to divide data for training, testing, and validating the network, and what would be an appropriate number of nodes in a hidden layer. Answers to these questions are vital for deriving and constructing a better network.

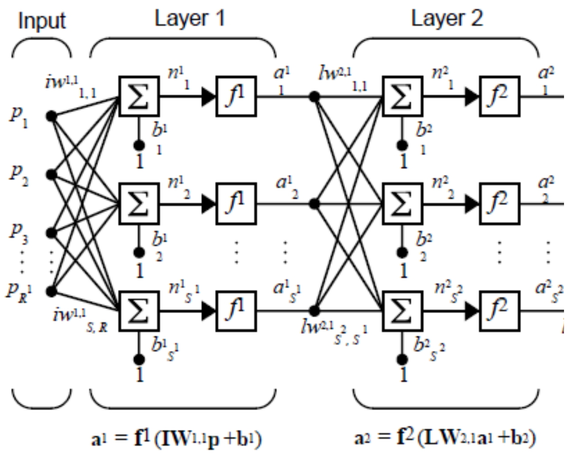


Figure 7-4: Example of the network architecture (Demuth, 2002).

I tested a tan-sigmoid function (a = tanh(x)) and a positive or rectified linear function (a = max(0,x)) in the hidden layers, and I used a linear function (a = x) in the output layers. I also tested the number of nodes and the number of hidden layers to obtain the optimal network structure using the feed-forward neural net.

Training is done through feed-forward and through back-propagation with Bayesian Regularization by using a gradient descent algorithm. Bayesian Regularization can be used to help achieve the goal of improved generalization. This can be done by adding to the previous performance function another term that includes the mean of the sum of the squares of the network weights and biases. To iteratively find weight and bias, I employ the Levenberg-Marquardt algorithm, which is a combination of the Gauss-Newton and the Steepest Descent algorithm. The combination ensures that the Hessian matrix (H) is invertible. If $\mu = 0$, the Levenberg-Marquardt algorithm is equivalent to the Gauss-Newton algorithm and if $\mu \rightarrow \infty$, the equation approaches the Steepest Descent algorithm:

$$\chi_{k+1} = \chi_k - [H(x_k) + \mu I]^{-1} J^T(x_k) e(x_k),$$

where e is the error vector, H is the Hessian Matrix, and J is the Jacobian matrix.

The performance can be regularized to prevent over-fitting as follows:

$$\mathrm{msereg} = (1-\delta)\cdot\frac{1}{N}\sum_{i=1}^{N}(e_i)^2 + \delta\cdot\frac{1}{M}\sum_{i=1}^{M}(w_i)^2$$

With this performance function, it is possible to minimize both mean square errors and mean square weights. The function therefore forces the network to have a smaller weight and bias, leading to smoother output.

7.4.2 Convolutional Neural Network

The convolutional neural network contains one extra layer (a convolution layer), which appears before the general multilayer neural network that is described in the previous section. For the convolutional layer, the features are 2-D and 3-D pattern distributions extracted from original images and upscaled images.

7.5 RESULTS

Table 7-1 and Table 7-2 summarize the performance of neural networks through mean-square errors (MSEs) for feed-forward (FF) and Bayesian Regularization (BR). For both FF and BR networks, obtaining features in multi-scales helps to lower test MSEs gradually, except for the case of multilayer neural networks (MNN) with Minkowski Functionals. The convolution at a larger scale may help capture global geometry and pore space connectivity from both 2-D and 3-D image inputs.

For FF networks, test MSEs of all cases are higher than train MSEs, as expected. The advantage of FF networks is that less time is required to train them, as seen from the number of iterations in Table 7-1. On the other hand, BR networks have test MSEs as good as or better than train MSEs due to the regularization feature. The Bayesian regularization algorithm prevents over-fitting by regularizing the function to minimize both weight and error. As shown in Table 7-2, the exception to this case is 2-D and 3-D CNN with no multi-scale, which are over-fitted as test MSEs are higher than train MSEs.

CNN with 2-D convolution with multi-scales (original, 5x upscale, 10x upscale) shows the best testing result overall from both FF and BR. The highest test MSE is from the Minkowski Functionals at the original scale, which is as expected since it contains only 4 features. The regression plots of the predicted data versus the target of features are shown in Figure 7-5 and Figure 7-6. The permeability prediction is generally in agreement with the target permeability.

Figure 7-7 shows both training and test MSE for each model and for different network architectures from the feed-forward neural net, where the x axis represents the number of nodes (5, 10, 20) and the y axis represents the number of hidden layers. There is no unique network architecture for digital rock images as different types of features have different optimum neurons and hidden layers. Although there is no unique answer, there are two observable trends from the feed-forward test MSE: (1) CNNs with multi-scale favor larger networks with a greater number of nodes in hidden layers and (2) MNN and 2-D CNN with only original images favor small and simple networks. This may be due to the number of features supplied to the network being small.

7.6 CONCLUSION

Deep Learning algorithms such as MNN and CNN can provide insights into important geometrical features in porous media. For example, obtaining cross-shape features in multi-scales helps improve the prediction because the patterns from a larger scale ensure that the neural network captures global pore connections in 2-D/3-D images.

The results of using machine learning to predict permeability are promising, especially for the case of 2-D CNN in multi-scales. As the cost of acquiring 2-D digital images is lower than that of acquiring 3-D digital images, 2-D CNN offers a good alternative for permeability prediction when 3-D images are too costly. In the future, we plan to include more features, such as pore size distribution, which can be extracted directly from the 2-D/3-D binary segmented images. Other interesting topological descriptors that can be used in machine learning include lineal-path, chord-length density function, pore-size function (Torquato, 2002; Lehmann et al., 2008) and Persistence (Zomorodian and Carlsson, 2005).

Table 7-1: Results from Feed-Forward MNN using tan-sigmoid function.

Model from Feed Forward Network	**# Features**	**Train MSE**	**Test MSE**	**Iterations**	**Correlation Value R**
MNN with Minkowski Functionals	4	0.2953e4	3.4147e5	10	0.58684
MNN with Minkowski Functionals (multi-scale)	504	1.4159e5	3.3678e5	10	0.87675
CNN with 2-D Convolution	16	1.6324e5	3.6123e5	16	0.50192
CNN with 2-D Convolution (multi-scale)	48	0.8750e5	2.4307e5	13	0.92475
CNN with 3-D Convolution	64	1.6410e5	3.3144e5	11	0.63542
CNN with 3-D Convolution (multi-scale)	192	0.6352e5	2.7400e5	10	0.90228

Table 7-2: Results from Bayesian Regularization MNN using rectified linear or positive linear function.

Model from Bayesian Regularization Network	**# Features**	**Train MSE**	**Test MSE**	**Iterations**	**Correlation Value R**
MNN with Minkowski Functionals	4	2.1379e5	1.1631e5	53	0.76999
MNN with Minkowski Functionals (multi-scale)	504	4.6362e5	2.3999e5	416	0.96225
CNN with 2-D Convolution	16	1.6802e5	5.2754e4	50	0.88749
CNN with 2-D Convolution (multi-scale)	48	1.1989e5	1.1049e5	77	0.95353
CNN with 3-D Convolution	64	2.1789e5	8.4937e4	57	0.86071
CNN with 3-D Convolution (multi-scale)	192	4.6362e5	2.3999e5	283	0.97467

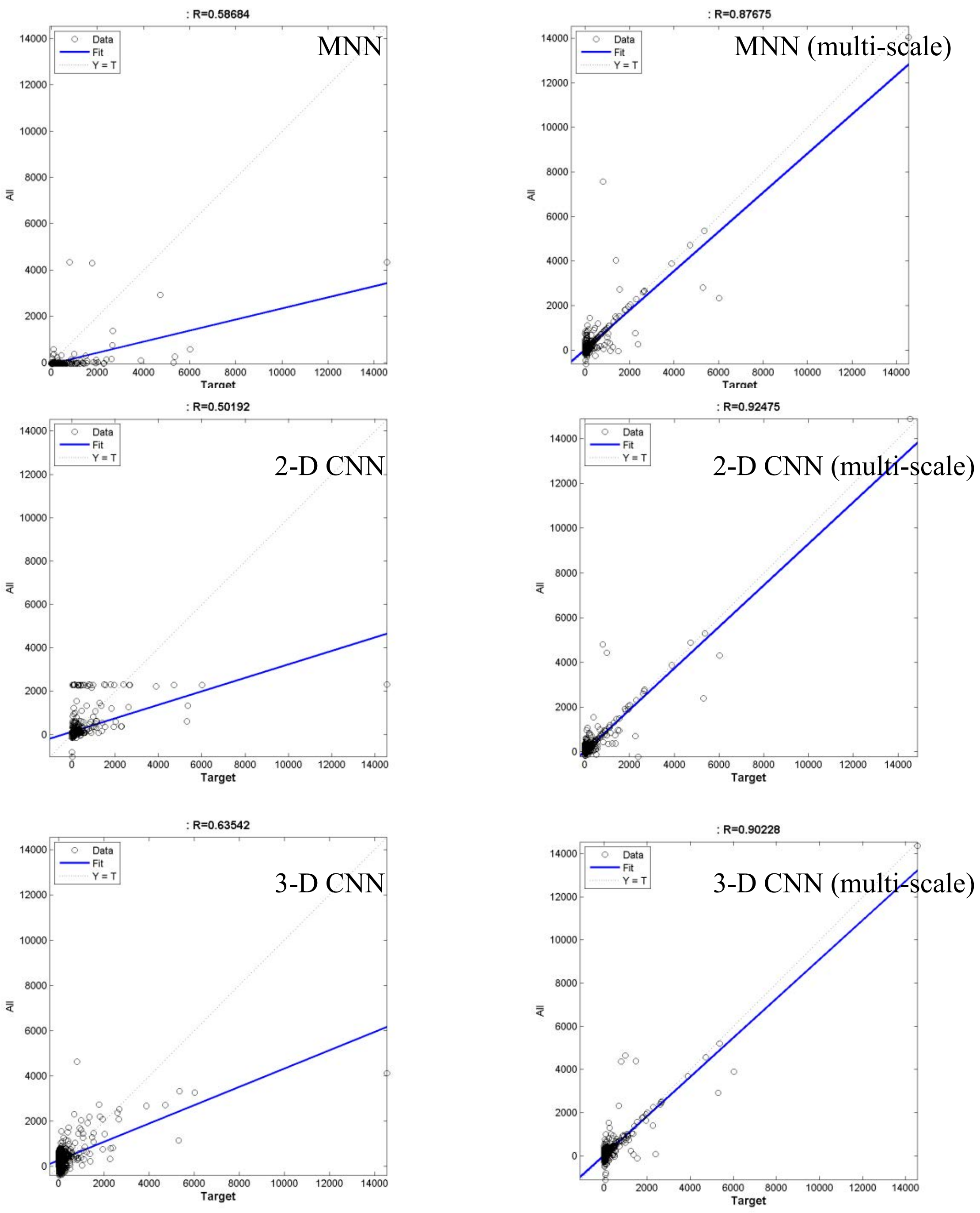


Figure 7-5: Regression plots of the predicted data on the y axis versus the target on the x axis for each model. This data is from the feed-forward network. The multi-scale models have better regression since they are near a 45 degree line.

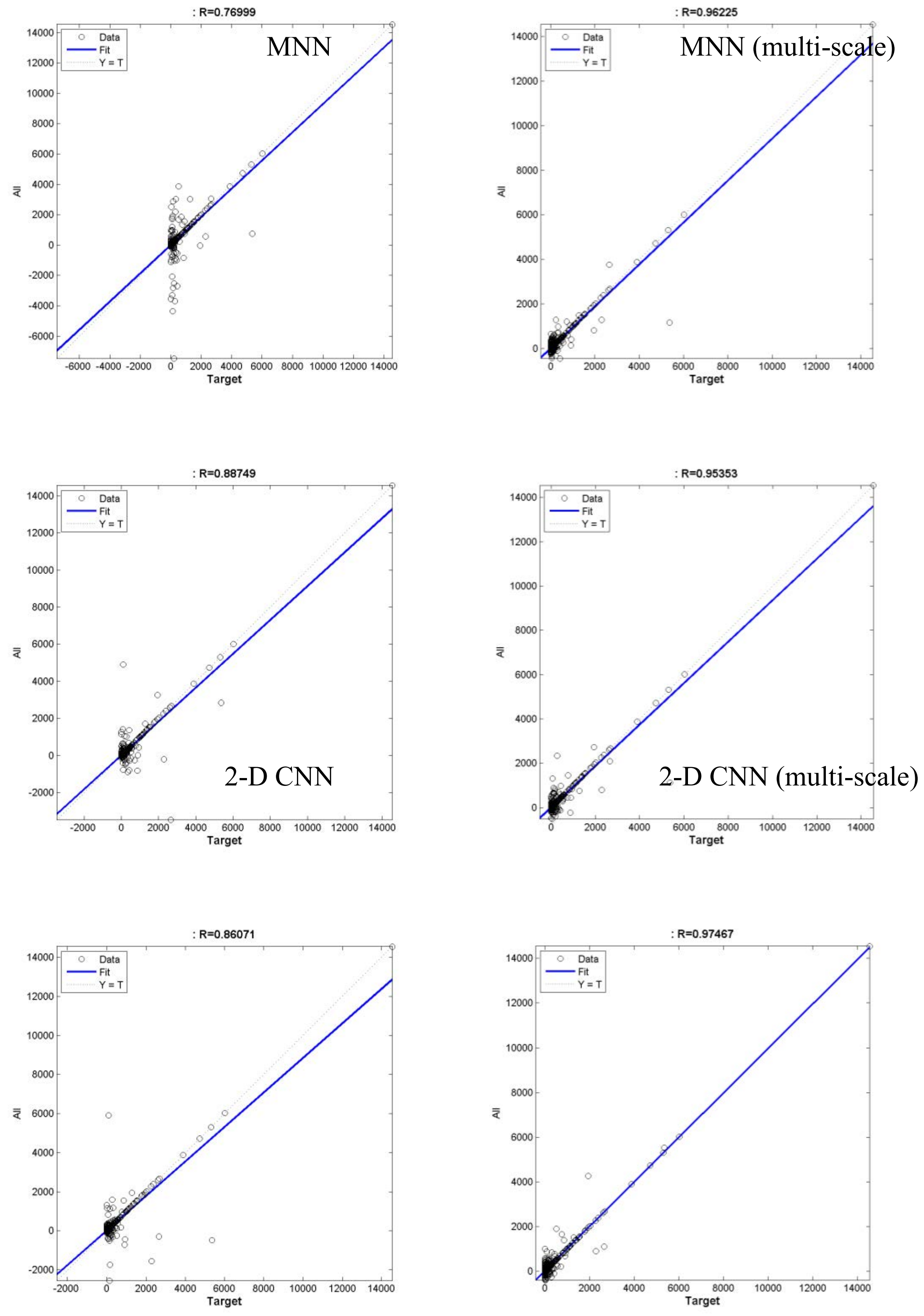


Figure 7-6: Regression plots of the predicted data on the y axis versus the target on the x axis for each model. This data is from the Bayesian regularization network. The multi-scale models have better regression since they are near a 45 degree line.

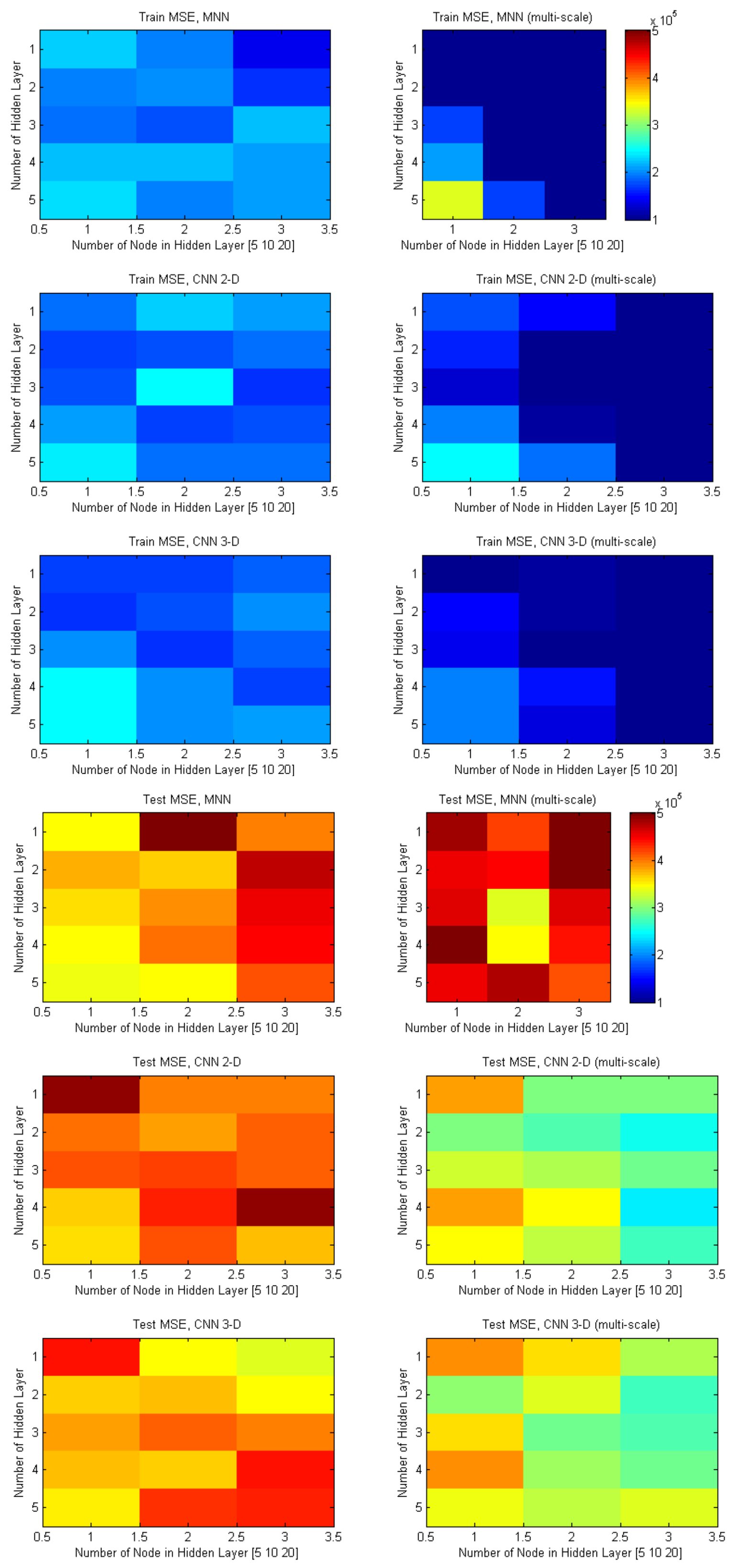


Figure 7-7: Mean square error of each network architecture from Training set (left) and Test set (right). The y axis is the number of hidden layers from 1 to 5, and the x axis is the number of nodes, where 1, 2, 3 correspond to 5, 10, 20 number of nodes.

Chapter 8

Computation of Grain Size Distribution in 2-D and 3-D binary images

Grain Size Distribution is one of the basic measurements for sediment classification. The conventional methods for grain size distribution include the sieve method, the laser diffraction method, and the point-count method. We aimed to develop a robust computer code that simulates these conventional methods. The code can measure grain size distribution on 2-D and 3-D binary images using a watershed algorithm to extract out individual grains, and using principal component algorithms to find the principal axes. The outputs include grain radius for different principal axes, grain volume, grain surface area, principal axes inclinations and azimuths, and the number of contacts for each grain. The calculated distribution can be volume-based, frequency-based, or grid-based. Digital microstructures used in this study include (1) identical sphere packs including a simple cubic pack and a Finney pack, and (2) natural rock geometry such as Berea sandstone, Castlegate sandstone, and Fontainebleau sandstone. Furthermore, we employed this code to provide additional value of information for the μXCT images by using μXCT to create

2-D to 3-D model of the grain size distribution, solving what is commonly known as Wicksell's corpuscle problem. We showed that our workflow successfully models a generalized 2-D to 3-D grain size distribution for a particular set of natural rocks we include in our study. We hope to be able to obtain more μXCT images in the future in order to create a universal model covering most types of natural rocks.

8.1 INTRODUCTION

The aim of this study was to develop a robust code to digitally measure grain size distribution on a 2-D or 3-D image. We also aimed to establish the workflow to estimate the grain size distribution from 2-D thin sections through Wicksell's corpuscle modeling on the μXCT images. Furthermore, we improved the precision of the method by incorporating principal component analysis to find the eigenvector of grain orientation. This method enables us to extract more information from the digital image about the grain size distribution such as the grain volume, grain surface area, grain principal axes inclination and azimuth, and the coordination number.

Grain size distribution is the basic measurement needed for sediment classification. It is defined as the distribution of the grains' diameter and their sorting. The grain size distribution can reveal information about the deposition process and sediment sources (Visher, 1969). For low energy depositional environments, the distribution becomes wider, signifying poorer sorting. For high energy depositional environments, the distribution becomes narrower and the grain size tends to be larger (Guéguen and Palciauskas, 1994). Well-known grain size classifications include Wentworth (1922) and Friedman and Sanders (1978). The grain size distribution has a wide range from 2 μm in clay to 2048 mm in cobbles.

Conventionally, grain size distribution is measured using the sieve method, the laser diffraction method, or the point-count method. The sieve method defines a grain diameter as the grain passes through a square hole in a sieve. Then, the volume of grains is measured. Laser diffraction, on the other hand, is based on forward scattering of monochromatic coherent light. This method is more accurate and repeatable (Konert and Vandenberghe, 1997). Our code can simulate these two methods of measurement by calculating volume-based distribution to simulate the sieve method and by calculating frequency-based distribution to simulate the laser diffraction method.

Alternatively, the grain size distribution can be measured by the point-count method using the 2-D thin sections. This can be done by laying out a grid on the 2-D thin sections. Then, a geologist measures the diameter of the grains at the grid intersections. The accuracy of this method is compromised by its bias toward large grains since the large grains have a higher probability of being on the grid intersections than small grains. We also add the capability in our code to simulate the point-count method by creating a specified grid and extract the grain size distribution only from grains that are on the grid intersection.

The estimation of size distribution from lower dimensional sampling probes is one of the classical problems in stereology known as Wicksell's corpuscle problem (Wicksell, 1925; Exner, 1972; Cruz-Orive, 1983). Ohser and Sandau (2000) presented an insightful summary of the problem. They pointed out that Wicksell's corpuscle problem is an ill-posed inverse problem and approached this problem using an EM algorithm (Expectation-Maximization) following Silverman et al. (1990). In this study, instead of solving an inverse problem, we set up forward modeling and solve it as linear least squares problems.

8.2 DIGITAL MICROSTRUCTURES

We studied the grain size distribution of two types of digital microstructures: (1) sphere packs for algorithm validation purposes, including a simple cubic pack (SCP), and a Finney pack, and (2) natural rocks including Berea sandstone, Castlegate sandstone, and Fontainebleau sandstone. Artificial and physical packs were created to validate the grain size distribution algorithm since their attributes are known. For natural rocks, we also have the scanned 2-D thin sections, which are in the RGB domain and have a resolution of 0.4 µm. Figure 8-1 shows the 3-D digital microstructures used in this study.

1. Simple cubic pack (SCP)

The SCP represents the loosest arrangement of sphere packs, with a porosity of 0.4764. The image is 500x500x500 voxels, with a voxel edge length of 2 µm. The SCP image contains 8^3 unit lattices; therefore, each sphere has a diameter of 62 voxels, which is equivalent to 0.124 mm.

2. Finney pack

The Finney pack is a physical random close packing of identical spheres (Finney, 1970). It is often considered a bridge between artificial sphere packs and a variety of natural rocks. The location of each sphere in the Finney pack was digitally rendered in a 3-D Cartesian coordinate system. The image is 500x500x500 voxels, with a voxel edge length of 2 µm. Each sphere in the Finney pack we created has a diameter of 82 voxels, which is equivalent to 0.164 mm.

3. Berea sandstone (Volume: B1 and B5)

The Berea sandstone is moderately well sorted, sub-angular to sub-rounded Mississippian sandstone, with a mean grain size of approximately 240 μm from the laboratory measurement. The segmented μXCT image is 1024x1024x1024 voxels, with a voxel edge length of 2.114 μm.

4. Castlegate sandstone (Volume: CG1)

The Castlegate sandstone is moderately sorted, sub-angular to sub-rounded Mesozoic sandstone, with a mean grain size of approximately 220 μm from the laboratory measurement. The segmented μXCT image is 1024x1024x1024 voxels, with a voxel edge length of 2.114 μm.

5. Fontainebleau sandstone (Volume: FB24)

The Fontainebleau sandstone is moderately well sorted, sub-rounded to rounded, with cementation of approximately 20%. The segmented μXCT image is 1024x1024x1024 voxels, with a voxel edge length of 2.072 μm.

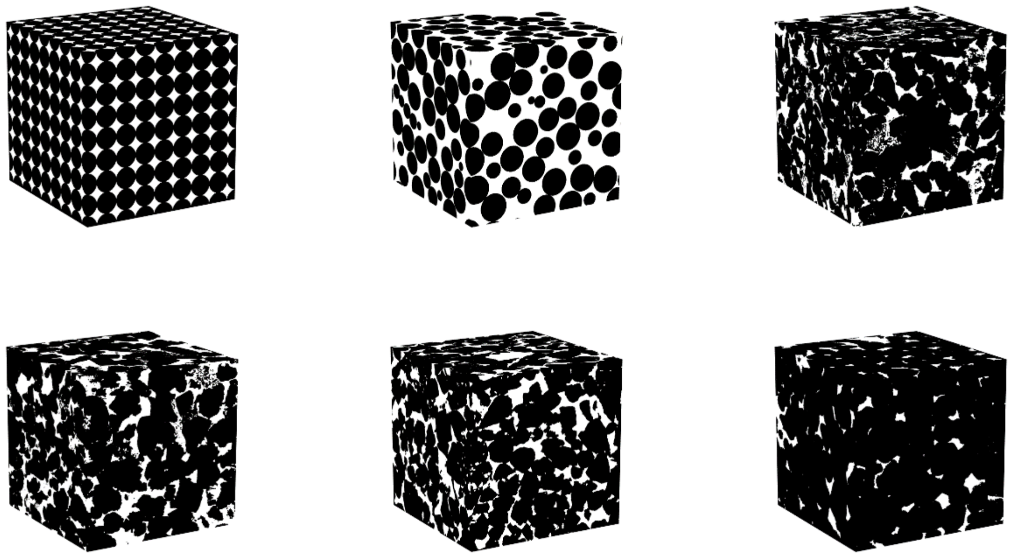


Figure 8-1: 3-D μXCT images in this study include a simple cubic pack (SCP), a Finney pack, Berea sandstone (B1 and B5), Castlegate sandstone (CG1), and Fontainbleau sandstone (FB24), from top-left to bottom-right respectively.

8.3 METHODS

In this section, we discuss our algorithm for grain size distribution measurement, the workflow on 2-D thin sections segmentation, and 2-D to 3-D grain size distribution modeling (Wicksell's corpuscle problem).

8.3.1 Grain size distribution

We developed MATLAB functions that can measure different grain properties from 2-D or 3-D binary segmented images (computeGSD.m). The analysis of grain properties involves four major steps (1) employing a watershed algorithm to draw the boundary between grains, (2) employing principal component analysis (PCA) to find the principal axes unit vectors (eigenvectors) of the grains and extend these vectors to find the grain size, (3) dilating the image of each grain to detect any contact with other grains, and (4) removing grains at the boundary of the image.

For the first step, we employed MATLAB's built-in watershed algorithm (watershed.m). To do so, we first created the scalar distance image by calculating the Euclidean distance from each solid voxel to its nearest pore voxel. We then employed the H-minima transform (imhmin.m) to prevent the image from being oversegmented by the watershed algorithm. The H-minima transform uses 8 connected neighborhoods for 2-D images and 26 connected neighborhoods for 3-D images. In the distance image, the algorithm suppresses all minima whose depth is less than the specified threshold. We set the default minima suppression at 3 voxels. The watershed algorithm finds the ridge between each pair of local minima in the distance image.

For the second step, we employed principal component analysis (PCA) to find the grain sizes along the grains' principal axes. We measured the size of grains on two perpendicular axes (a total of 4 radii: $(r_1\|r_2)\perp(r_3\|r_4)$) for a 2D image and on three perpendicular axes (a total of 6 radii: $(r_1\|r_2)\perp(r_3\|r_4)\perp(r_5\|r_6)$) for a 3D image Figure 8-2. Apart from the grain sizes, we also calculated grains' orientations. For a 2D image, we calculated how far the azimuth of the principal axes deviated from the y-axis. For a 3D image, we calculated the azimuth and inclination of the principal axes using spherical coordinates following the physics ISO convention.

The radius r is given by

$$r = \sqrt{x^2 + y^2 + z^2}.$$

The inclination from the z-axis (θ) is

$$\theta = \arctan\left(\frac{y}{x}\right).$$

The azimuthal angle from the x-axis in the counterclockwise direction is

$$\varphi = \arccos\left(\frac{z}{r}\right).$$

For the third step, we dilated each grain by two pixels on the watershed image and extracted the indices on each grain boundary. The number of unique indices on each grain boundary is the number of contacts per grain (i.e. coordination number).

After measuring the different properties of grains, the last step is to impose boundary conditions by removing grains at the boundary. We detected grains at the image boundaries by using the shortest perpendicular distance from the grain centroid to the boundary. If this distance was smaller than the radius of the grain, then the grain was at the boundary.

The code outputs (1) grain centroid - a matrix of Cartesian coordinates for each grain, (2) grain radius – a matrix of the radius measured from the grain centroid for each grain, (3) grain azimuth, - a matrix of the azimuth of all principal axes, (4) grain inclination – a matrix of the inclination of all principal axes, (5) grain volume – a vector of the volume in voxel for each grain, and (6) the number of contacts - a vector of the number of adjacent grains in contact. The code can also output the plot for the quality control process for the 2-D grain size distribution measurement. Figure 8-3 shows the example output for a 2-D slice for natural rocks. The green circles denote the grains that are included in the distribution since no part of them touches the edges of the image.

For 3-D μXCT images, we subdivided the image into the size 500^3 voxels for the faster running time of the watershed algorithm. After all of the simulations, the grain sizes are added to form the single distribution.

The grain size distribution can be plotted in volume-based (computeHistVB.m) to simulate the sieve method, in frequency-based (computeHistFB.m) to simulate the laser diffraction method, or in point-count method (computeHistPC.m). The volume-based calculation was done by adding up the grain volume in each bin as the volume fraction of the distribution.

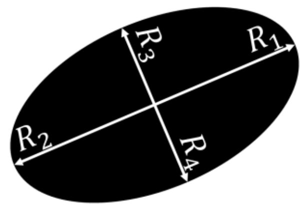


Figure 8-2: Principal Component Analysis (PCA) to find the grain sizes along principal axes. We measured the size of grains on two perpendicular axes (4 radii) for a 2-D image and three perpendicular axes (6 radii) for a 3-D image.

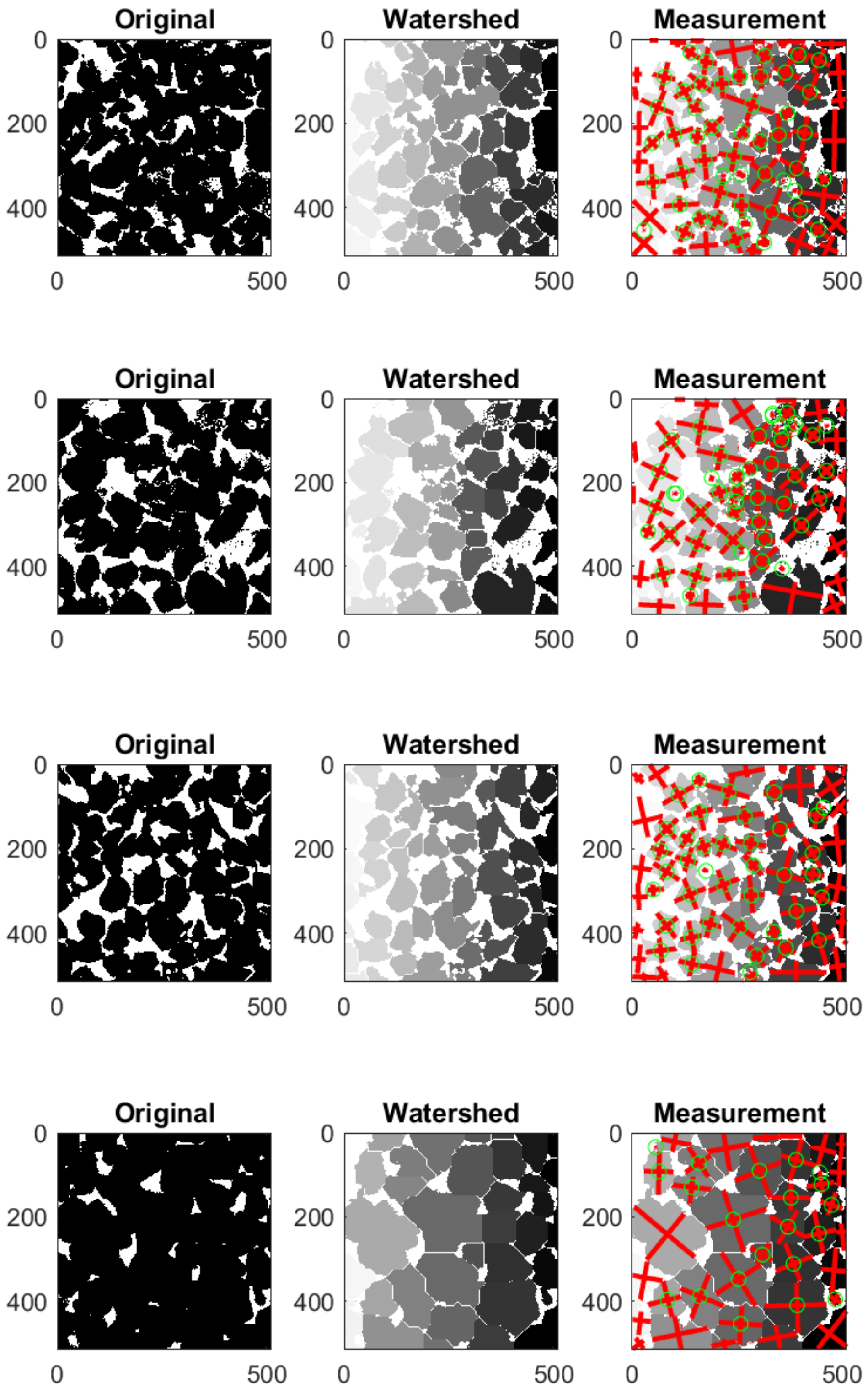


Figure 8-3: The plot for quality control in the grain size measurement in a 2-D slice of B1, B5, CG1, and FB24. The scale is in voxels.

8.3.2 2-D Thin Sections Segmentation

The scanned 2-D thin sections are in the RGB domain and have the resolution of 0.65µm. For each 2-D thin section, we subdivided the image into 5000x5000 pixels for the purpose of parallel computing (Figure 8-4). The chosen size of the subdivision was ensured to be larger than the largest grains in the 2-D thin sections. We then controlled the quality of the measurements by removing any subimages that contained any stains in the physical 2-D thin sections.

The K-Means segmentation was performed to transform the 2-D thin sections in the RGB domain into segmented binary images containing values 1 for solid and 0 for pores. K-Means segmentation is a well-known method for data clustering in various fields (Jain, 2010).

For each of the subimages, we randomly selected 10,000 pixels to form a training set for K-Means segmentation. Adding up all the subimages, we had more than 2,100,000 pixels for the training set. Each pixel contained the set of values between 0-255 for each channel in the RGB domain. Then, we selected only a unique set of values to prevent bias toward more common colors in the 2-D thin sections. Figure 8-5 and Figure 8-6 show 2-D thin sections before and after the segmentation process.

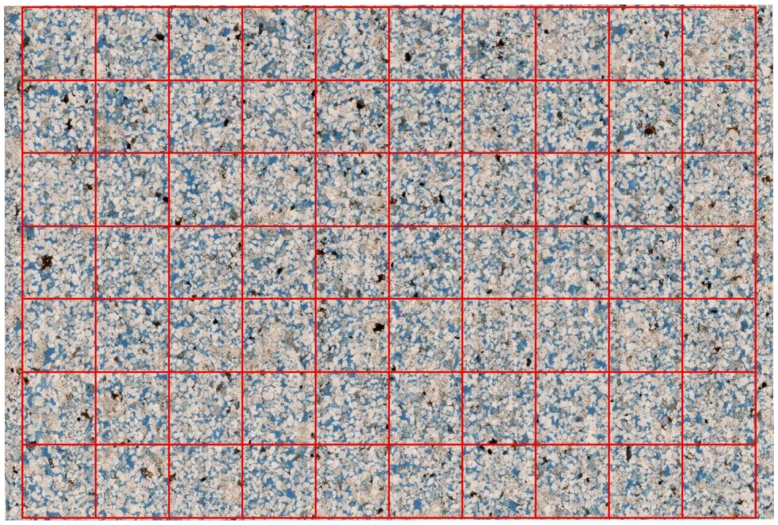


Figure 8-4: Example of a 2-D thin section with subdivision grid. The size of the each subimage is 5000x5000 pixels.

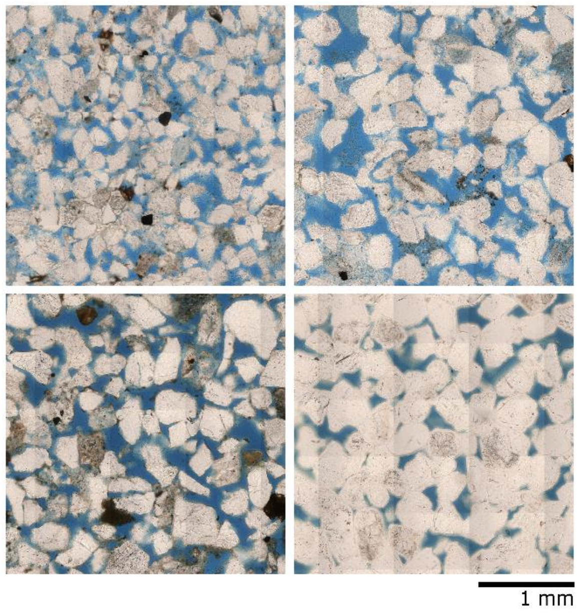


Figure 8-5: Cropped 2-D thin sections from B1, B5, CG1, and FB24 from top-left to bottom-right.

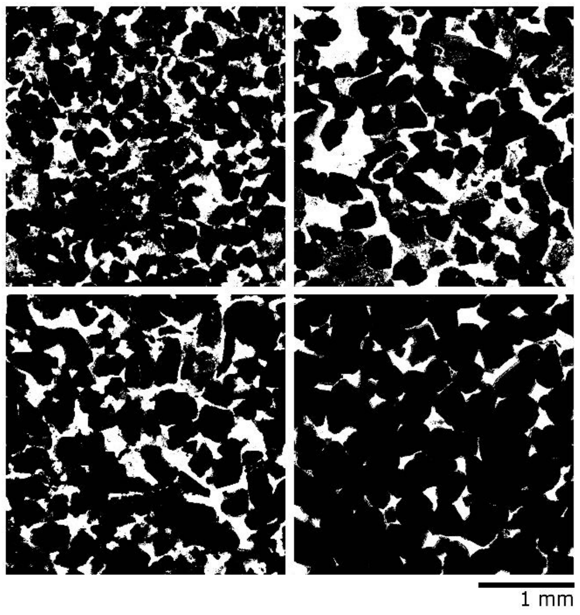


Figure 8-6: Cropped segmented 2-D thin sections from B1, B5, CG1, and FB24 from top-left to bottom-right before image cleaning.

8.3.3 2-D to 3-D grain size distribution modeling (Wicksell's corpuscle problem)

Wicksell's corpuscle modeling is a classic stereological problem addressing the estimation of the size distribution using a lower dimensional sampling (Stoyan et al., 1995; Ohser and Sandau, 2000). Finding the 3-D grain size distribution from a 2-D image is a challenging problem since a 2-D image alone does not contain the complete information on the distribution. For instance, a 2-D slice of closely packed equal spheres shows isolated spheres unless the slice is cut right where the spheres are in contact. To solve this problem, we modelled the 2-D to 3-D grain size distribution as a forward modeling problem. Since we can obtain both 3-D and 2-D histogram vectors from a μXCT image, we can use the μXCT image to understand better the relationship between the 2-D and 3-D grain size distribution by constructing the transform matrix. First, we sorted the grain size into the histogram using the same bin size for the 2-D and 3-D histogram (computeGSDHist.m). We set up the bin size in the log 10 scale with a total of 49 bins. Given the 2-D histogram vector x and 3-D histogram vector y, we related these two vectors through the transform matrix T as $y = Tx$, or in the matrix form

$$\begin{bmatrix} y_1 \\ \cdot \\ \cdot \\ \cdot \\ y_n \end{bmatrix} = \begin{bmatrix} t_{11} & \cdot & \cdot & \cdot & t_{1n} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ tn_1 & \cdot & \cdot & \cdot & t_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$$

For each µXCT image, there is a single 3-D histogram vector y and there are multiple sets of 2-D histogram vectors x. We assumed that all of the 2-D histogram vectors x contribute to each bin in the 3-D histogram vector independently as follows:

$$y_i = \sum_{k=1}^{n} t_{ik}\, x_k.$$

Then we combined the grain size distribution from different 2-D slices and reformed the problem for the histogram bin i of the 3-D grain size distribution histogram as

$$\begin{bmatrix} y_i \\ \cdot \\ \cdot \\ \cdot \\ y_i. \end{bmatrix} = \begin{bmatrix} x_1^{slice\ 1} & \cdot & \cdot & \cdot & x_n^{slice\ 1} \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ \cdot & & & & \cdot \\ x_1^{slice\ m} & \cdot & \cdot & \cdot & x_n^{slice\ m} \end{bmatrix} \begin{bmatrix} t_{i1} \\ \cdot \\ \cdot \\ \cdot \\ t_{in} \end{bmatrix}$$

This system of equations can be solved for the transform matrix of row i through the linear least-square method (lsqnonneg.m). The process is repeated until all the coefficients in the transform matrix are solved. Furthermore, we added 2-D and 3-D grain size distribution from other µXCT images to create a more generalized transform matrix. For the 3-D grain size distribution prediction from 2-D thin sections, we created the generalized transform matrix for all natural rocks: B1, B5, CG1, and FB24.

8.4 RESULTS AND DISCUSSIONS

8.4.1 Grain Size Distribution

The grain size distribution code can be validated by using artificial and physical sphere packs since their properties are known. The results show that the grain sizes from the 3-D μXCT images are constant, which is consistent with their geometry since they are packs with equal-sized spheres (Figure 8-7). For sphere packs with identical spheres, there is no difference between calculating the grain size distribution using the volume-based method, the frequency-based method, or the point-count method since all spheres have the same size. For the grain size distribution from 2-D slices of the 3-D μXCT images, Figure 8-7 shows non-representative grain sizes when the slices are not at the center of the grains.

Figure 8-8 shows that the program reports the coordination number (number of contacts per grain). The coordination number for a sample cubic pack is 6 which is equal to its theoretical number. The correlation number from the Finney ranges from 2 to 6 in our study. Figure 8-9 shows the coordination number for natural rocks computed from our code. The coordination numbers of natural rocks ranges from 1 to 20.

For natural rocks, Figure 8-10 shows 3-D grain distribution using different methods of calculating the distribution (volume-based, frequency-based, and point-count). We observe that the small grains especially with diameters in the silt range are not represented well in the volume-based method since their grains are small and do not have weight in the distribution. However, it is also possible that these small grains really simply noise in the digital images as well since their diameters are approximately 3 pixels. The most surprising results are from the point-count grain size distributions. The point-count grid detects many small grains. We previously expected that this method would have the higher mean than other methods since there is higher probability that the grid will fall on the larger grains.

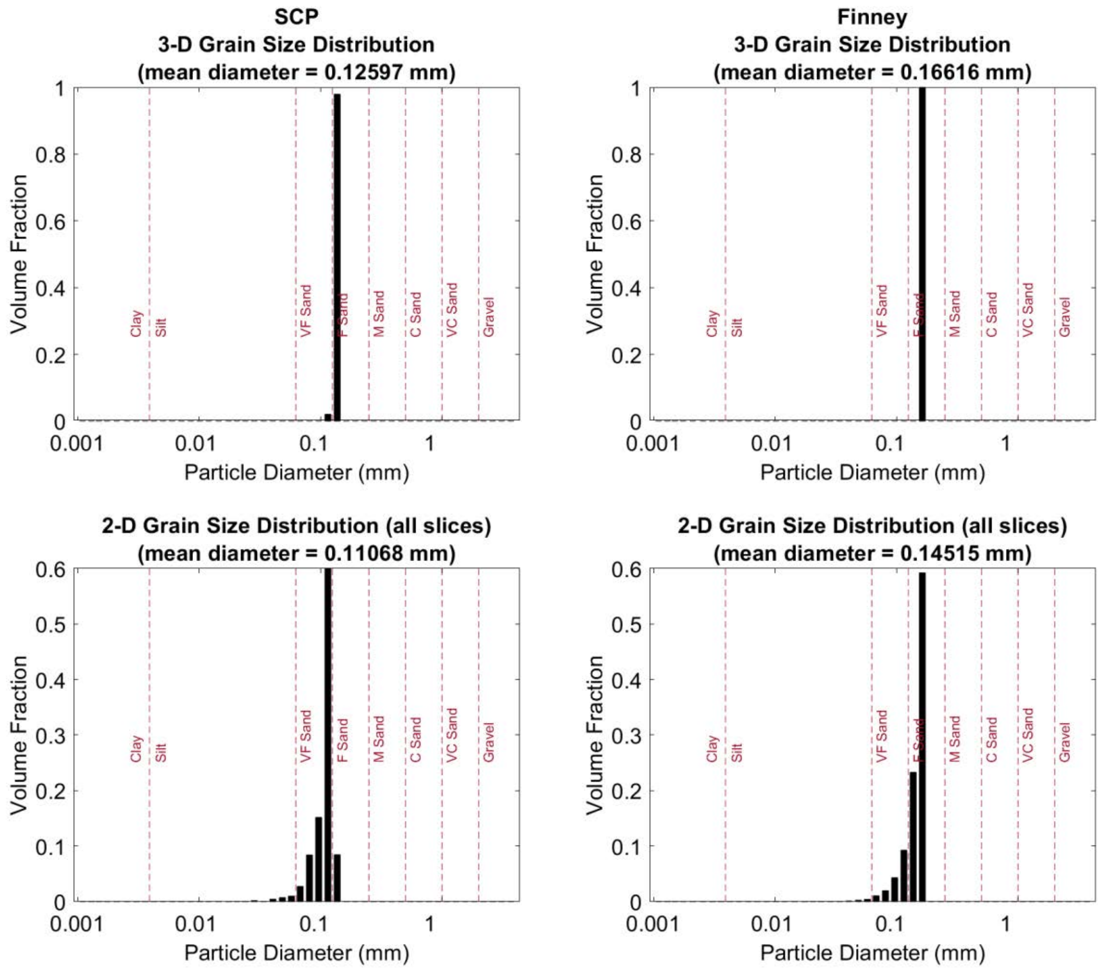


Figure 8-7: 3-D and 2-D Grain size distributions from the μXCT images of a simple cubic pack and a Finney pack for algorithm validation. For these sphere packs, volume-based and frequency-based distribution show the same results since they are packs of identical spheres. 2-D grain size distribution clearly shows how 2-D slicing would result in apparent grain size measurements.

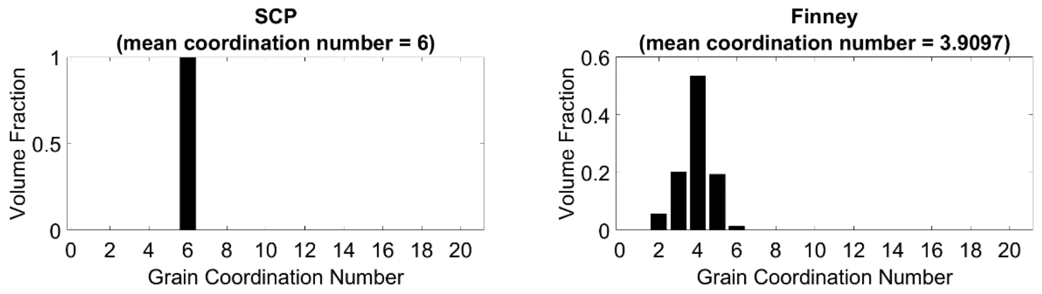


Figure 8-8: The coordination number (number of contacts per grain) from 3-D μXCT images of a simple cubic pack and a Finney pack. The simple cubic pack shows the coordination number of 6, which is equal to the theoretical number.

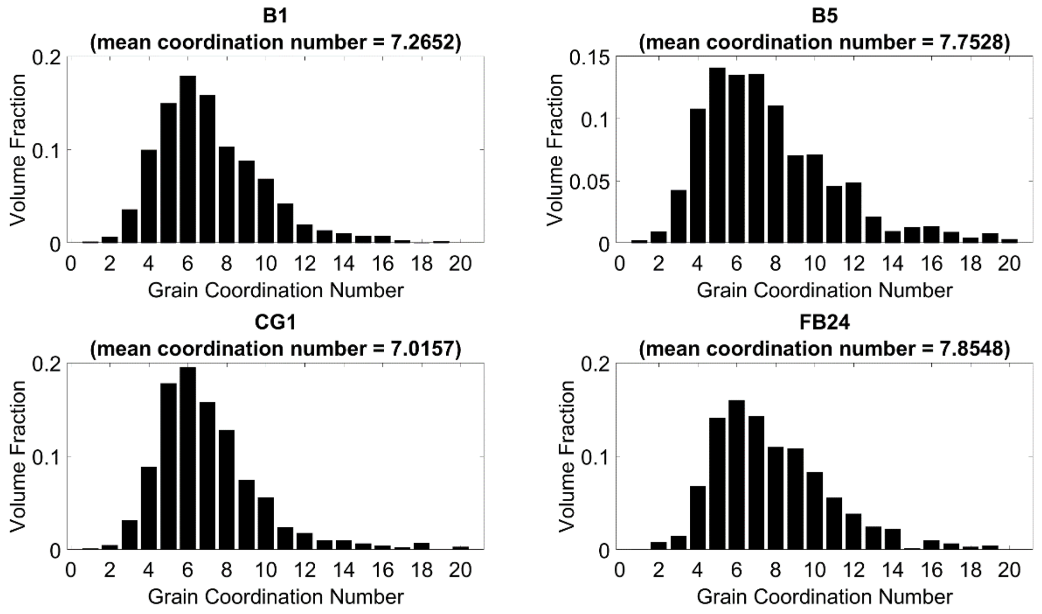


Figure 8-9: The coordination number (number of contacts per grain) from 3-D μXCT images of natural rocks.

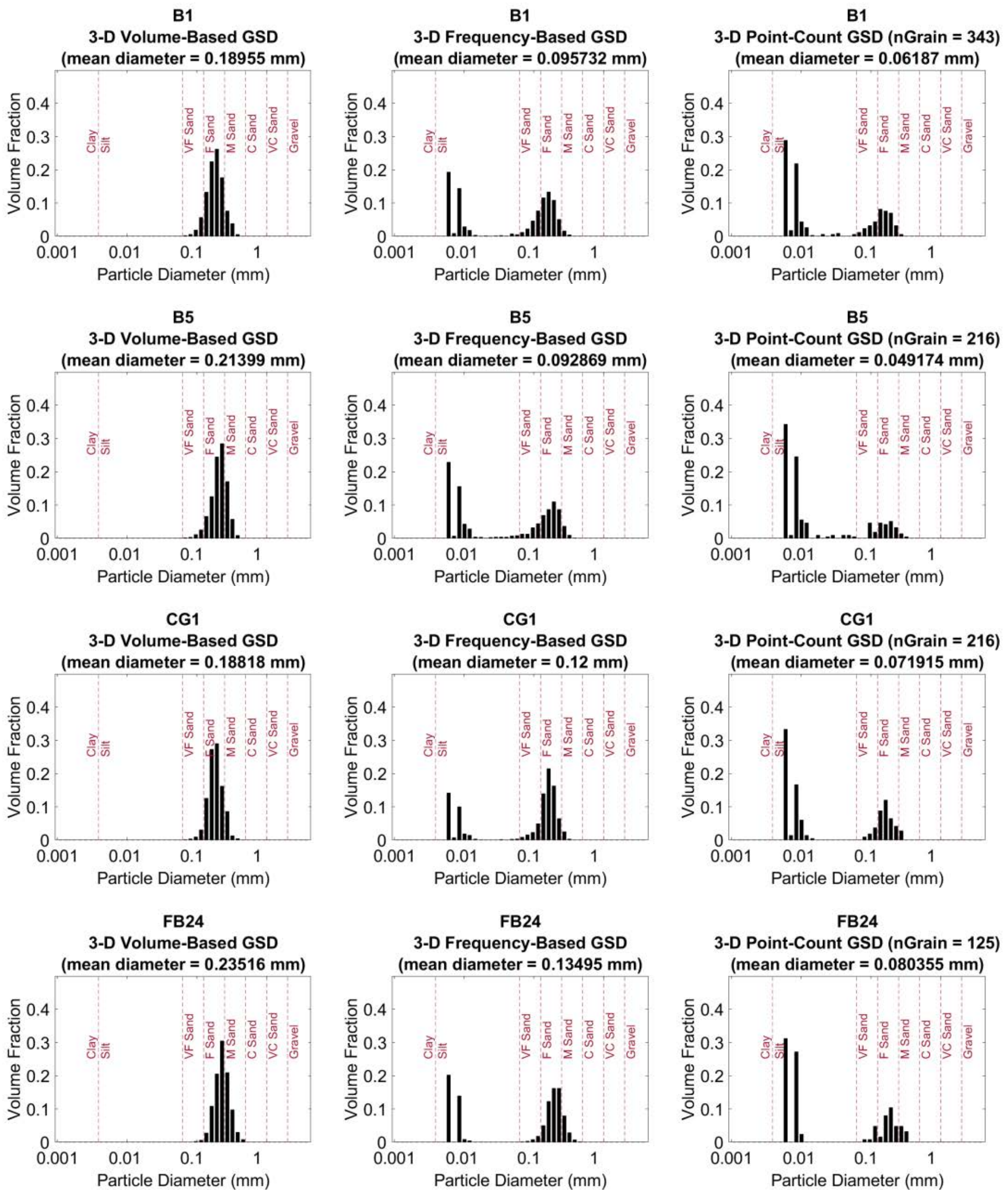


Figure 8-10: 3-D grain-size distributions with (1) volume-based (simulating the sieve method), (2) frequency-based (simulating the laser diffraction method), (3) point-count (simulating the point-count method on a 2-D thin section).

8.4.2 2-D to 3-D grain size distribution modeling (Wicksell's corpuscle problem)

We compared the predicted 3-D grain size distributions from thin sections to the 3-D distributions from μXCT images in Figure 8-11 and Figure 8-12. We observed that the mean diameters from 2-D thin sections are similar to laboratory measurement. However, these numbers are all larger than the mean diameters from μXCT images. This could result from the fact that μXCT images have smaller field of view than thin sections. We created a transform matrix from all natural rocks in this study and still obtained a strong correlation between the prediction and the target. However, when we investigated the transform matrix, we found many missing coefficients that may require more μXCT images to model.

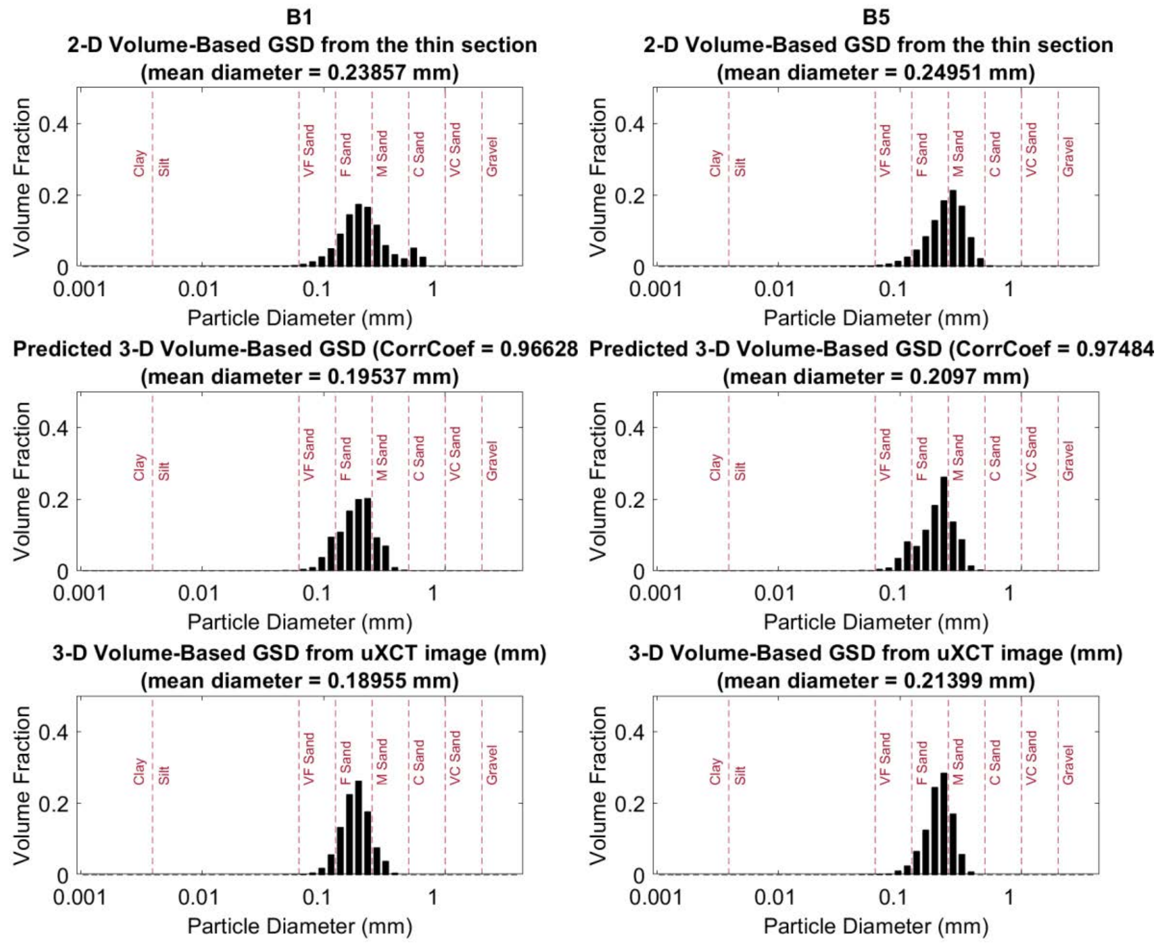


Figure 8-11: The comparison between 2-D volume-based grain size distribution from the thin section, predicted 3-D volume-based grain size distribution, and the 3-D volume-based grain size distribution from uXCT image for B1 and B5.

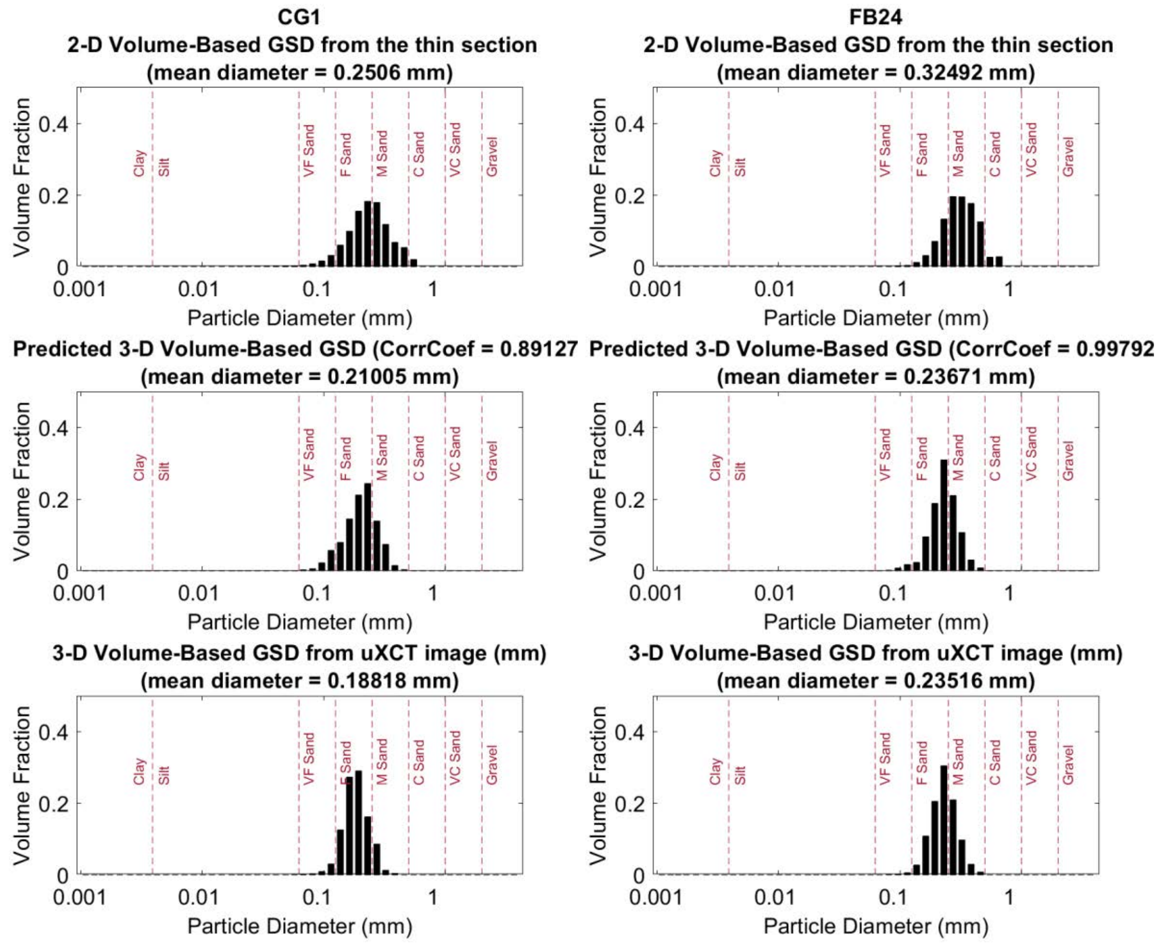


Figure 8-12: The comparison between 2-D volume-based grain size distribution from the thin section, predicted 3-D volume-based grain size distribution, and the 3-D volume-based grain size distribution from uXCT image for CG1 and FB24.

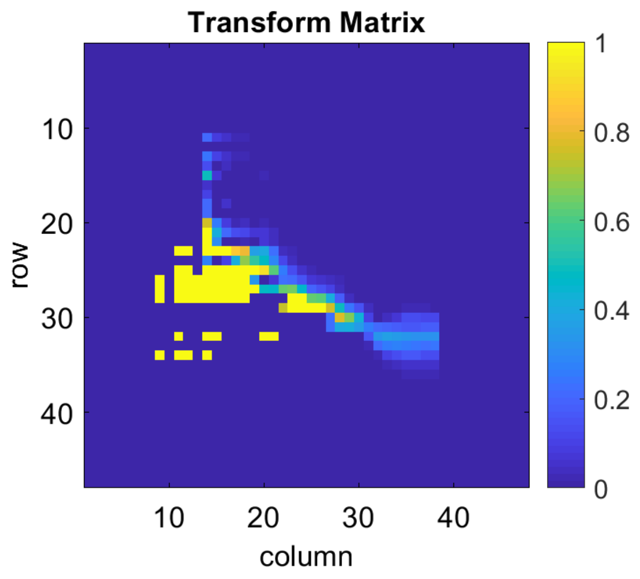


Figure 8-13: 2-D to 3-D grain size distribution transform matrix. The color yellow shows the coefficient within the transform matrix that is greater than one.

8.5 CONCLUSION

The grain size distribution code is a robust method to find the grain size distribution from either 2-D or 3-D binary images. We validated the code using identical sphere packs. Using the code can reduce the time and increase the accuracy of acquiring the grain size distribution. However, there are still challenges for extracting the grain size distribution from digital images. These challenges include the segmentation process from 2-D thin sections and the image processing steps to remove noise. Our code can also provide the additional value of information to the μXCT images if they are used to create 2-D to 3-D grain size distribution model. We showed that our workflow successfully models a generalized 2-D to 3-D grain size distribution for a particular set of natural rocks we include in our study. We hope to be able to obtain more μXCT images in the future in order to create a universal model covering most types of natural rocks.

Appendix A

Digital Microstructures

Digital microstructures used in this dissertation except ones in Chapter 8 include (1) pipes of various cross-sections, (2) artificial and physical sphere packs, including simple cubic packs (SCP), face-centered cubic packs, and Finney packs, and (3) natural rocks including Fontainebleau sandstone, Bituminous sands, Berea sandstones, and Grosmont carbonates.

A.1 PIPES OF VARIOUS CROSS-SECTIONS

We generated 3-D segmented binary image cubes containing straight pipes with different cross-sections, parallel to the x-direction. All images are 200x100x100 voxels, with a voxel edge length of 0.002 mm. We specified the center at y0 = 50 and z0 = 50. The pipes of various cross-sections in this study are as follows (Figure A-1):

A.1.1 Round pipes

We created a total of 9 realizations of straight round pipes with different radii (r) ranging from 4 to 36 pixels with increments of 4 pixels. The equation for creating a round pipe is $(y - y_0)^2 + (z - z_0)^2 < r^2$.

A.1.2 Elliptical pipes

We created a total of 9 realizations of elliptical pipes whose porosities match the porosities of round pipes. The aspect ratio (AR) of elliptical pipes is 0.8. The equation for an elliptical pipe is $\left(\frac{y-y_0}{a}\right)^2 + \left(\frac{z-z_0}{b}\right)^2 < r^2$. To match the porosity of a round pipe, we specified $a = \frac{r}{\sqrt{AR}}$ and $b = a * AR$.

A.1.3 Triangle pipes

We created a total of 9 realizations of equilateral triangle pipes whose porosities match the porosities of round pipes. The equations for an equilateral triangle pipe with a side length t is

(1) for $y < ct,\ abs(z - z_0) < \frac{\sqrt{3}t}{4}\ and\ z < \sqrt{3}\left(y - y_0 + \frac{t}{4}\right) + ct$ and

(2) for $y < ct,\ abs(z - z_0) < \frac{\sqrt{3}t}{4}\ and\ z < -\sqrt{3}\left(y - y_0 + \frac{t}{4}\right) + ct.$

To match the porosity of a round pipe, we specified $t = \sqrt{\frac{4\pi r^2}{\sqrt{3}}}$.

A.1.4 Square pipes

We created a total of 9 realizations of square pipes whose porosities match the porosities of round pipes. The equations for a square pipe with a side length a is

$$abs(y - y_0) < \frac{a}{2}, \text{ and}$$

$$abs(z - z_0) < \frac{a}{2} \ .$$

To match the porosity of a round pipe, we specified $a = \sqrt{\pi r^2}$.

A.1.5 Sinusoidal pipes

We created a total of 9 realizations of pipes with sinusoidally varying radius and different fractional changes in radius (δ). The equation for creating sinusoidal cross-sections is

$$(y - y_0)^2 + (z - z_0)^2 = r_s^2$$

, where $r_s = r_0\left(1 + \delta \cdot \sin\left(\frac{t}{2r_0}\right)\right)$ and $t = 0{:}\,2\pi$. We specified r_0 to be 20 pixels.

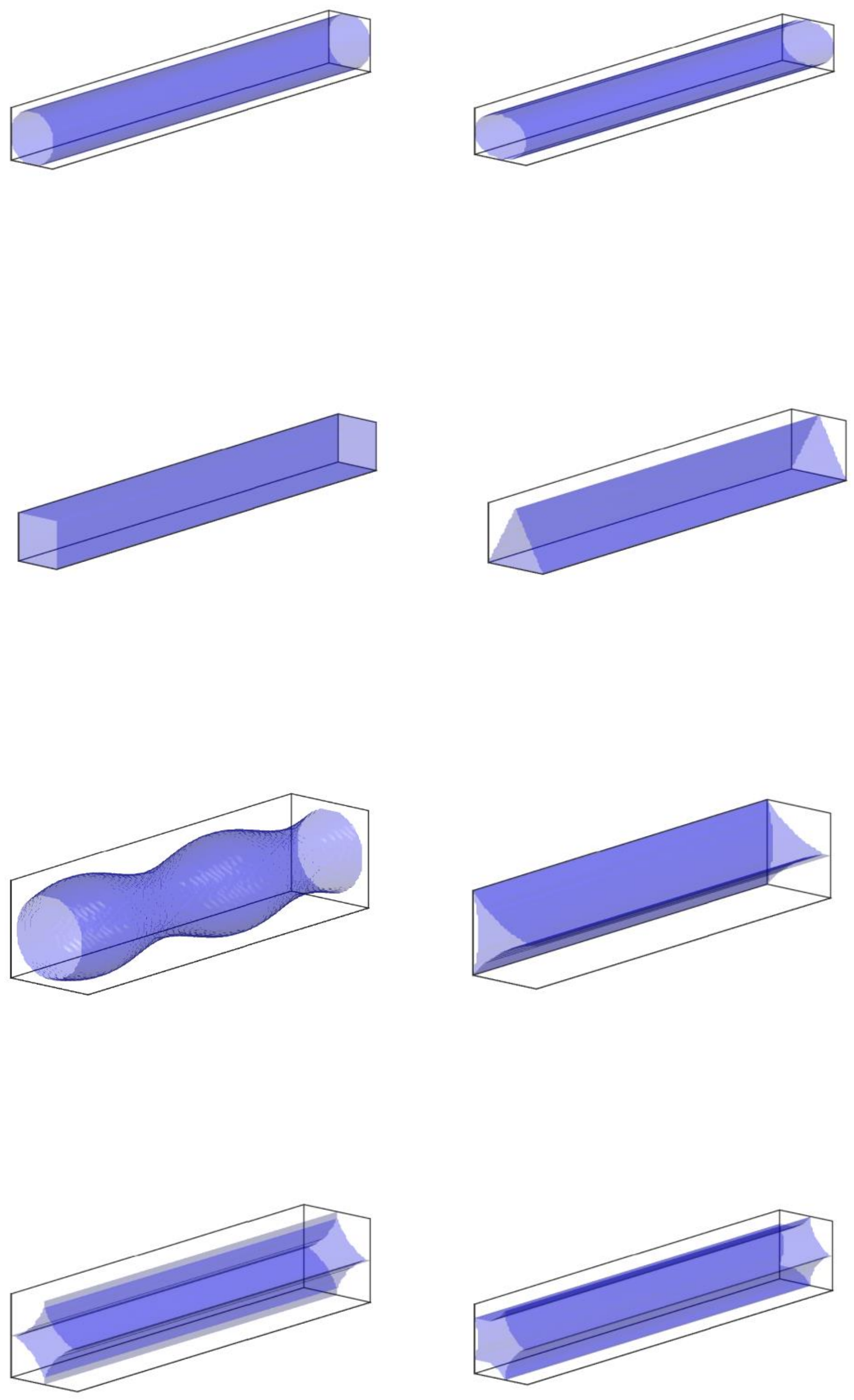
A.1.6 k-cusps hypotrochoidal pipes

We created k-cusps hypotrochoidal pipes with k ranges from 3 to 7. For each number of k, we created 9 realizations of k-cusps hypotrochoidal pipes whose porosities match the porosities of round pipes. Therefore, there is a total of 45 realizations. The equation for a k-cusps hypotrochoidal pipe is

$$y(\theta) = a \cdot \left((k-1)cos(\theta) + \cos\left((k-1)\theta\right)\right) + y_0$$

$$z(\theta) = a \cdot \left((k-1)sin(\theta) + \sin\left((k-1)\theta\right)\right) + z_0$$

, where $\theta = 0: 2\pi$. To match the porosity of a round pipe, we specified $= \frac{r}{\sqrt{2}}$.



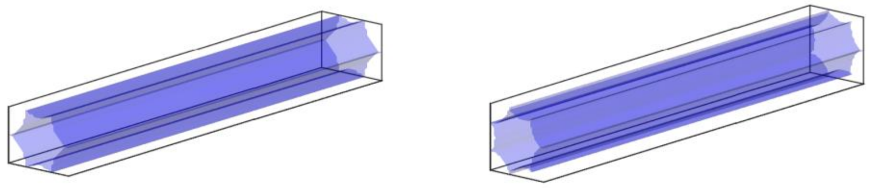


Figure A-1: Pipes with different cross-sections from left to right: round pipes, elliptical pipes, triangle pipes, square pipes, sinusoidal pipes, 3-cusps hypotrochoidal pipes, 4-cusps hypotrochoidal pipes, 5-cusps hypotrochoidal pipes, 6-cusps hypotrochoidal pipes, 7-cusps hypotrochoidal pipes.

A.2 ARTIFICIAL AND PHYSICAL SPHERE PACKS

For effective elastic and transport properties, the close packing of identical spheres has long been studied since it resembles realistic rock geometry. Graton and Fraser (1935) studied systematic packings of spheres to form the empirical equation of flow through the close packing. The close packing of identical spheres can be a common simple model of granular media for both effective elastic and transport properties. Spherical packs can also be created from granular dynamic simulations that fully reflect the compaction effect (Silin et al., 2004; Sain, 2011). In this dissertation, we generated 3-D segmented binary image cubes containing artificial and physical packings of spheres. All images are 200x200x200 voxels, with a voxel edge length of 12.5 μm. This voxel scale was chosen to make a sphere of radius 10 voxels equivalent to fine sand (0.125 mm radius) and a sphere of radius 40 voxels equivalent to coarse sand (0.5 mm radius). The artificial and physical packings in this study is as follows:

A.2.1 Simple cubic pack (SCP)

The SCP represents the loosest arrangement of sphere packs, with a porosity of 0.4764. We created a total of 8 realizations of SCP. The original image of a SCP contains 3^3 unit lattices; therefore, each sphere has a radius of 33 voxels. The other 7 realizations were created by dilation of grains in the original image by increments of 5% (Figure A-2). With grain dilation effects, the porosities of SCP range from 0.0685 to 0.4764.

A.2.2 Face-centered cubic pack (FCP)

The FCP represents the densest arrangement of sphere packs, with a porosity of 0.2595. We created a total of 8 realizations of a FCP. The original image of a FCP contains 2^3 unit lattices; therefore, each sphere has a radius of 35 voxels. The other 7 realizations were created by grain dilation at increments of 5% (Figure A-2). With grain dilation effects, the porosities of FCP range from 0.0001 to 0.2595.

A.2.3 Finney Pack

The Finney pack is a physical random close packing of identical spheres (Finney, 1970). The Finney pack consists of 4021 spheres, in which the location of each sphere was digitally rendered in a 3-D Cartesian coordinate system. The Finney pack acts as a bridge between artificial packing models and natural rocks, and it is used widely in computational experiments (Jin et al., 2009; Richa, 2010; Sain, 2011; Dvorkin et al., 2012). For this experiment, the Finney pack was also digitally altered by changing the radius of each sphere (LX = 3,6,9,12) (Figure A-3). For LX equals 3, each sphere has a radius of 40 voxels and is equivalent to coarse grain (0.5. For LX equals 12, each sphere has a radius of 10 voxels and it is equivalent to fine grain (0.125 mm). For each value of LX, we created 8 realizations of Finney packs with grain dilation at increments of 5%. Therefore, there are a total of 24 realizations.

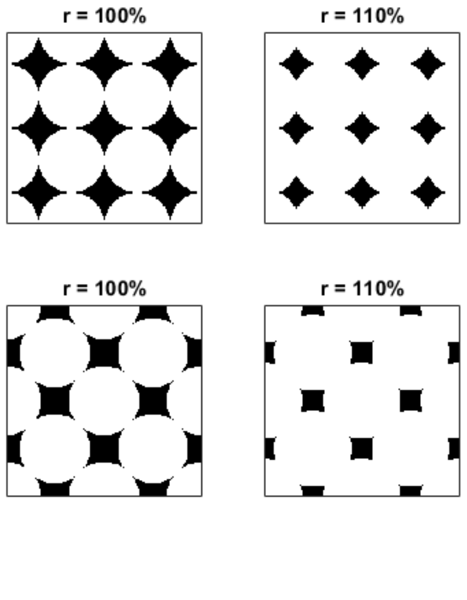


Figure A-2: Grain dilation effect on simple cubic pack (Top) and face-centered cubic pack (Bottom)

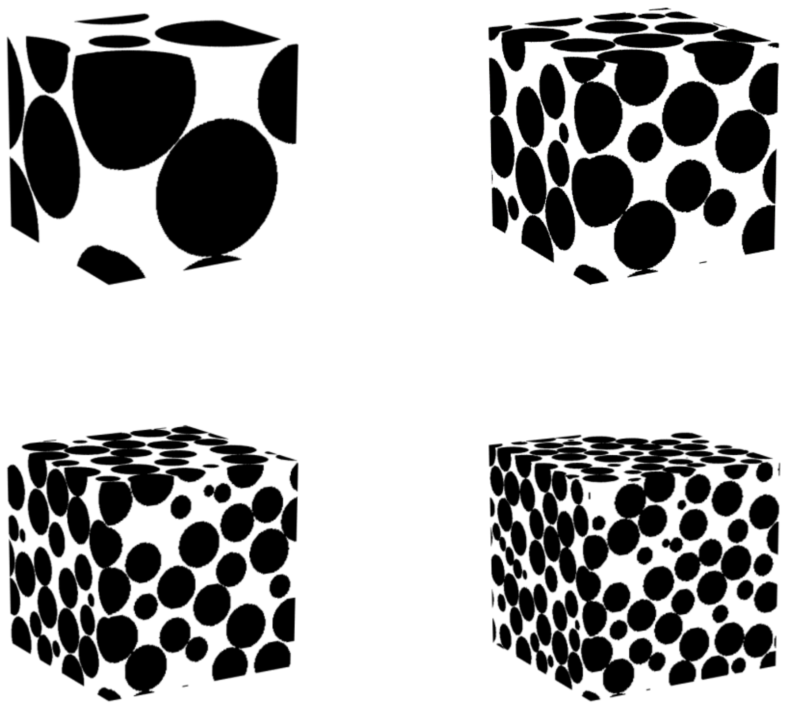


Figure A-3: Finney packs with different radii of spheres (LX = 3, 6, 9, 12 from left to right).

A.3 NATURAL ROCKS

For natural rocks, we subsampled all 3-D segmented binary images to the size 200x200x200 voxels in order to gain more samples and to test the variability of the tortuosity. The subsamples are large enough to reach the representative elementary volume since their calculated properties such as porosity and permeability are similar to these found in laboratory measurements. The voxel edge lengths of each subsample are the same as the voxel edge lengths of the original images. Natural rock samples in this study are as follows (Figure A-4):

A.3.1 Fontainebleau sandstone

The original size of Fontainebleau sandstone is 288x288x288 voxels, with a voxel edge length of 7.5 μm. We generated 8 subsamples of size 200x200x200 voxels. In this case, the subsamples are overlapped. For Fontainebleau sandstone, the laboratory measurements found a porosity of approximately 0.152, and a permeability of approximately 1100 mD (Andrä et al., 2013a). In comparison, the subsamples have an average porosity of 0.147 and a permeability range from 1541.3 mD (P10) to 2094.3 mD (P90), with a median (P50) of 1798.6 mD.

A.3.2 Bituminous sand

The original size of the bituminous sand image is 400x400x400 voxels, with a voxel edge length of 4 μm. We generated 8 subsamples of size 200x200x200 voxels. The subsamples have an average porosity of 0.368 and a permeability range from 6520.1 mD (P10) to 9300.7 mD (P90), with a median (P50) of 7428.2 mD. This sample was first studied in substitution of two phases in a three phase multimineralic rock (Saxena, 2014).

A.3.3 Berea sandstone

The original size of Berea sandstone is 1024x1024x1024 voxels, with a voxel edge length of 0.74 µm. We generated 125 subsamples of size 200x200x200 voxels. For Berea sandstone, the laboratory measurements found a porosity of approximately 0.20, and a permeability range from 200 to 500 mD (Andrä et al., 2013a). In comparison, the subsamples have an average porosity of 0.19 and a permeability range from 15.9 mD (P10) to 210.5 mD (P90), with a median (P50) of 70.2 mD.

A.3.4 Grosmont carbonate

The original size of Grosmont carbonate is 1024x1024x1024 voxels, with a voxel edge length of 2.02 µm. We generated 125 subsamples of size 200x200x200 voxels. For Grosmont carbonate, the laboratory measurements found a porosity of approximately 0.21, and a permeability range from 150 to 470 mD (Andrä et al., 2013a). In comparison, 3-D segmented binary samples have an average porosity of 0.19 and a permeability range from 6.7 mD (P10) to 1262.8 mD (P90), with a median (P50) of 149.4 mD

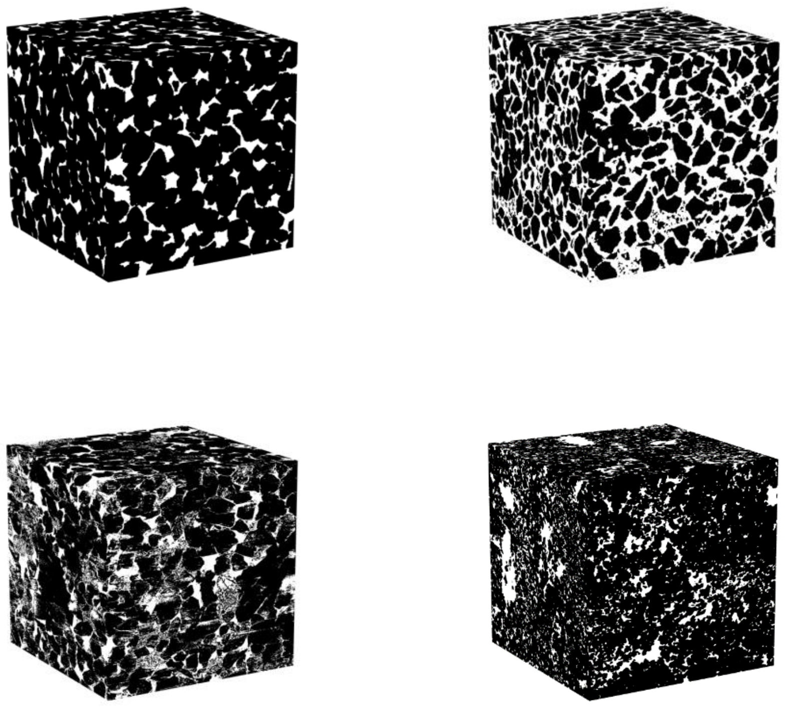


Figure A-4: From top-left to bottom-right, Fontainebleau sandstone, Bituminous sand, Berea sandstone, Grosmont carbonate in their original sizes.

Appendix B:

Numerical Simulations

This section describes different numerical simulations used in digital rock physics such as the Lattice Boltzmann flow simulations and finite element method. I will also discuss the effective of discretization in different numerical simulations.

B.1 LATTICE BOLTZMANN FLOW SIMULATION

Absolute permeability can be solved for numerically by using the Lattice Boltzmann (LB) flow simulation, which is an approximation of the Navier-Stokes equations for the pore space (Fredrich, 1999; Succi, 2001; Keehm, 2003). The LB algorithm is implemented in Windows C++ wrapped in MATLAB. The input to the algorithm is a 2-D or 3-D binary image, where 0 represents the pore space and 1 represents the mineral skeleton in a 3-D rectangular matrix.

Four different versions of SRB's LB simulation can be classified based on algorithm (time-dependent/time-independent) and boundary conditions (fixed flow rate/constant forcing with mirrored pore geometry) (Table B-1).

Time-dependent LB simulation utilizes collisions of imaginary particles and recovers the Navier-Stokes equation for long timespans and large spatial scales (Chen et al., 1992; Ladd, 1994). The algorithm involves three steps: (1) determination of the initial state of the density distribution, (2) particle collision, and (3) particle propagation. The time-independent (steady-state) computation, on the other hand, is done by formulating the Lattice Botlzmann algorithm in matrix form and solving the resulting linear system of equations (Verberg and Ladd, 1999).

On the side walls, no-flow boundary conditions are assumed. The fixed flow rate scheme simulates the pressure gradient along the flow path (Fredrich et al., 1999; Zhang and Zhang, 2000) and does not require mirroring of the pore space. Instead, it adds a buffer zone 15 pixels deep to the inlet and outlet faces (Keehm, 2003). In contrast, the constant forcing with mirrored pore geometry simulates the pressure gradient along the flow path using a constant forcing scheme (Gunstensen et al., 1991; Ladd, 1994; Keehm and Bosl,

2003) (Figure B-1 and Figure B-2), which does require mirroring of the pore space geometry. More details regarding this version of Lattice Boltzman simulation can be found in “Comparison of different Lattice-Boltzmann flow simulation implementations: efficiency, convergence and stability” (SRB Annual Meeting 2003).

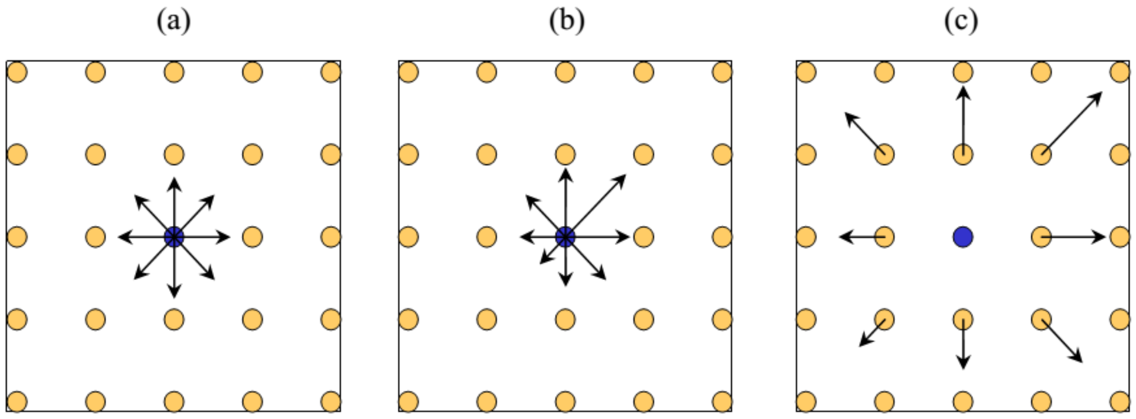


Figure B-1: Steps of the Lattice-Boltzmann algorithm: (a) initial state of the density distribution, (b) collision step, (c) propagation step (Keehm and Bosl, 2003).

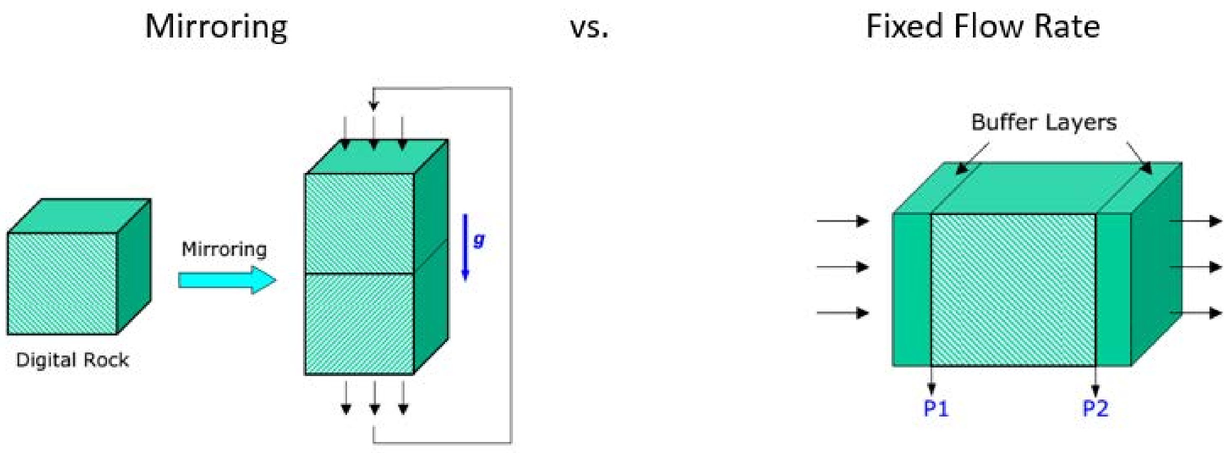


Figure B-2: (Left) constant forcing scheme with mirrored pore space geometry; (right) fixed flow rate scheme with a buffer zone of 15 pixels (Keehm and Bosl, 2003).

Table B-1: The four most current versions of the Lattice Boltzmann algorithm from Keehm (2003). The versions can be classified based on algorithm (time dependent/time-independent) and boundary conditions (fixed flow rate /constant forcing with mirrored pore geometry). Numbers in parentheses show the most current version available in SRB Tools.

	Time-dependent	Time-independent
Constant forcing with mirrored pore geometry	MR (3.1.0)	IMR (1.0.0)
Fixed Flow Rate	FP (2.0.0)	IFP (1.1.0)

Table B-2: Summary of characteristics of four implementations for the LB flow simulation (adapted from Keehm and Bosl, 2003).

Low →			High
Complexity			
IMR/MR		IFP/FP	
Grid Resolution			
IFP/FP		IMR/MR	
Memory			
FP	IFP	MR	IMR
Convergence			
FP	MR	IFP	IMR
Stability			
IFP	FP	IMR	MR

For complex porous media, Keehm and Bosl (2003) recommend using the LB FP and IFP versions for accuracy, but for simple geometry, they recommend using the LB MR and IMR versions (Table B-2). We tested our four LB versions and the COMSOL multi-physics finite element program by generating 3-D geometries of straight circular cross-sectional pipes with three different pipe lengths (100, 200, and 400 voxels) and computing absolute permeability from these 3-D images.

Figure B-3 shows that the fixed flow rate LB simulations (FP and IFP versions) predict absolute permeability closer to the theoretical value (blue line) as the pipes get longer. On the other hand, for constant forcing with mirrored pore geometry LB simulations (MR and IMR versions), the calculated absolute permeabilities do not depend on the pipe length. The LB IMR version performs best since the calculated permeabilities are the closest to the theoretical permeabilities, which assumes that the pipe is infinitely long. Therefore, the LB IMR version is well suited for simplified pore space geometry with high grid resolution. Figure B-4 shows that different SRB versions of the Lattice Boltzmann program perform well in a realistic complex geometry (Keehm & Bosl, 2003). The blue lines show the laboratory measurements of absolute permeability.

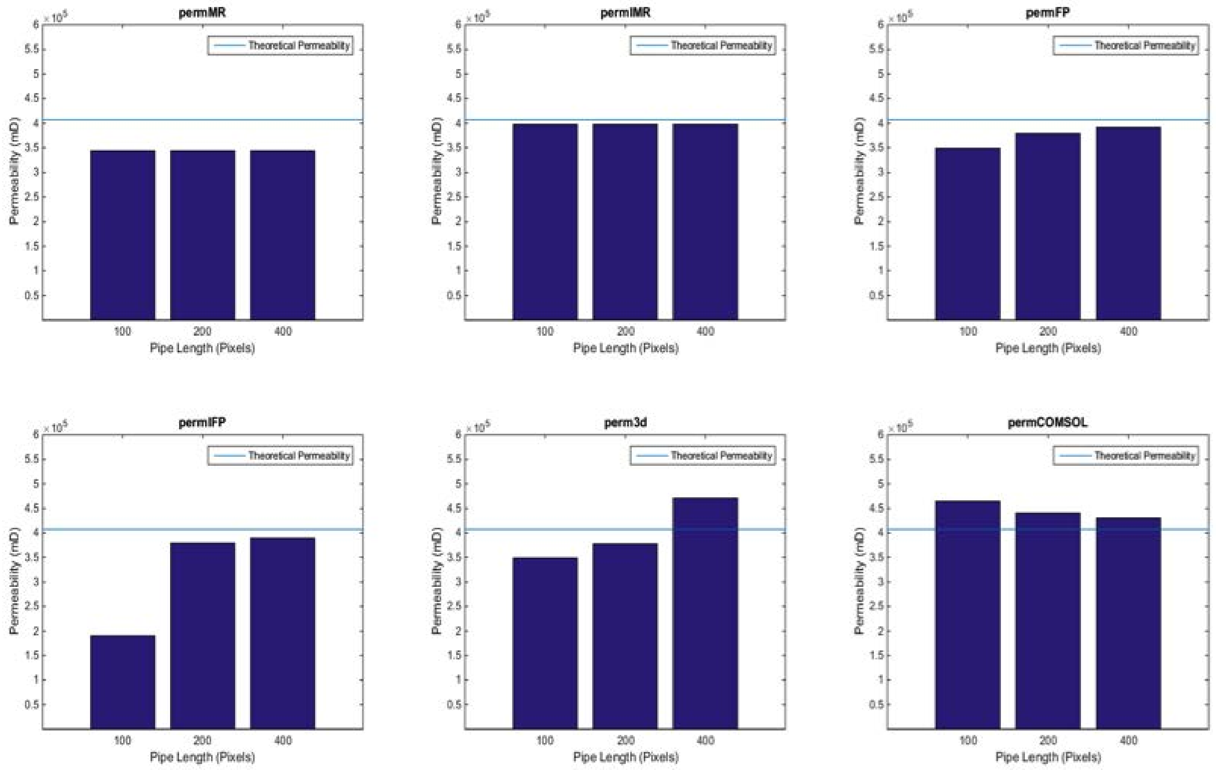


Figure B-3: Comparison of permeability predictions in a circular pipe from five different versions of the Lattice Boltzmann program and the COMSOL finite element program. The blue line in each graph represents the theoretical permeability value. In each graph, the length of the pipe is 100, 200, or 400 pixels.

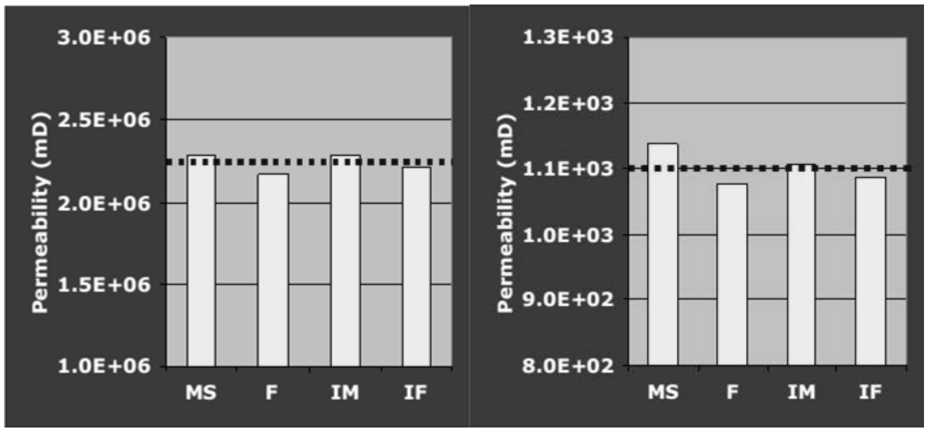


Figure B-4: Calculated Permeability (mD) of Finney Pack (left) and Fontainebleau Sandstone (right) for four different versions of the Lattice Boltzmann algorithm (MS = MR, F = FP, IM = IMR, IF = IFP) from (Keehm and Bosl, 2003).

B.1.1 Usage Notes

We use the LB FP version in the MATLAB wrapper since it is suitable for complex geometry. The LB simulation is performed by applying the pressure gradient along the x-direction.

The image3D input can be either a 3-D matrix of a single 3-D porous image or a cell array containing 3-D matrices of multiple 3-D porous images. The program will run multiple simulations if a cell array containing 3-D matrices is the input. The value dx is the voxel edge length (the length of 1 pixel in mm). The outputs are (1) a double for a single porous image or a vector for multiple porous images and (2) local flux in a 4-D matrix or a cell array containing 4-D matrices. The local flux can be used further for finding streamlines. The order of indices in an image is (nx, ny, nz).

B.2 ELECTRICAL RESISTIVITY

We used a finite-element solver to compute elastic moduli and electrical resistivity (Garboczi, 1998; Arns et al., 2002). The solver utilizes discrete forms of partial differential equations on a regular Cartesian grid. The program (EC3D/ELECFEM3D) is available from the National Institute of Standards and Technology (NIST). The output is the current in amperes in the x, y, and z directions, which can be used to calculate the resistivity using Ohm's law. A potential difference of 1 volt across the sample is implemented, and the boundary condition for the electric field is periodic. Material conductivity is shown in Table B-3.

B.3 ELASTIC MODULI

The effective elastic moduli were also calculated using the NIST Finite element programs (EMC3D/ELAS3D) (Garboczi, 1998) using a periodic boundary condition. The program yields the effective linear elastic properties of the rocks including bulk modulus, shear modulus, and density.

Since the purpose of this study is to investigate the influence of scale and resolution on model results, material properties (conductivity, bulk modulus, shear modulus, and density) are set to be constant for all digital samples, to focus only on changes to the scale and the geometry of pore space, which is assumed to contain brine. The chosen properties are listed in Table B-3.

Table B-3: A list of material properties used in the electrical resistivity and elastic moduli finite element solver.

	Conductivity ($\Omega^{-1}m^{-1}$)	Bulk Modulus (K) (GPa)	Shear Modulus (G) (GPa)	Density (g/cm^3)
Solid (1) – Quartz	$0.5*10^{-14}$	36.6	44.0	2.65
Pore space (0) – Brine	1	3.014	0	1.055

B.4 EFFECT OF DISCRETIZATION IN NUMERICAL SIMULATIONS

Since effective elastic and transport properties also depend on the discretization of digital samples, we tested the influence of scale (the size of sample) and resolution (voxel edge length) on Minkowski Functionals (porosity, specific surface area, integral of mean curvature, and Euler number), absolute permeability, electrical resistivity, and effective elastic properties. The appropriate scale and resolution were described using autocorrelation (Keehm, 2003), yet the autocorrelation requires us to obtain the digital sample first. This method then cannot be used as the feasibility test of appropriate digital sample scale and resolution. In the future we suggest to evaluate appropriate scale and resolution based on practical parameters such as grain size.

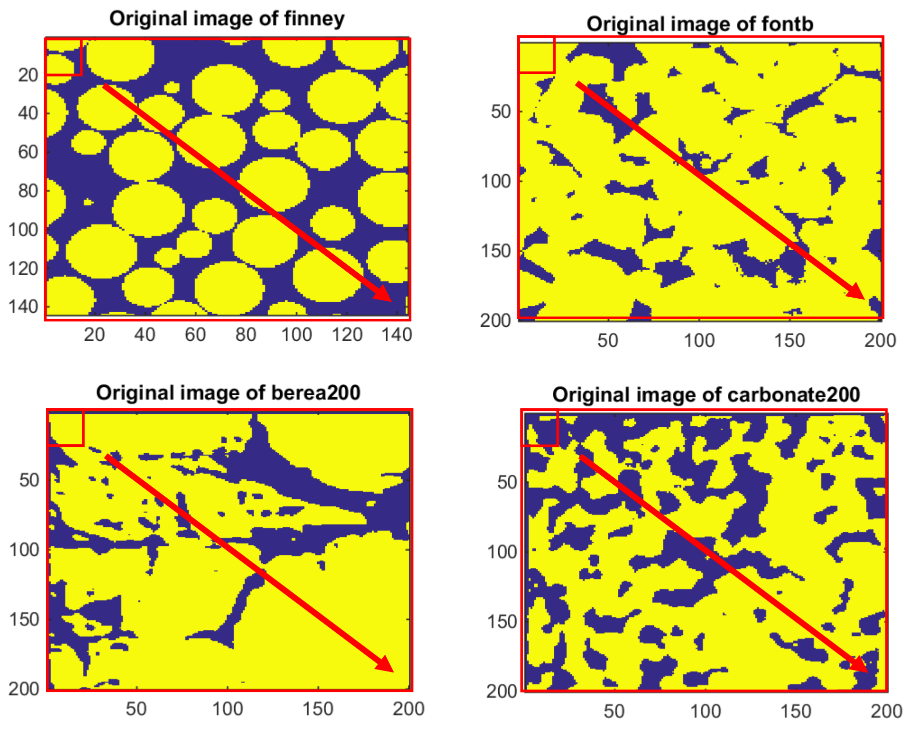


Figure B-5: 2-D representation of the Finney Pack, Fontainebleau sandstone, Berea sandstone, and carbonate. The red line shows the method for extracting images to investigate the scale effect.

B.4.1 Scale

For investigating the effect of changes in scale, binary images of increasing size were selected, ranging from 10^3 voxels to the maximum size of each sample (140^3 or 200^3 voxels) in 10 voxel increments. (Figure B-5).

The 2-D autocorrelation function can be obtained either from the MATLAB 2-D cross correlation function (xcorr2.m) or by using Fourier transforms. For Fourier transforms, the autocorrelation of distance h is as follows:

$$Autocorr(h) = F^{-1}\{F(A).*\,F^{*}(A)\}.$$

In this expression, A is a 2-D binary image, F denotes the Fourier transform, F* denotes the conjugate of the Fourier transform, and F^{-1} denotes the inverse Fourier transform (Keehm, 2003). Both methods yield a similar 2-D autocorrelation functions. After obtaining a 2-D autocorrelation, 1-D horizontal and vertical autocorrelations can be extracted from the center of the 2-D autocorrelation image to the boundary. Autocorrelation length can be obtained by finding the distance from the origin at which the autocorrelation function stops decreasing (Keehm, 2003).

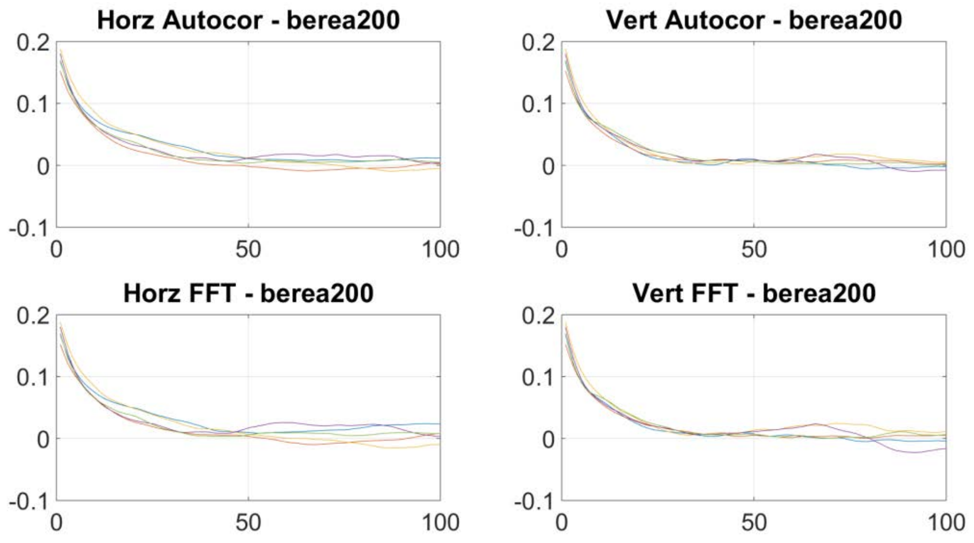


Figure B-6: 1-D horizontal and vertical autocorrelation from xcorr2 function in MATLAB (top) compared with Fourier transforms (bottom). Both methods yield similar autocorrelation length.

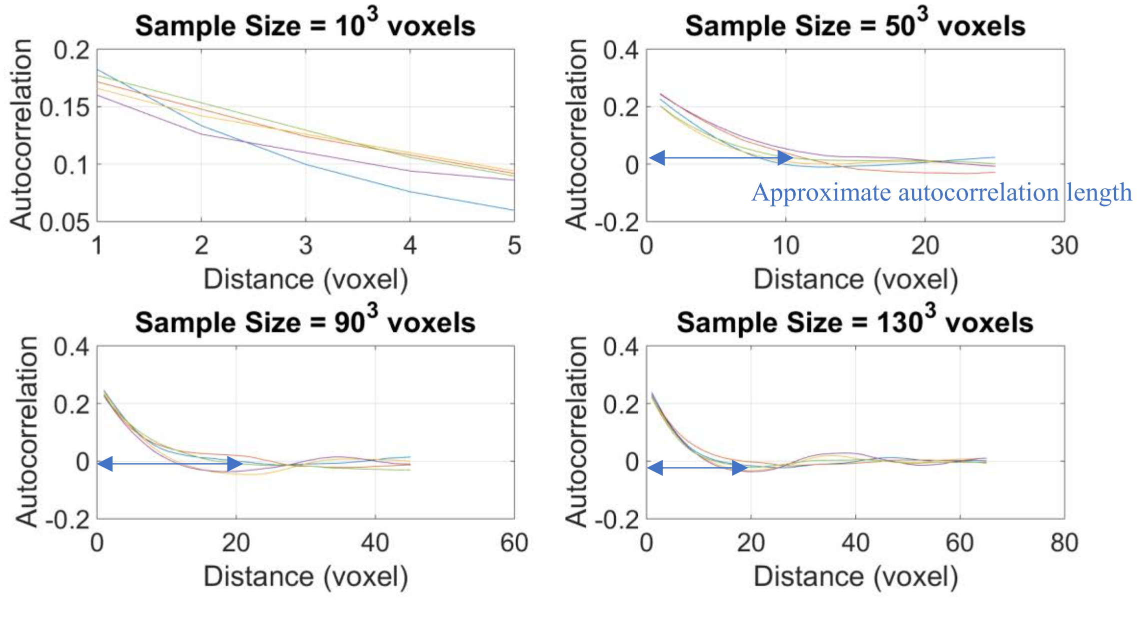


Figure B-7: The 1-D horizontal autocorrelation function of the Finney pack. The autocorrelation length increases as the size of the sample increases. For example, for sample size 10^3 voxels, the autocorrelation length is approximately 10 voxels, and for sample size 130^3 voxels, the autocorrelation length is approximately 20 voxels.

The representative elementary volume (REV) can be defined as the minimum sample size that would yield the value representative of the entire rock (Bear, 1988). It should be noted that autocorrelation length often increases as the size of sample increases, even in the extremely homogeneous case of the Finney pack, which is a random pack of identical spheres (Figure B-7). This raises the question of whether or not autocorrelation length is a good measure of appropriate sample size. There are two arguments here: (1) one needs to obtain the digital sample of appropriate size in order to estimate the appropriate sample size, which defeats the purpose of saving time and resources, and (2) the larger the sample size, the larger the REV derived from the sample, meaning that there is not one solution for ideal sample size using this method. Using digital rock samples, the REV from the physical properties can also be estimated by extracting the new sample images ranging in size from 10^3 voxels to the maximum size (140^3 or 200^3 voxels) in 10 voxel increments and running the simulation to obtain the physical properties of each image.

Figure B-8 shows the convergence of physical properties as the size of the sample approaches the REV. The black dots in the graph show the REV value determined using the following convergence criteria: if the physical property changes by less than 7% for 5 consecutive size steps after a given size, then that size is regarded as the REV value. The REV values are summarized in Table B-4.

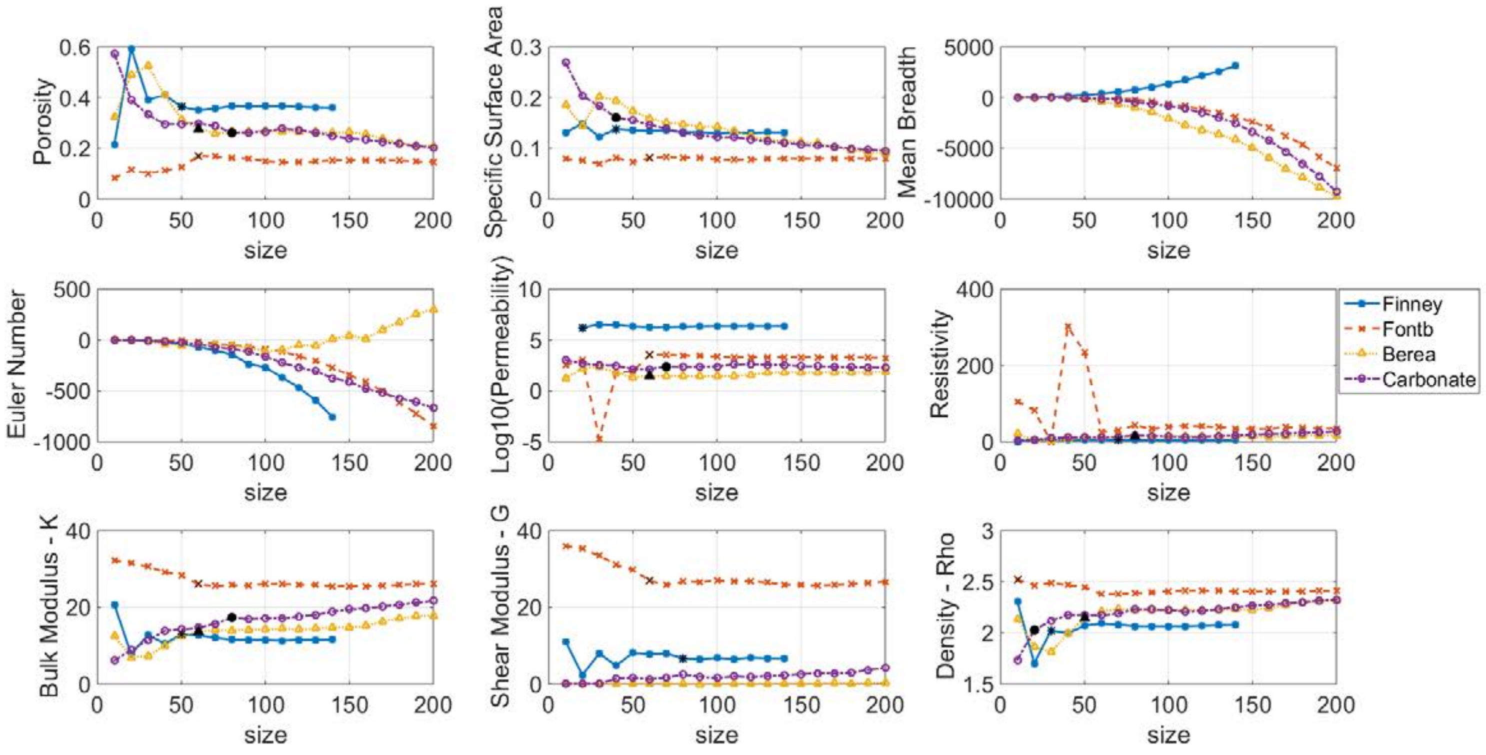


Figure B-8: Physical properties vs. sample size for all benchmark digital rock samples. The properties are (1) porosity, (2) specific surface area, (3) mean breadth, (4) Euler number, (5) permeability on a log 10 scale, (6) resistivity, (7) bulk modulus, (8) shear modulus, and (9) density.

Mean Breadth and the Euler Number do not have convergence since these properties grow as the size of the sample increases. Observe that elastic properties generally have larger REV compared to other properties such as absolute permeability and electrical resistivity.

Table B-4: REV in length of the cube (in voxel units) for each physical property of the benchmark digital rocks.

	Porosity	Specific Surface Area	Mean Breadth	Euler Number	Absolute Permeability	Resistivity	Bulk Modulus (K)	Shear Modulus (G)	Density (Rho)
Finney	50	40	0	0	20	70	50	80	30
Fontainebleau	60	60	0	0	60	0	60	60	10
Berea	60	0	0	0	60	80	60	0	50
Carbonate	80	40	0	0	70	0	80	0	20

B.4.2 Resolution

We investigated the resolution effect by increasing voxel edge length (dx), or the length per pixel, which has the effect of decreasing resolution. For each image, dx is artificially increased by averaging the image in a specified window. For example, an image with ½ the resolution of the original image can be created by averaging 2^3 voxels into 1 larger voxel. If the averaged value is greater than 0.5, the new voxel is assigned a value of 1 to represent the solid. Table B-5 shows the dx associated with new images at 1/2x, 1/4x, and 1/8x resolutions. Figure B-9 shows visualization of the new images. At 1/8x resolution, the images hardly resemble the original pore space geometry.

Table B-5: dx for each resolution on the benchmark digital rock.

Resolution	Finney dx(mm)	Fontainebleau dx(mm)	Berea dx(mm)	Carbonate dx(mm)
1x	0.07	0.0075	0.00074	0.00202
1/2 x	0.14	0.015	0.00148	0.00404
1/4 x	0.28	0.03	0.00296	0.00808
1/8 x	0.56	0.06	0.00592	0.01616

After producing lower resolution digital samples, we can then use the numerical simulation techniques discussed above to determine the effect of voxel size on the results (Figure B-10). As the resolution decreases, the porosity tends to become lower and the specific surface area (SSA) increases. The mean breadth and the Euler number of a sample are smaller because lower resolution images show fewer distinct grains. The numerical simulations of absolute permeability perform relatively well in low resolution images, but they begin to perform poorly at 1/8x resolution. The effect of resolution on electrical resistivity is similar. This is because these properties largely depend on the pore-filling fluid conductivity and the connectivity of pore space as in general the solid is less conductive than the pore fluid. In conclusion, the reductions in resolution begin to have an effect on transport properties only when major connected flow paths begin to disappear. Effective elastic moduli are governed by mineral components and mechanics at the grain contacts. As resolution decreases, the geometry of pore space is oversimplified and grain contacts becomes larger, leading to a stiffer simulated rock frame. As a result, both the bulk and shear moduli are highly sensitive to the resolution of digital sample. For numerical simulation involving effective elastic moduli, the resolution of digital images should be as high as possible.

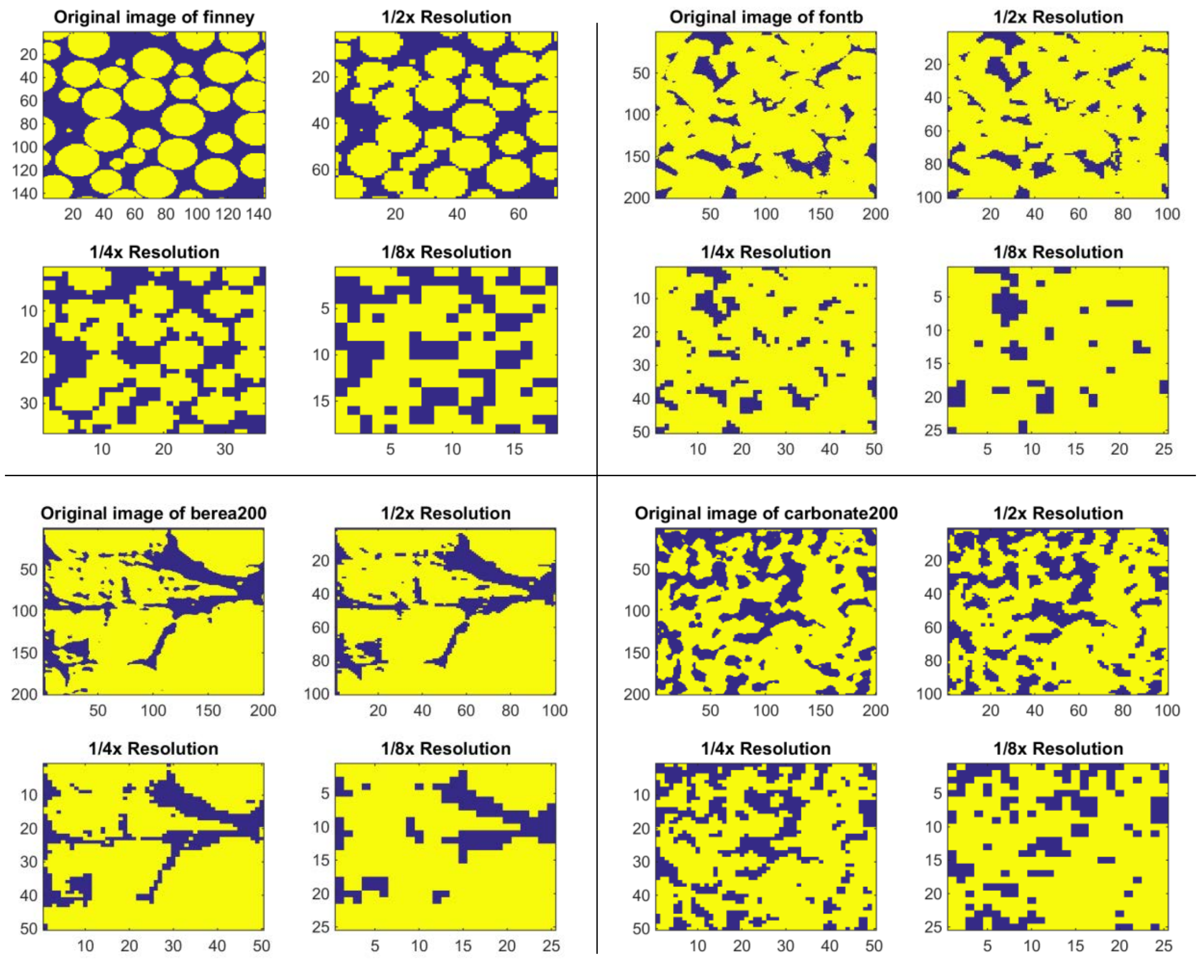


Figure B-9: Images of benchmark digital rock sample in 1x, 1/2x, 1/4x, 1/8x resolutions.

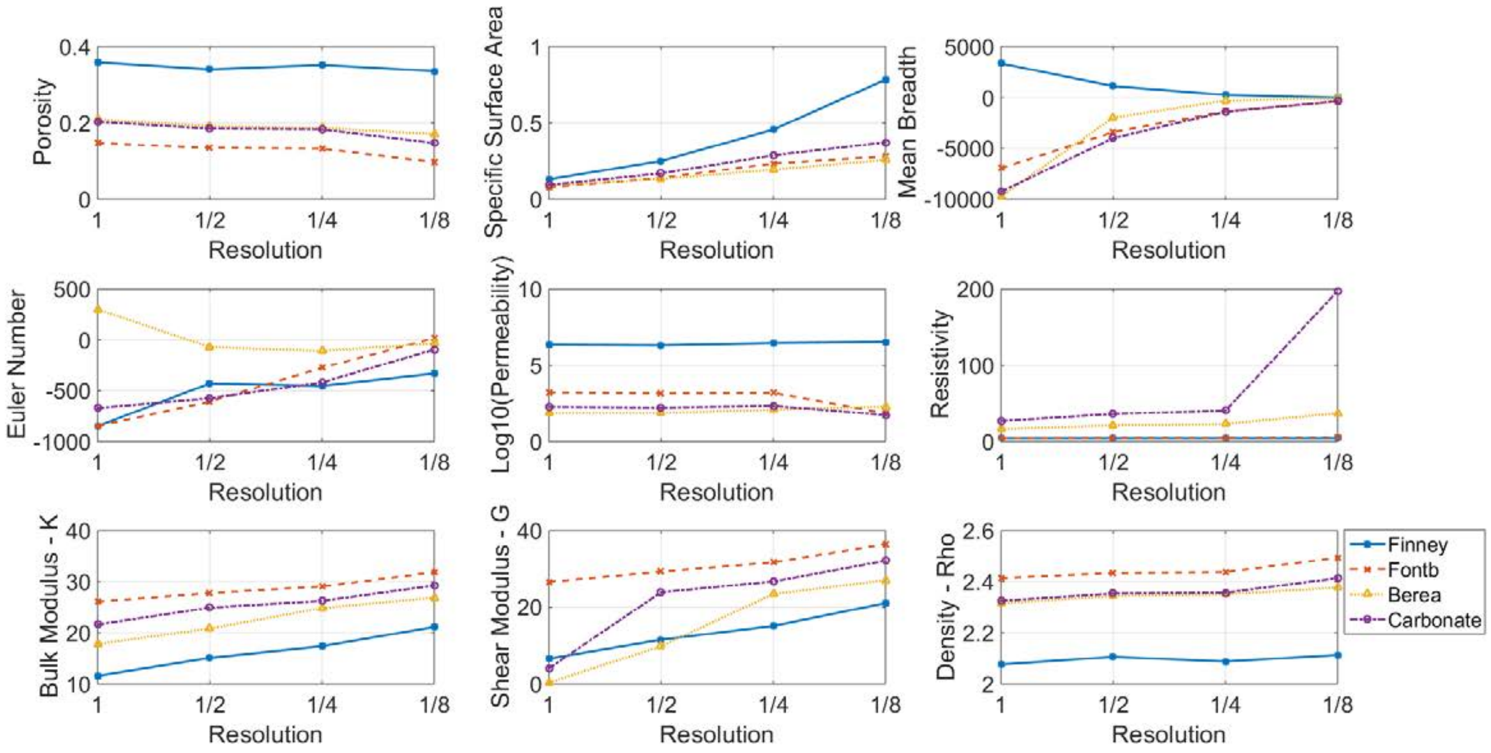


Figure B-10: Physical properties vs. resolution of each sample. The properties are (1) porosity, (2) specific surface area, (3) mean breadth, (4) Euler number, (5) permeability in log 10 scale, (6) resistivity, (7) bulk modulus, (8) shear modulus, and (9) density.

Appendix C:

Codes

```
function [image3D] = createCylinder(nx,ny,nz,r,ct)
%createCylinder creates a 3-D binary image of a round pipe
%
%   Input Arguments
%   - nx      : an integer, number of pixel in x-direction
%   - ny      : an integer, number of pixel in y-direction
%   - nz      : an integer, number of pixel in z-direction
%   - r0      : an integer, radius of a cylinder pipe
%   - ct      : an integer, center of a cylinder pipe
%
%   Output Arguments
%   - image3D       : a (ny*nx*nz) uint8 matrix, 3-D binary image of
%                     pore space (0 = pore, 1 = grain)
%
%   Note:
%       In order to run this code, qCBinary.m file is needed.

%   Revision 1: August 2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)



%% Program
% Create the image
[x y z]      = meshgrid(1:ny, 1:nx, 1:nz);
tempImage    = sqrt((x-ct).^2 + (z-ct).^2) < r;
tempImage    = qCBinary(tempImage);

% Output
image3D     = abs(1-tempImage);
end
```

```
function [image3D] = createEllipse(nx,ny,nz,a,b,ct)
%createEllipse creates a 3-D binary image of an elliptic pipe
%
%   Input Arguments
%   - nx      : an integer, number of pixel in x-direction
%   - ny      : an integer, number of pixel in y-direction
%   - nz      : an integer, number of pixel in z-direction
%   - a       : an integer, semi-major axes
%   - b       : an integer, semi-minor axes
%   - ct      : an integer, center of a cylinder pipe
%
%   Output Arguments
%   - image3D       : a (ny*nx*nz) uint8 matrix, 3-D binary image of
%                     pore space (0 = pore, 1 = grain)
%
%   Note:
%       In order to run this code, qCBinary.m file is needed.

%   Revision 1: August 2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)
```

```
%% Program
% Create the image
[x y z] = meshgrid(1:nx, 1:ny, 1:nz);
temp = ((x-ct)./a).^2 + ((z-ct)./b).^2 < 1;
temp = QCbinary(temp);

% Output
image3D = abs(1-temp);

end
```

```
function [image3D] = createEqTriangle(nx,ny,nz,t,ct)
%createEqTriangle creates a 3-D binary image of an equilateral
triangle pipe
%
%   Input Arguments
%   - nx     : an integer, number of pixel in x-direction
%   - ny     : an integer, number of pixel in y-direction
%   - nz     : an integer, number of pixel in z-direction
%   - t      : an integer, length of equilateral triangle
%   - ct     : an integer, center of a cylinder pipe
%
%   Output Arguments
%   - image3D      : a (ny*nx*nz) uint8 matrix, 3-D binary image of
%                    pore space (0 = pore, 1 = grain)
%
%   Note:
%       In order to run this code, qCBinary.m file is needed.

%   Revision 1: August 2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)


%% Program

% Create the image
[x y z] = meshgrid(1:nx, 1:ny, 1:nz);
c = t/2;
d = sqrt(3)*t/2;
temp1 = and(and(abs(x-ct+t/4)<c/2,abs(z-ct)<d/2),z < d/c*(x-ct+t/4) +
ct );
temp2 = and(and(abs(x-ct-t/4+1)<c/2,abs(z-ct)<d/2),z < -d/c*(x-ct-t/4)
+ ct );
temp = temp1+temp2;
temp = qCBinary(temp);

% Output
image3D = abs(1-temp);
end
```

```
function [image3D] = createCrescent(nx,ny,nz,r0,phi,ct)
%createCrescent creates a 3-D binary image of a crescent pipe
%
%   Input Arguments
%   - nx      : an integer, number of pixel in x-direction
%   - ny      : an integer, number of pixel in y-direction
%   - nz      : an integer, number of pixel in z-direction
%   - r0      : an integer, radius of a cylinder pipe that is required
..
%               to change the size of crescent pipe
%   - ct      : an integer, center of the pipe
%   - phi     : an integer, angle governing how curve the crescent pipe
is
%
%
%   Output Arguments
%   - image3D      : a (ny*nx*nz) uint8 matrix, 3-D binary image of
%                    pore space (0 = pore, 1 = grain)
%
%   Note
%       (1) In order to run this code, qCBinary.m file is needed.
%       (2) the largest r within nx,nz = 100 is 24
%   Example
%       [Crescent] = createCrescent(100,200,100,24,pi/4,50);

%   Revision 1: August 2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)




%% Program
% Initialziation
[x y z] = meshgrid(1:nx, 1:ny, 1:nz);
Area = pi*r0^2;
b = sqrt(Area./((2-(2.*cos(phi)).^2).*phi+sin(2.*phi))); %verified to
give the same porosity
eps = cos(phi).*2;
a = b.*eps;

% Create the image
temp1 = sqrt((x-ct+b).^2 + (z-ct).^2) < a;
temp2 = sqrt((x-ct).^2 + (z-ct).^2) < b;
temp = temp2 - temp1;
temp = qCBinary(temp);

% Output
image3D = abs(1-temp);
end
```

```
function [image3D] = createRectangle(nx,ny,nz,a,b,ct)
%createRectangle creates a 3-D binary image of a rectangle pipe
%
```

```
%   Input Arguments
%   - nx      : an integer, number of pixel in x-direction
%   - ny      : an integer, number of pixel in y-direction
%   - nz      : an integer, number of pixel in z-direction
%   - a       : an integer, semi-major axes
%   - b       : an integer, semi-minor axes
%   - ct      : an integer, center of a cylinder pipe
%
%   Output Arguments
%   - image3D       : a (ny*nx*nz) uint8 matrix, 3-D binary image of
%                     pore space (0 = pore, 1 = grain)
%
%   Note:
%       In order to run this code, qCBinary.m file is needed.

%   Revision 1: August 2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)



%% Program

% Create the image
[x y z] = meshgrid(1:nx, 1:ny, 1:nz);
temp = and(abs(x-ct)<a/2,abs(z-ct)<b/2);
temp = qCBinary(temp);

% Output
image3D = abs(1-temp);

end
```

```
function [image3D] = createHypotrochoid(nx,ny,nz,r,k,ct)
%createHypotrochoid create a 3-D binary image of a hypotrochoidal pipe
%
%   Input Arguments
%   - nx      : an integer, number of pixel in x-direction
%   - ny      : an integer, number of pixel in y-direction
%   - nz      : an integer, number of pixel in z-direction
%   - r       : an integer, radius of the hypotrochoid
%   - k       : an integer, number of sides
%   - ct      : an integer, center of a cylinder pipe
%
%   Output Arguments
%   - image3D       : a (ny*nx*nz) uint8 matrix, 3-D binary image of
%                     pore space (0 = pore, 1 = grain)
%
%   Note:
%       (1) In order to run this code, qCBinary.m file is needed.
%       (2) For more information,
http://mathworld.wolfram.com/Hypocycloid.html

%   Revision 1: August 2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)
```

```
%% Program
% Create the image
[x y z] = meshgrid(1:nx, 1:ny, 1:nz);
theta = linspace(0,2*pi,200);

a = sqrt(1/2).*r; % valid for k =3;
for t=1:ny
    xv(t) = (k-1)*a*cos(theta(t)) + a*cos((k-1)*theta(t)) + ct;
    zv(t) = (k-1)*a*sin(theta(t)) - a*sin((k-1)*theta(t)) + ct;
end

for j = 1:ny
    temp(:,j,:) = inpolygon(x(:,j,:),z(:,j,:),xv,zv);
end
temp = qCBinary(temp);

% Output
image3D = abs(1-temp);

end
```

```
function [image3D] = createSinusoidalPipe(nx,ny,nz,r0,ct,delta)
%createSinusoidalPipe creates a 3-D binary image of a sinusoidal pipe
%
%   Input Arguments
%   - nx     : an integer, number of pixel in x-direction
%   - ny     : an integer, number of pixel in y-direction
%   - nz     : an integer, number of pixel in z-direction
%   - r0     : an integer, initial radius
%   - ct     : an integer, center of the pipe
%   - delta  : an integer, fractional change in radius
%
%   Output Arguments
%   - image3D      : a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                    pore space (0 = pore, 1 = grain)

%   Revision 1: April 2016 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)


%% Program

% Initialization
lambda = 2*r0;

% Grid meshing
[x, z] = meshgrid(1:nx, 1:nz);

t = linspace(0,4*pi*lambda,200);
```

```
% Equation
rr = r0*(1 + delta*sin(t./lambda));

for iSlice = 1:ny
    temp(iSlice,:,:) = sqrt((x - ct).^2 + (z - ct).^2) < rr(iSlice);
end

temp = qCBinary(temp);
image3D = abs(1-temp);

end
```

```
function [ image3DQC ] = qCBinary( image3D )
%qCBinary QC the image after any mathematical operation that it is
binary.
%
%   Input Arguments
%   - image3D       : a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                     pore space to be checked (0 = pore, 1 = grain)
%
%   Output Arguments
%   - image3DQC     : a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                     pore space (0 = pore, 1 = grain)

%   Revision 2: December  2015 Nattavadee Srisutthiyakorn (more
efficient)
%   Revision 1: September 2014 Nattavadee Srisutthiyakorn (QCbinary.m)
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)



%% Program
% Find the size
[nx, ny, nz]     = size(image3D);

% Replace any value greater than 1 with 1
tempImage        = image3D(:);
tempImage(tempImage > 1)  = 1;

% Reshape back into the same shape
image3DQC        = reshape(tempImage,[nx, ny, nz]);
```

```
function [ image3D ] = createSCP( cubeLength, nUnitCell,
grainDilationRatio )
%createSCP creates a 3D image of simple cubic pack
%
%   Input Arguments
%   - cubeLength    : an integer, length of a 3D image cube in pixel
%                     (cubeLength = nx = ny = nz)
%   - nUnitCell     : an integer, number of unit cell of the size,
%                     for example, nUnitcell = 2 resulting in 2^3 unit
cells
%                     in 3-D images
%                    (Default: 1 for unit cell of SCP)
%   - grainDilationRatio
%                   : an integer, the size of sphere in relation to
%                     the original one. If it's greater than 1 then the
%                     spheres overlap
%                     (Default: 1 = using the original radius of
spheres)
%
%   Output Arguments
%   - image3D       : a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                     pore space (0 = pore, 1 = grain)
%
%   Note
%   - need to use qCBinary.m

%   Revision 1: December 2015 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)
%% QC Inputs
if nargin < 2
    nUnitCell = 1;
    grainDilationRatio = 1;
end



%% Initialization
unitCellLength = ceil(cubeLength./nUnitCell);

sphereRadius = unitCellLength/2;
endpt = unitCellLength;

CT      = [ 0       0       0;
            endpt   0       0;
            0       endpt   0;
            0       0       endpt;
            endpt   endpt   0;
            endpt   0       endpt;
            0       endpt   endpt;
            endpt   endpt   endpt];

% Create a mesh
[x, y, z]   = meshgrid(1:unitCellLength, 1:unitCellLength,
1:unitCellLength);

image3DUnit = zeros(unitCellLength, unitCellLength, unitCellLength);

% Filling in the identical spheres
for iSphere = 1:8
    tempImage   = sqrt((x - CT(iSphere,1)).^2 + (y - CT(iSphere,2)).^2
...
                + (z - CT(iSphere,3)).^2) <
sphereRadius.*grainDilationRatio;
    image3DUnit = image3DUnit + tempImage;
```

```
end

% QC the overlap
image3DUnit = qCBinary(image3DUnit);

% Expand the unit cell
if nUnitCell > 1
    image3D = expandUnitCell( image3DUnit, nUnitCell);
    image3D = image3D(1:cubeLength,1:cubeLength,1:cubeLength);
else
    image3D = image3DUnit;
end



end
```

```
function [ image3D ] = createFCP( cubeLength, nUnitCell,
grainDilationRatio )
%createFCP creates a 3D image face-centered cubic pack
%
%   Input Arguments
%   - cubeLength   : an integer, length of a 3D image cube in pixel
%                     (cubeLength = nx = ny = nz)
%   - nUnitCell    : an integer, number of unit cell of the size,
%                    for example, nUnitcell = 2 resulting in 2^3 unit
cells
%                    in 3-D images
%                   (Default: 1 for unit cell of SCP)
%   - grainDilationRatio
%                  : an integer, the size of sphere in relation to
%                    the original one. If it's greater than 1 then the
%                    spheres overlap
%                    (Default: 1 = using the original radius of
spheres)
%
%   Output Arguments
%   - image3D      : a (nx*ny*nz) uint8 matrix, 3-D binary image of a
%                    face-centered cubic pack (0 = pore, 1 = grain)
%
%   Note
%   - need to use qCBinary.m

%   Revision 1: March 2016 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)
%% QC Inputs
if nargin < 2
    nUnitCell = 1;
    grainDilationRatio = 1;
end
if nargin < 3
    grainDilationRatio = 1;
end
```

```
%% Initialization
unitCellLength = ceil(cubeLength./nUnitCell);

sphereRadius = ceil(unitCellLength./(2*sqrt(2)));
midpt = floor(unitCellLength/2);
endpt = unitCellLength;

edgeCT   = [ 0       0       0;
             endpt   0       0;
             0       endpt   0;
             0       0       endpt;
             endpt   endpt   0;
             endpt   0       endpt;
             0       endpt   endpt;
             endpt   endpt   endpt];

midCT    = [ 0       midpt   midpt;
             midpt   0       midpt;
             midpt   midpt   0;
             endpt   midpt   midpt;
             midpt   endpt   midpt;
             midpt   midpt   endpt;];

% Create a mesh
[x, y, z]   = meshgrid(1:unitCellLength, 1:unitCellLength,
1:unitCellLength);

image3DUnit = zeros(unitCellLength, unitCellLength, unitCellLength);

% Filling in spheres at the edge
for iSphere = 1:8
    tempImage   = sqrt((x - edgeCT(iSphere,1)).^2 + (y -
edgeCT(iSphere,2)).^2 ...
                + (z - edgeCT(iSphere,3)).^2) <
sphereRadius.*grainDilationRatio;
    image3DUnit = image3DUnit + tempImage;
end

% Filling in spheres at the middle
for iSphere = 1:6
    tempImage   = sqrt((x - midCT(iSphere,1)).^2 + (y -
midCT(iSphere,2)).^2 ...
                + (z - midCT(iSphere,3)).^2) <
sphereRadius.*grainDilationRatio;
    image3DUnit = image3DUnit + tempImage;
end

% QC the overlap
image3DUnit = qCBinary(image3DUnit);

% Expand the unit cell
if nUnitCell > 1
    image3D = expandUnitCell( image3DUnit, nUnitCell);
    image3D = image3D(1:cubeLength,1:cubeLength,1:cubeLength);
```

```
else
    image3D = image3DUnit;
end

end
```

```
function [ image3D ] = createSphericalPack( locationX, locationY,
locationZ,...
                                            radius, cubeSize )
%createSphericalPack creates a 3-D binary image of a sphere pack
%
%   Input Arguments
%   - locX          : a (nSph*1) double vector, x coordinate location
%   - locY          : a (nSph*1) double vector, y coordinate location
%   - locZ          : a (nSph*1) double vector, z coordinate location
%   - radius        : a (nSph*1) double vector, radius of a sphere
%   - cubeSize      : an integer, size of the pack
%                     (Example: cubeSize = 200 -> 200^3 px cube;
%
%   Output Arguments
%   - image3D       : a (nx*ny*nz) int8 matrix, 3-D binary image of a
%                     rock (0 = pore, 1 = grain)
%
%   Revision 1: January 2016 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)


%% Program
% Initialization
bufferZone  = 10;
nz          = cubeSize + bufferZone*2;

% Find the dimension of this pack
minVec = [min(locationX) min(locationY) min(locationZ)];
maxVec = [max(locationX) max(locationY) max(locationZ)];

% Scaling the cube size
locationX    = locationX.*nz./maxVec(1);
locationY    = locationY.*nz./maxVec(2);
locationZ    = locationZ.*nz./maxVec(3);
radius   = radius.*nz./maxVec(1);

% Create the geometry
[x, y, z] = meshgrid(1:nz, 1:nz, 1:nz);
tempImage = zeros(nz,nz,nz);

for iSph = 1:size(radius,1)
    tempSph = sqrt((x - locationX(iSph)).^2 ...
                 + (y - locationY(iSph)).^2 ...
                 + (z - locationZ(iSph)).^2) < radius(iSph);
    tempImage = tempImage + tempSph;
end
```

```
tempImage = qCBinary(tempImage);

% Output
image3D = int8(tempImage(11:cubeSize + bufferZone, ...
                         11:cubeSize + bufferZone, ...
                         11:cubeSize + bufferZone));



end
```

```
function [ image3DConnected ] = createConnectedPorespace( image3D )
%createConnectedPorespace creates a 3-D binary image of connected pore
space
%
%   Input Arguments
%   - image3D           : a (nx*ny*nz) uint8 matrix, 3-D binary image
of
%                         pore space (0 = pore, 1 = grain)
%
%   Output Arguments
%   - image3DConnected  : a (nx*ny*nz) uint8 matrix, 3-D binary image
of
%                         effective (connected) pore space
%                        (0 = pore, 1 = grain)

%   Revision 1: October  2015 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)




%% Initialization
[nx, ny, nz]        = size(image3D);
image3DConnected    = ones(nx, ny, nz);


% Step 1: Labeling the pores
% Inverse grains <-> pores
image3DInverse      = abs(1 - image3D);
poreLabel           = bwconncomp(image3DInverse);
image3DInverseLabel = labelmatrix(poreLabel);


% Step 2: Find the label number that exist on both ends
tempFirstSlide  = image3DInverseLabel(:,:,1);
tempLastSlide   = image3DInverseLabel(:,:,end);

labelFirstSlide = unique(tempFirstSlide);
labelLastSlide  = unique(tempLastSlide);
labelEffective  = intersect(labelFirstSlide, labelLastSlide);


% Step 3: Create connected pore space
```

```
nLabel = length(labelEffective);
for iLabel = 1:nLabel
    label = labelEffective(iLabel);
    if label >= 1 % Pore = 1+ -> 0
        image3DConnected(image3DInverseLabel == label) = 0;
    else % Grain = 0 -> 1
        image3DConnected(image3DInverseLabel == label) = 1;
    end
end




end
```

```
function [ porosity, specificSurfaceArea, meanBreadth, eulerNumber ]
...
    = computeMinkowski3D( image3D, option )
%computeMinkowski3D porosity, specific surface area, mean breadth,
eulerNo
%
%   Input Arguments
%   - image3D      : Two types of inputs are possible
%                    (1) a single digital rock
%                     a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                    pore space (0 = pore, 1 = grain)
%                           ---- or ----
%                    (2) a cell array of digital rocks
%                    a cell array containing matrix as specified above
%   - option       : an integer, 0 for nConnection (6)  and nDirection
(3)
%                                1 for nConnection (26) and nDirection
(13)
%
%   Output Arguments
%   - porosity            : a vector (nImage*1), porosity
%   - specificSurfaceArea : a vector (nImage*1), surface area/length^3
%   - meanBreadth         : a vector (nImage*1), mean breadth
%   - eulerNumber         : a vector (nImage*1), Euler's number
%
%   Example
%       [BereaFRS200_Results.Original.porosity, ...
%        BereaFRS200_Results.Original.specificSurfaceArea, ...
%        BereaFRS200_Results.Original.meanBreadth, ...
%        BereaFRS200_Results.Original.eulerNumber] ...
%        = computeMinkowski3D(BereaFRS200, 1)
%   Note
%       In order to run this code, imMinkowski files are needed.
%       "Computation of Minkowski measures on 2D and 3D binary
images".
%       David Legland, Kien Kieu and Marie-Francoise Devaux (2007)
%       Image Analysis and Stereology, Vol 26(2), June 2007
%       web: http://www.ias-iss.org/ojs/IAS/article/view/811

%   Revision 3: April  2016 Nattavadee Srisutthiyakorn
%   Revision 2: August 2015 Nattavadee Srisutthiyakorn
%   Revision 1: June   2014 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)



%% QC Inputs

if nargin < 2
    option = 1;
end



%% Check whether the input is a matrix or a cell array of matrices

if iscell(image3D)

    % Initialization
    nImage              = length(image3D);
    porosity            = zeros(nImage, 1);
    specificSurfaceArea = zeros(nImage, 1);
    meanBreadth         = zeros(nImage, 1);
    eulerNumber         = zeros(nImage, 1);

    for iImage = 1:nImage

        disp(['Current image:
(',num2str(iImage),'/',num2str(nImage),')'])

        try
        [ porosity(iImage), specificSurfaceArea(iImage),...
            meanBreadth(iImage), eulerNumber(iImage) ] ...
            = computeMK3D( image3D{iImage}, option );
        end

        % Save every 50 iteration
        if nImage > 50 && floor(iImage/50) == iImage/50
            save('tempMinkowski','porosity','specificSurfaceArea',...
                'meanBreadth','eulerNumber');
        end

    end

else
    [ porosity, specificSurfaceArea, meanBreadth, eulerNumber ] ...
        = computeMK3D( image3D, option );
end

end
```

```
function [ porosity, specificSurfaceArea, meanBreadth, eulerNumber ]
...
    = computeMK3D( image3D, option )
%% QC Inputs
[~, ~, nZ] = size(image3D);

% Check inputs
if nZ < 2
    help(mfilename);
    error('Error: image3d input is required to be 3-D')
end



%% Create reverse image(s)
image3DInverse = abs(1-image3D);


%% Run Minkowski Functionals
switch option
    case 0
        nConnectivity = 6;
        nDirection    = 3;
    case 1
        nConnectivity = 26;
        nDirection = 13;
end

% Perform a check whether it's all solid or all pore space
checkOriginal = any(any(any(image3D,3)));
checkReverse  = any(any(any(image3DInverse,3)));

if and(checkOriginal, checkReverse)
    % Porosity
    try
        porosity = 1 - imVolumeDensity(image3D);
    catch
        porosity = NaN;
    end

    % Specific Surface Area (Surface Estimate/Length^3)
    try
        specificSurfaceArea = imSurfaceDensity(image3D, nDirection);
    catch
        specificSurfaceArea = NaN;
    end

    % Mean Breadth
    try
        meanBreadth = imMeanBreadth(image3D, nDirection);
    catch
        meanBreadth = NaN;
    end
    % Euler Number
    try
```

```
        eulerNumber = imEuler3d(image3D, nConnectivity);
    catch
        eulerNumber = NaN;
    end
    % Warning if number of element > 1
    if numel(porosity) > 1
        warning(['WARNING: imMinkowski yileds the number'...
            ' of element greater than 1'])
    end

end

end
```

```
function [ Proximity2D, Proximity3D ] ...
    = compute2D3DProximity( image3D )
%computeStreamlines2D3DProximity extracts proximity to the nearest
solid
%
%   Input Arguments
%   - image3D         : a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                       pore space (0 = pore, 1 = grain)
%
%   Output Arguments
%   - Proximity2D     : a (nx*ny*nz) uint8 matrix, 3-D double image of
%                       2-D proximity (0 = pore, 1 = grain)
%   - Proximity3D     : a (nx*ny*nz) uint8 matrix, 3-D double image of
%                       3-D proximity (0 = pore, 1 = grain)

%   Revision 1: April    2016 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)



%% Program
% Rotate matrix into the the flow along x direction
[nx, ny, nz] = size(image3D);
for iSlice = 1:nx
    image3DRot(:,:,iSlice) = reshape(image3D(iSlice,1:ny,1:nz),ny,nz);
end

% Find the distance matrix in 2D, 3D
Proximity3D = bwdist(image3DRot,'euclidean');
Proximity2D = zeros(ny, nz, nx);

for iSlice = 1:nx
    image2D = image3DRot(:,:,iSlice);
    Proximity2D(:,:,iSlice) = bwdist(image2D,'euclidean');
end
```

```
function [ StreamlinesXYZ, StreamlinesAbsFlux, totalDistance,
totalTime, ...
    totalFlux, tortuosity, tortuosityMin, tortuosityMean,
tortuosityMax, ...
    tortuosityFluxWeighted, tortuosityStd ] ...
    = computeStreamlines( localFlux )
%computeStreamlines compute streamlines from a local flux matrix
%
%   Input Arguments
%   - localFlux    : a (nx*ny*nz*3) matrix, local flux in x, y, z
direction.
%                  This is an output from latticeBoltzmannFP3D.m
%                           ---- or ----
%                  a cell array containing matrix as specified above
%
%   Output Arguments
%   for a single localFlux matrix,
%   - StreamlinesXYZ  : a cell array (nStreamline*1), x, y, z
locations of
%                       each streamline
%   - StreamlinesAbsFlux : a cell array (nStreamline*1),
%                          flux along streamline
%   - totalDistance   : a vector (nStreamline,1), distance
%   - totalTime       : a vector (nStreamline,1), distance/velocity
%   - totalFlux       : a vector (nStreamline,1), total flux of each
flow path
%   - tortuosity      : a vector (nStreamline,1), distance/nz of each
flow path
%   - tortuosityMin   : a double, minimum tortuosity of all
streamlines
%   - tortuosityMean  : a double, mean tortuosity of all streamlines
%   - tortuosityMax   : a double, maximum tortuosity of all
streamlines
%   - tortuosityFluxWeighted : a double, flux weighted mean tortuosity
of
%                              all streamlines
%   - tortuosityStd   : a double, standard deviation tortuosity of all
%                       streamlines

%   Revision 5: April    2016 Nattavadee Srisutthiyakorn
%   Revision 4: February 2016 Nattavadee Srisutthiyakorn
%   Revision 3: December 2015 Nattavadee Srisutthiyakorn
%   Revision 2: August   2015 Nattavadee Srisutthiyakorn
%   Revision 1: February 2015 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)




%% Program
% Internal Option
cleanPath = 1;

% Check whether the input is a matrix or a cell array of matrices

if iscell(localFlux)
```

```
    % Initialize
    nFlux              = size(localFlux, 2);

    for iFlux = 1:nFlux

        disp(['Current flux:
(',num2str(iFlux),'/',num2str(nFlux),')'])

        try
            [ StreamlinesXYZ{iFlux}, StreamlinesAbsFlux{iFlux}, ...
                totalDistance{iFlux}, totalTime{iFlux}, ...
                totalFlux{iFlux}, tortuosity{iFlux}, ...
                tortuosityMin(iFlux,:), tortuosityMean(iFlux,:), ...
                tortuosityMax(iFlux,:),
tortuosityFluxWeighted(iFlux,:), ...
                tortuosityStd(iFlux,:) ] ...
                = computeIndividalStreamlines( localFlux{iFlux},
cleanPath );

            % Save every 50 iteration
            if nFlux > 50 && floor(iFlux/50) == iFlux/50
                save('tempFlowPath', 'StreamlinesXYZ', 'tortuosity',
...
                    'totalTime', 'Stats');
            end
        end
    end


else
    [ StreamlinesXYZ, StreamlinesAbsFlux, totalDistance, totalTime,
...
        totalFlux, tortuosity, tortuosityMin, tortuosityMean,
tortuosityMax, ...
        tortuosityFluxWeighted, tortuosityStd ] ...
        = computeIndividalStreamlines( localFlux, cleanPath );
end

end



function [ StreamlinesXYZ, StreamlinesAbsFlux, totalDistance,
totalTime, ...
    totalFlux, tortuosity, tortuosityMin, tortuosityMean,
tortuosityMax, ...
    tortuosityFluxWeighted, tortuosityStd ] ...
    = computeIndividalStreamlines( localFlux, cleanPath )
%% QC Inputs
[~, ~, ~, type] = size(localFlux);
if type < 2
    help(mfilename)
    error('Incorrect Flux Type')
```

```
end




%% Extracting flow paths
uXRaw = localFlux(:,:,:,1);
uYRaw = localFlux(:,:,:,2);
uZRaw = localFlux(:,:,:,3);
[nx, ny, nz] = size(uXRaw);

% %% Rearrange x axis to z axis for interpolation
% % Initialization

uX         = zeros(ny,nz,nx);
uY         = zeros(ny,nz,nx);
uZ         = zeros(ny,nz,nx);

for iSlice = 1:nx
    uX(:,:,iSlice) = reshape(uXRaw(iSlice,1:ny,1:nz),ny,nz);
    uY(:,:,iSlice) = reshape(uYRaw(iSlice,1:ny,1:nz),ny,nz);
    uZ(:,:,iSlice) = reshape(uZRaw(iSlice,1:ny,1:nz),ny,nz);
end

% % QC Plots
% figure
% subplot(1,3,1)
% imagesc(uX(:,:,1))
% subplot(1,3,2)
% imagesc(uY(:,:,1))
% subplot(1,3,3)
% imagesc(uZ(:,:,1))




%% Interpolation to find location and flux of each streamline
% as of now each streamline is stored in grid (for matrix calculation)

% Initialization
streamlinesLocY = zeros(ny,nz,nx);
streamlinesLocZ = zeros(ny,nz,nx);
streamlinesuX   = zeros(ny,nz,nx);
streamlinesuY   = zeros(ny,nz,nx);
streamlinesuZ   = zeros(ny,nz,nx);

% Z axis -> X axis when plotted (start from top left)
[Z,Y] = meshgrid(1:nz,1:ny);


for jSlice = 1:nx
    tempuX = uX(:,:,jSlice);
    tempuY = uY(:,:,jSlice);
    tempuZ = uZ(:,:,jSlice);

    if jSlice == 1
```

```
        % Find new location (Beware of NaN)
        diffX = 1;
        diffY = (tempuY./tempuX);

        streamlinesLocY(:,:,1) = diffY + Y;
        streamlinesLocZ(:,:,1) = (tempuZ) ./ (sqrt(tempuX.^2 +
tempuY.^2)) ...
            .* (sqrt(diffX.^2 + diffY.^2)) + Z;

    else
        % Find new location (Beware of NaN)
        tempvX = streamlinesuX(:,:,jSlice - 1);
        tempvY = streamlinesuY(:,:,jSlice - 1);
        tempvZ = streamlinesuZ(:,:,jSlice - 1);

        diffX = 1;
        diffY = (tempvY ./ tempvX);
        streamlinesLocY(:,:,jSlice) = diffY +
streamlinesLocY(:,:,jSlice-1);
        streamlinesLocZ(:,:,jSlice) = (tempvZ) ./ (sqrt(tempvX.^2 +
tempvY.^2)) ...
            .* (sqrt(diffX.^2+diffY.^2)) ...
            + streamlinesLocZ(:,:,jSlice - 1);
    end

    % Interpolation to create velocity vector
    streamlinesuX(:,:,jSlice) = interp2(Z, Y, tempuX,
streamlinesLocZ(:,:,jSlice), ...
        streamlinesLocY(:,:,jSlice), 'linear');
    streamlinesuY(:,:,jSlice) = interp2(Z, Y, tempuY,
streamlinesLocZ(:,:,jSlice), ...
        streamlinesLocY(:,:,jSlice), 'linear');
    streamlinesuZ(:,:,jSlice) = interp2(Z, Y, tempuZ,
streamlinesLocZ(:,:,jSlice), ...
        streamlinesLocY(:,:,jSlice), 'linear');
end

% % QC Plots
% figure
% subplot(1,3,1)
% imagesc(streamlinesuX(:,:,1))
% subplot(1,3,2)
% imagesc(streamlinesuY(:,:,1))
% subplot(1,3,3)
% imagesc(streamlinesuZ(:,:,1))
%
% figure
% subplot(1,2,1)
% imagesc(streamlinesLocY(:,:,1))
% subplot(1,2,2)
% imagesc(streamlinesLocZ(:,:,1))

disp('Step 1: Interpolation')
```

```
%% Transform data into individual path
%Initialization
tempStreamlinesXYZ  = cell(ny*nz,1);
tempStreamlinesFlux = cell(ny*nz,1);
count = 1;

for j = 1:ny
    for k = 1:nz

        tempStreamlinesXYZ{count}(:,1) = [1:nx]';
        tempStreamlinesXYZ{count}(:,2) = streamlinesLocY(j,k,:);
        tempStreamlinesXYZ{count}(:,3) = streamlinesLocZ(j,k,:);

        tempStreamlinesFlux{count}(:,1)    = streamlinesuX(j,k,:);
        tempStreamlinesFlux{count}(:,2)    = streamlinesuY(j,k,:);
        tempStreamlinesFlux{count}(:,3)    = streamlinesuZ(j,k,:);

        count = count + 1;
    end
end

disp('Step 2: Extract individual paths')

%% Clean any path that contains NaN -> outside the boundary
if cleanPath
    nPath = numel(tempStreamlinesXYZ);
    count = 1;

    for i = 1:nPath
        if any(any(isnan(tempStreamlinesXYZ{i}))) == 0

            StreamlinesXYZ{count}  = tempStreamlinesXYZ{i};
            StreamlinesFlux{count} = tempStreamlinesFlux{i};
            StreamlinesAbsFlux{count} =
sqrt(tempStreamlinesFlux{i}(:,1).^2 ...
                + tempStreamlinesFlux{i}(:,2).^2 ...
                + tempStreamlinesFlux{i}(:,3).^2);
            count                  = count + 1;

        end
    end
else
    StreamlinesXYZ     = tempStreamlinesXYZ;
    StreamlinesFlux    = tempStreamlinesFlux;
end

disp('Step 3: Clean paths with NaN')

%% Find Tortuosity
% Initialization
nFlowPathClean = numel(StreamlinesXYZ);
totalDistance  = zeros(1, nFlowPathClean);
totalTime      = zeros(1, nFlowPathClean);
totalFlux      = zeros(1, nFlowPathClean);
```

```
for i = 1:numel(StreamlinesXYZ)

    pathX = StreamlinesXYZ{i}(:,1);
    pathY = StreamlinesXYZ{i}(:,2);
    pathZ = StreamlinesXYZ{i}(:,3);

    velocityX = StreamlinesFlux{i}(:,1);
    velocityY = StreamlinesFlux{i}(:,2);
    velocityZ = StreamlinesFlux{i}(:,3);

    % Initialization
    flowDistance    = zeros(size(pathX,1)-1, 1);
    flowAvgVelocity = zeros(size(pathX,1)-1, 1);
    flowTime        = zeros(size(pathX,1)-1, 1);

    for j = 1:size(pathX,1) - 1
        if and(any(StreamlinesXYZ{i}(j,:))   ~= 0, ...
                any(StreamlinesXYZ{i}(j+1,:)) ~= 0)

            flowDistance(j)  = sqrt( (pathX(j+1)-pathX(j)).^2 + ...
                (pathY(j+1)-pathY(j)).^2 + ...
                (pathZ(j+1)-pathZ(j)).^2 );

            flowAvgVelocity(j) ...
                = (sqrt(velocityX(j+1).^2 + velocityY(j+1).^2 + ...
                velocityZ(j+1).^2) + ...
                sqrt(velocityX(j).^2   + velocityY(j).^2   + ...
                velocityZ(j).^2))./2;

            flowTime(j)    = flowDistance(j)./flowAvgVelocity(j);

        end
    end

    % For each path
    totalDistance(1,i)  = sum(flowDistance);
    totalTime(1,i)      = sum(flowTime);
    totalFlux(1,i)      = sum(flowAvgVelocity);
end

tortuosity  = (totalDistance./(nx));

% Find the major statistics
tortuosityMin    = nanmin(tortuosity);
tortuosityMean   = nanmean(tortuosity);
tortuosityMax    = nanmax(tortuosity);
tortuosityFluxWeighted =
nansum(totalDistance.*totalFlux)./nansum(totalFlux)./nx;
tortuosityStd    = nanstd(tortuosity);

disp('Step 4: Calculate tortuosity')
```

```
end
```

```
function [ StreamlinesProximity2D, StreamlinesProximity3D, ...
    StreamlinesProximity2DNorm, StreamlinesProximity3DNorm, ...
    Proximity2DMin, Proximity2DMean, Proximity2DMax, Proximity2DStd,
...
    Proximity3DMin, Proximity3DMean, Proximity3DMax, Proximity3DStd]
...
    = computeStreamlines2D3DProximity( image3D, StreamlinesXYZ )
%computeStreamlines2D3DProximity extract proximity to the nearest
solid
%
%   Input Arguments
%   - image3D         : a (nx*ny*nz) uint8 matrix, 3-D binary image of
%                       pore space (0 = pore, 1 = grain)
%                           ---- or ----
%                       a cell array containing matrix as specified
above
%
%   - StreamlinesXYZ  : a cell array (nStreamline*1), x, y, z
locations of
%                       each streamline (output from
computeStreamlines)
%                           ---- or ----
%                       a cell array containing cell array as
specified above
%
%   Output Arguments
%   - StreamlinesProximity2D : a cell array (nStreamline*1) containing
%                              vector (nx*1) of nearest distance to
solid
%                              in 2-D slice.
%   - StreamlinesProximity3D : a cell array (nStreamline*1) containing
%                              vector (nx*1) of nearest distance to
solid
%                              in 3-D slice.
%   - StreamlinesProximity2DNorm : a cell array (nStreamline*1)
containing
%                              vector (nx*1) of nearest distance to
solid
%                              in 2-D slice normalized with the
maximum.
%   - StreamlinesProximity3DNorm : a cell array (nStreamline*1)
containing
%                              vector (nx*1) of nearest distance to
solid
%                              in 3-D slice normalized with the
maximum.
%   - Proximity2DMin  : a vector (nStreamlines*1) containing a minimum
value
%                       of 2D proximity along each streamline
%   - Proximity2DMean : a vector (nStreamlines*1) containing a mean
value
%                       of 2D proximity along each streamline
%   - Proximity2DMax  : a vector (nStreamlines*1) containing a maximum
value
%                       of 2D proximity along each streamline
%   - Proximity2DStd  : a vector (nStreamlines*1) containing a std
value
%                       of 2D proximity along each streamline
%   - Proximity3DMin  : a vector (nStreamlines*1) containing a minimum
value
%                       of 3D proximity along each streamline
%   - Proximity3DMean : a vector (nStreamlines*1) containing a mean
value
%                       of 3D proximity along each streamline
%   - Proximity3DMax  : a vector (nStreamlines*1) containing a maximum
value
%                       of 3D proximity along each streamline
%   - Proximity3DStd  : a vector (nStreamlines*1) containing a std
value
%                       of 3D proximity along each streamline

%   Revision 1: April    2016 Nattavadee Srisutthiyakorn
%   Stanford Rock Physics and Borehole Geophysics Project (SRB)




%% Program

if iscell(image3D)
    % Initialization
    nImage                  = length(image3D);
    for iImage = 1:nImage

        disp(['Current image:
(',num2str(iImage),'/',num2str(nImage),')'])

        try
            [ StreamlinesProximity2D{iImage},
StreamlinesProximity3D{iImage}, ...
                StreamlinesProximity2DNorm{iImage},
StreamlinesProximity3DNorm{iImage},...
                Proximity2DMin{iImage}, Proximity2DMean{iImage}, ...
                Proximity2DMax{iImage}, Proximity2DStd{iImage}, ...
                Proximity3DMin{iImage}, Proximity3DMean{iImage}, ...
                Proximity3DMax{iImage}, Proximity3DStd{iImage}] ...
                = computeSTL2D3DProximity( image3D{iImage},
StreamlinesXYZ{iImage});
        end
    end

else
    [ StreamlinesProximity2D, StreamlinesProximity3D, ...
        StreamlinesProximity2DNorm, StreamlinesProximity3DNorm, ...
        Proximity2DMin, Proximity2DMean, Proximity2DMax,
Proximity2DStd, ...
        Proximity3DMin, Proximity3DMean, Proximity3DMax,
Proximity3DStd] ...
```

```
        = computeSTL2D3DProximity( image3D, StreamlinesXYZ );
end


end




function [ StreamlinesProximity2D, StreamlinesProximity3D, ...
    StreamlinesProximity2DNorm, StreamlinesProximity3DNorm, ...
    Proximity2DMin, Proximity2DMean, Proximity2DMax, Proximity2DStd,
...
    Proximity3DMin, Proximity3DMean, Proximity3DMax, Proximity3DStd]
...
    = computeSTL2D3DProximity( image3D, StreamlinesXYZ )

% Rotate matrix into the the flow along x direction
[nx, ny, nz] = size(image3D);
for iSlice = 1:nx
    image3DRot(:,:,iSlice) = reshape(image3D(iSlice,1:ny,1:nz),ny,nz);
end

% Find the distance matrix in 2D, 3D
Proximity3D = bwdist(image3DRot,'euclidean');
Proximity2D = zeros(ny, nz, nx);

for iSlice = 1:nx
    image2D = image3DRot(:,:,iSlice);
    Proximity2D(:,:,iSlice) = bwdist(image2D,'euclidean');
end

% Find the distance from the flow path
nStreamline = length(StreamlinesXYZ);

for iStreamline = 1:nStreamline
    % Initialization
    x = StreamlinesXYZ{iStreamline}(:,1);
    y = StreamlinesXYZ{iStreamline}(:,2);
    z = StreamlinesXYZ{iStreamline}(:,3);

    for iSlice = 1:nx
        StreamlinesProximity2D{iStreamline}(iSlice) ...
            =
Proximity2D(round(y(iSlice)),round(z(iSlice)),round(x(iSlice)));
        StreamlinesProximity3D{iStreamline}(iSlice) ...
            =
Proximity3D(round(y(iSlice)),round(z(iSlice)),round(x(iSlice)));
    end

    % Analyze the statistics of proximity
    Proximity2DMin(iStreamline)     =
min(StreamlinesProximity2D{iStreamline});
    Proximity2DMean(iStreamline)    =
mean(StreamlinesProximity2D{iStreamline});
```

```
    Proximity2DMax(iStreamline)      =
max(StreamlinesProximity2D{iStreamline});
    Proximity2DStd(iStreamline)      =
std(StreamlinesProximity2D{iStreamline});
    StreamlinesProximity2DNorm{iStreamline} ...
        =
StreamlinesProximity2D{iStreamline}./Proximity2DMax(iStreamline);

    Proximity3DMin(iStreamline)      =
min(StreamlinesProximity3D{iStreamline});
    Proximity3DMean(iStreamline)     =
mean(StreamlinesProximity3D{iStreamline});
    Proximity3DMax(iStreamline)      =
max(StreamlinesProximity3D{iStreamline});
    Proximity3DStd(iStreamline)      =
std(StreamlinesProximity3D{iStreamline});
    StreamlinesProximity3DNorm{iStreamline} ...
        =
StreamlinesProximity3D{iStreamline}./Proximity3DMax(iStreamline);

end


end
```

```
function [ PSD ] = modelPSD( nX, yMin, yMax)
%modelPSD create different model of pore size distribution



% model 1 straight line
PSD{1} = linspace(yMin, yMax, nX);

% model 2 sinusoidal
xRange = linspace(3*pi/2, 5*pi/2, nX);
yEqn   = sin(xRange);
% transpose the equation to the specified range
PSD{2} = ((yMax-yMin)/(max(yEqn) - min(yEqn))*(yEqn - min(yEqn))) +
yMin;

% model 3 gauss error equation
xRange = linspace(-2,2,nX);
yEqn   = erf(xRange);
% transpose the equation to the specified range
PSD{3} = ((yMax-yMin)/(max(yEqn) - min(yEqn))*(yEqn - min(yEqn))) +
yMin;

% model 4 sinusoidal
xRange = linspace(2*pi/2, 4*pi/2, nX);
yEqn   = sin(xRange);
% transpose the equation to the specified range
PSD{4} = -(PSD{2} - PSD{1}) + PSD{1};
```

```
% model 5 gauss err flip curvature
PSD{5} = -(PSD{3} - PSD{1}) + PSD{1};

end
```

```
function [ KCcorrection, phi_ssa ] = computeKCcorrection( PSD )
%computeKCcorrection compute the correction for the Kozeny-Carman
equation
%   Input Argument
%   - PSD          : a vector (nVoxel), the pore size distribution
along
%                    the flow path
%   Output Argument
%   - KCcorrection : a double, the value needed for correction from
%                    Kozeny-Carman permeability to Lattice Boltzmann
%                    Permeability
%   Note



%% Program
% Initialization
PSD  = PSD(:);
% Remove the number 0
PSD = PSD(PSD ~= 0);
nPSD = length(PSD);

% compute the approximate surface area using a frustum formula
for iPSD = 2:nPSD
    r1 = PSD(iPSD-1);
    r2 = PSD(iPSD);
    saSlice(iPSD-1) = pi.*(r2 + r1).*sqrt((r2-r1).^2 + 1^2);
    % The discretization is 1 voxel at a time
end

pv = pi.*sum(PSD.^2);        % Pore volume
sa = sum(saSlice);           % Surface Area

phi_ssa = pv/sa;             % Pore volume/Surface area ratio

% calculate the hydraulic radius
rH = 2.*pv./sa;

% calculate the equivalent radius of a circular pipe that has the same
% porosity
rC = sqrt(sum(PSD.^2)/nPSD);

% calculate the apparent radius (same definition as the Kozeny-Carman)
rA = sqrt(rH.*rC);

% calculate permeability ratio
% Definition:
% permTheo = pi*L/(8*A*l*sum(1./(rr*dl).^4))*m2tomD;
% permKC = pi.*(rA.*dl).^4./(8.*A).*m2tomD;

permTheo = nPSD./(sum(1./PSD.^4));
permKC = (rA).^4;

% the correction
KCcorrection = permTheo./permKC;



end
```

```
function [ grainCentroid, grainRadius, grainAzimuth, grainInclination,
...
           grainVolume, nContact, grainSurfaceArea ] ...
    = computeGSD( image, minThres, bc, qcPlot )
%computeGSD compute grain size distribution
%   Input Arguments
%   - image       : an (nx*ny) or (nx*ny*nz) uint8 matrix, 2-D or 3-D
binary
%                   image of porespace (1 = grain, 0 = pore)
%   - minThres    : an integer, a threshold to suppress all minima in
the
%                   intensity image whose depth is less than this
number
%   - bc          : an integer, boundary condition
%                   0: impose no boundary condition
%                   1: remove grains that are close to the boundary
%   - qcPlot      : an integer, plot the QC image
%                   1: show plot
%
%   Output Arguments
%   - grainCentroid    : a (nGrain*2) or (nGrain*3) integer matrix, xy
or xyz
%                         location of the grain in voxel
%   - grainRadius      : a (nGrain*4) or (nGrain*6) double matrix,
radius
%                         of each grain in voxel. The vector is based
on the
%                         principal component analysis
%   - grainAzimuth     : a (nGrain*4) or (nGrain*6) double matrix,
%                         azimuth on each axis of radius based on
%                         spherical cooridnate (ISO physics).
%   - grainInclination : (For 3D), a (nGrain*6) double matrix,
%                         inclination on each axis of radius based on
%                         spherical cooridnate (ISO physics).
%   - grainVolume      : a (nGrain*1) vector, the volume of grain in
voxel
%   - nContact         : a (nGrain*1) vector, the number of contact of
each
%                         grain
%   - grainSurfaceArea : a (nGrain*1) vector, the surface area of
grain in voxel
```

```
%   Revision 3: Nov 2017 Natt Srisutthiyakorn - Add spherical
%                                              coordinate/qc3D
%   Revision 2: Oct 2017 Natt Srisutthiyakorn - Add grain contact




%% Program
% Default parameters
if nargin < 2
    minThres    = 1;
    bc          = 1;
    qcPlot      = 0;
end

% Determine whether it is 2-D or 3-D image
[imSize(1), imSize(2), imSize(3)]    = size(image);




%% Watershed algorithm
% Create the distance map image (100+ s for 1024^3 voxels)
disp('Step 1: Create distance map - find the distance of each point to
the nearest solid')
imageDistGrain          = bwdist(~image);
imageDistGrain          = -imageDistGrain;
imageDistGrain(~image)  = -Inf;

% Impose height threshold (500+ s for 1024^3 voxels) if applicable
if minThres > 1
    imageDistGrain = imhmin(imageDistGrain,minThres);
end

% Watershed algorithm (3500+ s for 1024^3 voxels)
disp('Step 2: Apply watershed algorithm')
imageGrainIdx = watershed(imageDistGrain);

% QC pore (component 0 is boundary and now pore)
imageGrainIdx(~image) = 0;

% Find region properties (100+ s for 1024^3 voxels)
disp('Step 3: Find center of mass and volume')
stats       = regionprops('table', imageGrainIdx, 'Centroid', 'Area');
componentNo = unique(imageGrainIdx);




% Get grain properties (exclude 0)
grainNo             = componentNo(componentNo ~= 0);
grainNo             = grainNo(grainNo ~= 1);
grainNo             = grainNo(grainNo ~= 2); % Noise in 1-2?
nGrain              = length(grainNo);
allGrainCentroid    = round(stats.Centroid(grainNo,:));
allGrainVolume      = round(stats.Area(grainNo,:));
```

```
nTotalSize          = length(imageGrainIdx(:));
if imSize(3) == 1     % 2D--------------------------------------------
-----
    grainRadius      = zeros(nGrain,4);
    grainAzimuth     = zeros(nGrain,4);
    grainInclination = zeros(nGrain,4);
elseif imSize(3) > 1  % 3D--------------------------------------------
-----
    grainRadius      = zeros(nGrain,6);
    grainAzimuth     = zeros(nGrain,6);
    grainInclination = zeros(nGrain,6);
end

% QC
%imageGrainIdx(:,:,1)
%[vol, idx] = max(allGrainVolume)

% QC Plot
if qcPlot
    if imSize(3) == 1 % 2D--------------------------------------------
-----
        figure
        subplot(1,3,1)
        imagesc(image);
        axis equal; xlim([0 imSize(2)]); ylim([0 imSize(1)]);
        colormap(flipud(gray))
        title('Original')

        subplot(1,3,2)
        imagesc(imageGrainIdx)
        title('Watershed')
        axis equal; xlim([0 imSize(2)]); ylim([0 imSize(1)]);

        subplot(1,3,3)
        imagesc(imageGrainIdx)
        title('Measurement')
        axis equal; xlim([0 imSize(2)]); ylim([0 imSize(1)]);
    elseif imSize(3) > 1  % 3D----------------------------------------
-----
        figure
        colors = jet(nGrain);
        colors = colors(randperm(length(colors)),:); % Randomize the
color

        for iGrain = 1:nGrain
            idxSingleGrain  = (imageGrainIdx(:) == grainNo(iGrain));
            imageGrain      = zeros(nTotalSize,1);
            imageGrain(idxSingleGrain) = 1;
            imageGrain      = reshape(imageGrain,[imSize(1),
imSize(2), imSize(3)]);

            fv = isosurface(imageGrain,0);
            patch(fv,'EdgeColor','none','facecolor',colors(iGrain,:));
            box on;
            view(45,45);axis equal
            alpha(0.2)
            hold on
        end
    end
end




%% Find the surface area of each grain using the isosurface
grainSurfaceArea = zeros(nGrain,1);
tic
for iGrain = 1:nGrain
    if iGrain/100 == round(iGrain/100)
    disp(['Step 4: Measure grain surface area (', num2str(iGrain),...
        '/',num2str(nGrain), ')'])
    end
    idxSingleGrain      = (imageGrainIdx(:) == grainNo(iGrain));
    imageGrain    = zeros(nTotalSize,1);
    imageGrain(idxSingleGrain) = 1;

    if imSize(3) > 1
        imageGrain      = reshape(imageGrain,[imSize(1), imSize(2),
imSize(3)]);
        % Extract the surface surface
        fv = isosurface(imageGrain,0);
        % p =  patch(fv,'facecolor','cyan','EdgeColor','none'); - QC
plots
        % verts = get(p, 'Vertices');
        % faces = get(p, 'Faces');
        % close all;

        % Find the surface area
        vertices = fv.vertices;
        faces = fv.faces;
        a = vertices(faces(:, 2), :) - vertices(faces(:, 1), :);
        b = vertices(faces(:, 3), :) - vertices(faces(:, 1), :);
        c = cross(a, b, 2);
        grainSurfaceArea(iGrain) = 1/2 * sum(sqrt(sum(c.^2, 2)));
    end

end
toc




%% Find the number of contact
nContact = zeros(nGrain,1);

for iGrain = 1:nGrain
    if iGrain/100 == round(iGrain/100)
    disp(['Step 5: Measure number of contact(', num2str(iGrain),...
        '/',num2str(nGrain), ')'])
    end
    idxSingleGrain      = (imageGrainIdx(:) == grainNo(iGrain));
    imageGrain    = zeros(nTotalSize,1);
    imageGrain(idxSingleGrain) = 1;

    if imSize(3) == 1
        imageGrain    = reshape(imageGrain,[imSize(1), imSize(2)]);
        % Dilate the grain first to get the outer boundary.
        imageGrain = bwmorph(imageGrain,'dilate');
        imageGrain = bwmorph(imageGrain,'dilate');
        % Find boundary of a grain
        grainBound  = bwboundaries(imageGrain,'noholes');
        linearInd   = sub2ind([imSize(1),imSize(2)],
grainBound{1}(:,1), grainBound{1}(:,2));
        contactIdx  = unique(imageGrainIdx(linearInd));
    elseif imSize(3) > 1
        contactIdx      = [];
        imageGrain      = reshape(imageGrain,[imSize(1), imSize(2),
imSize(3)]);
        for iz = 1:imSize(3)
            % Dilate the grain first to get the outer boundary.
            imageGrain(:,:,iz) = bwmorph(imageGrain(:,:,iz),'dilate');
            imageGrain(:,:,iz) = bwmorph(imageGrain(:,:,iz),'dilate');
            % Find boundary of a grain
            grainBound  = bwboundaries(imageGrain(:,:,iz),'noholes');
            if length(grainBound) == 1
                linearInd   = sub2ind([imSize(1),imSize(2)],
grainBound{1}(:,1), grainBound{1}(:,2));
                contactIdx  = [contactIdx; imageGrainIdx(linearInd)];
            end
        end
    end

    % Find the unique index that is not 0 (the pore) and the itself.
    contactIdx  = unique(contactIdx);
    contactIdx  = contactIdx(contactIdx ~= 0);
    contactIdx  = contactIdx(contactIdx ~= grainNo(iGrain));
    nContact(iGrain) = length(contactIdx);
end



%% Measure the grain size
for iGrain = 1:nGrain
    if iGrain/100 == round(iGrain/100)
    disp(['Step 6: Measure grain size (', num2str(iGrain),...
        '/',num2str(nGrain), ')'])
    end
    idxSingleGrain      = (imageGrainIdx(:) == grainNo(iGrain));
    imageGrain    = zeros(nTotalSize,1);
    imageGrain(idxSingleGrain) = 1;


    if imSize(3) == 1 % 2D-------------------------------------------
-----
        imageGrain    = reshape(imageGrain,[imSize(2), imSize(1)]);
        [idxXX, idxYY] = find(imageGrainIdx == grainNo(iGrain));
        % Obtain principal direction
        [coeff] = pca([idxXX idxYY]);

        nVec = size(coeff,2);
        X0 = allGrainCentroid(iGrain,1);
        Y0 = allGrainCentroid(iGrain,2);

        for iVec = 1:nVec
            clear temp*

            % Find index of a straight line in both direction on
principal axis
            a = coeff(1,iVec);
            b = coeff(2,iVec);

            if a >= b
                tempX1 = [X0:imSize(2)]';
                tempX2 = [X0:-1:1]';
                nL1    = ones(length(tempX1),1);
                nL2    = ones(length(tempX2),1);
                tempY1 = nL1.*round(b./a.*(tempX1-X0) + Y0);
                tempY2 = nL2.*round(b./a.*(tempX2-X0) + Y0);
            elseif b > a
                tempY1 = [Y0:imSize(1)]';
                tempY2 = [Y0:-1:1]';
                nL1    = ones(length(tempY1),1);
                nL2    = ones(length(tempY2),1);
                tempX1 = nL1.*round(a./b.*(tempY1-Y0) + X0);
                tempX2 = nL2.*round(a./b.*(tempY2-Y0) + X0);
            end

            % Initialization of lines for measurement
            nLine1 = length(tempX1);
            nLine2 = length(tempX2);
            tempLine1 = zeros(nLine1,1);
            tempLine2 = zeros(nLine2,1);

            for iLine1 = 1:nLine1
                try % instead of checking boundary
                tempLine1(iLine1,1) = imageGrain(tempY1(iLine1),...

tempX1(iLine1));
                end
            end

            for iLine2 = 1:nLine2
                try
                tempLine2(iLine2,1) = imageGrain(tempY2(iLine2),...

tempX2(iLine2));
                end
            end

            % Find the boundary wihtin the lines
            bound1 = find(tempLine1 == 0, 1, 'first');
            if isempty(bound1)
                bound1 = nLine1;
            end

            bound2 = find(tempLine2 == 0, 1, 'first');
            if isempty(bound2)
                bound2 = nLine2;
            end

            % Calculate grain size to the boundary
            dX1 = tempX1(bound1) - tempX1(1);
            dY1 = tempY1(bound1) - tempY1(1);
            dX2 = tempX2(bound2) - tempX2(1);
            dY2 = tempY2(bound2) - tempY2(1);

            % Compute 2 radius from PCA at a time
            grainRadius(iGrain,2.*(iVec-1) + 1) ...
                = sqrt(dX1^2 + dY1^2);
            grainRadius(iGrain,2.*(iVec-1) + 2) ...
                = sqrt(dX2^2 + dY2^2);

            grainAzimuth(iGrain,2.*(iVec-1) + 1) ...
                = atan(dX1/dY1)*180/pi;
            grainAzimuth(iGrain,2.*(iVec-1) + 2) ...
                = atan(dX2/dY2)*180/pi;

            if qcPlot
                if imSize(3) == 1
                hold on

plot(tempX1(1:bound1),tempY1(1:bound1),'r','LineWidth',2)

plot(tempX2(1:bound2),tempY2(1:bound2),'r','LineWidth',2)
                end
            end

        end


    elseif imSize(3) > 1 % 3D----------------------------------------
-----
        imageGrain    ...
            = reshape(imageGrain,[imSize(1), imSize(2), imSize(3)]);

        [idxXX, idxYY, idxZZ] = find(imageGrainIdx ==
grainNo(iGrain));

        % principal component analysis to get principal direction
(each column
        % = one principal component
        [coeff] = pca([idxXX idxYY idxZZ]);
        nVec = size(coeff,2);

        % Centroid
        X0 = allGrainCentroid(iGrain,1);
        Y0 = allGrainCentroid(iGrain,2);
        Z0 = allGrainCentroid(iGrain,3);

        for iVec = 1:nVec
            clear temp*

            % Obtain the value of secondary and tertiary direction
            a = coeff(1,iVec);
            b = coeff(2,iVec);
            c = coeff(3,iVec);

            [~,idxMax] = max([a,b,c]);

            if idxMax == 1;
                tempX1 = [X0:imSize(1)]';
                tempX2 = [X0:-1:1]';
                nL1    = ones(length(tempX1),1);
                nL2    = ones(length(tempX2),1);
                tempY1 = nL1.*round(b./a.*(tempX1-X0) + Y0);
                tempY2 = nL2.*round(b./a.*(tempX2-X0) + Y0);
                tempZ1 = nL1.*round(c./a.*(tempX1-X0) + Z0);
                tempZ2 = nL2.*round(c./a.*(tempX2-X0) + Z0);
            elseif idxMax == 2;
                tempY1 = [Y0:imSize(2)]';
                tempY2 = [Y0:-1:1]';
                nL1    = ones(length(tempY1),1);
                nL2    = ones(length(tempY2),1);
                tempX1 = nL1.*round(a./b.*(tempY1-Y0) + X0);
                tempX2 = nL2.*round(a./b.*(tempY2-Y0) + X0);
                tempZ1 = nL1.*round(c./b.*(tempY1-Y0) + Z0);
                tempZ2 = nL2.*round(c./b.*(tempY2-Y0) + Z0);
            elseif idxMax == 3;
                tempZ1 = [Z0:imSize(3)]';
                tempZ2 = [Z0:-1:1]';
                nL1    = ones(length(tempZ1),1);
                nL2    = ones(length(tempZ2),1);
                tempX1 = nL1.*round(a./c.*(tempZ1-Z0) + X0);
                tempX2 = nL2.*round(a./c.*(tempZ2-Z0) + X0);
                tempY1 = nL1.*round(c./c.*(tempZ1-Z0) + Y0);
                tempY2 = nL2.*round(c./c.*(tempZ2-Z0) + Y0);
            end

            % Initialization of lines for measurement
            nLine1 = length(tempX1);
            nLine2 = length(tempX2);
            tempLine1 = zeros(nLine1,1);
            tempLine2 = zeros(nLine2,1);

            for iLine1 = 1:nLine1
                try % instead of checking boundary
                    tempLine1(iLine1,1) ...
                        = imageGrain(tempY1(iLine1),...
                                     tempX1(iLine1),...
                                     tempZ1(iLine1));
                end
            end

            for iLine2 = 1:nLine2
                try
                    tempLine2(iLine2,1) ...
                        = imageGrain(tempY2(iLine2),...
                                     tempX2(iLine2),...
                                     tempZ2(iLine2));
                end
            end

            % Find the boundary wihtin the lines
            bound1 = find(tempLine1 == 0, 1, 'first');
            if isempty(bound1)
                bound1 = nLine1;
            end

            bound2 = find(tempLine2 == 0, 1, 'first');
            if isempty(bound2)
                bound2 = nLine2;
            end

            % Calculate grain size to the boundary
            dX1 = tempX1(bound1) - tempX1(1);
            dY1 = tempY1(bound1) - tempY1(1);
            dZ1 = tempZ1(bound1) - tempZ1(1);
            dX2 = tempX2(bound2) - tempX2(1);
            dY2 = tempY2(bound2) - tempY2(1);
            dZ2 = tempZ2(bound2) - tempZ2(1);

            grainRadius(iGrain,2.*(iVec-1) + 1) ...
                = sqrt((dX1)^2 + (dY1)^2 + (dZ1)^2);
            grainRadius(iGrain,2.*(iVec-1) + 2) ...
                = sqrt((dX2)^2 + (dY2)^2 + (dZ2)^2);

            grainAzimuth(iGrain,2.*(iVec-1) + 1) ...
                = acos(dZ1/grainRadius(iGrain,2.*(iVec-1) + 1));
            grainAzimuth(iGrain,2.*(iVec-1) + 2) ...
                = acos(dZ2/grainRadius(iGrain,2.*(iVec-1) + 2));

            grainInclination(iGrain,2.*(iVec-1) + 1) ...
                = atan(dY1/dX1)*180/pi;
            grainInclination(iGrain,2.*(iVec-1) + 2) ...
                = atan(dY2/dX2)*180/pi;

        end
    end
end



%% Output
% Clean 0 data
```

```
idxNonZero           = all(grainRadius,2);
grainRadius          = grainRadius(idxNonZero,:);
grainCentroid        = allGrainCentroid(idxNonZero,:);
grainVolume          = allGrainVolume(idxNonZero,:);
grainAzimuth         = grainAzimuth(idxNonZero,:);
grainInclination     = grainInclination(idxNonZero,:);
nContact             = nContact(idxNonZero,:);
grainSurfaceArea     = grainSurfaceArea(idxNonZero,:);

%% Boundary condition Exclude grains at the boundary
if bc
    radius       = max(grainRadius,[],2);
    nGrain       = length(radius);
    idxAccept    = [];
    idxReject    = [];

    if imSize(3) == 1 %2D-------------------------------------------------
---------
        % Define all the corner points
        c1 = [1, 1];
        c2 = [imSize(1), 1];
        c3 = [1, imSize(1)];
        c4 = [imSize(1), imSize(1)];

        % Calculate the distance from a point to a line for each
centroid
        for iGrain = 1:nGrain
            p = grainCentroid(iGrain,:);
            distanceLine1 = abs(det([c2 - c1; p - c2])/sqrt(sum((c2 -
c1).^2)));
            distanceLine2 = abs(det([c3 - c1; p - c3])/sqrt(sum((c3 -
c1).^2)));
            distanceLine3 = abs(det([c4 - c3; p - c3])/sqrt(sum((c4 -
c3).^2)));
            distanceLine4 = abs(det([c4 - c2; p - c2])/sqrt(sum((c4 -
c2).^2)));
            if (distanceLine1 > radius(iGrain) && distanceLine2 >
radius(iGrain) &&...
                    distanceLine3 > radius(iGrain) && distanceLine4 >
radius(iGrain))
                idxAccept = [idxAccept, iGrain];
            else
                idxReject = [idxReject, iGrain];
            end

        end

    elseif imSize(3) > 1 %3D-------------------------------------------
---------
        % Define the corner points
        corner = [1 1 1;
                  1 imSize(1) 1;
                  1 1 imSize(1);
                  1 imSize(1) imSize(1);
                  imSize(1) 1 1;
                  imSize(1) imSize(1) 1;
                  imSize(1) 1 imSize(1);
                  imSize(1) imSize(1) imSize(1)];

        % Define 6 different planes from corners point
        plane = [1 2 3;
                 1 3 5;
                 3 4 7;
                 2 4 6;
                 1 2 5;
                 5 6 7];
        % Calculate the distance from a point to a line for each
centroid
        for iGrain = 1:nGrain
            point = grainCentroid(iGrain,:);

            for iPlane = 1:6
                c1 = corner(plane(iPlane,1),:);
                c2 = corner(plane(iPlane,2),:);
                c3 = corner(plane(iPlane,3),:);

                normal = cross(c1 - c2, c1 - c3);

                d = dot(normal, c1);

                % Find the closest distance from a point to plane
                distance(iGrain,iPlane) = (dot(normal, point) -
d)./sqrt(dot(normal,normal));
            end

            if min(abs(distance(iGrain,:))) > radius(iGrain)
                idxAccept = [idxAccept, iGrain];
            else
                idxReject = [idxReject, iGrain];
            end

        end

    end

    % Screen the data to the number
    grainRadius         = grainRadius(idxAccept,:);
    grainCentroid       = grainCentroid(idxAccept,:);
    grainVolume         = grainVolume(idxAccept,:);
    grainAzimuth        = grainAzimuth(idxAccept,:);
    grainInclination    = grainInclination(idxAccept,:);
    nContact            = nContact(idxAccept,:);
    grainSurfaceArea    = grainSurfaceArea(idxAccept,:);


    if qcPlot
        if imSize(3) == 1
        scatter(grainCentroid(:,1), grainCentroid(:,2),'go')
        end
    end
```

```
end
```

```
function [grainDiameter] = computeGrainDiameter(grainRadius,
measureOption)
%computeGrainDiameter compute grain diameter
%   Input Arguments
%   - grainRadius   : a (nGrain*6) or (nGrain*4) double matrix,
%                     radius of each grain [r1 r2 r3 r4 r5 r6] or
%                     [r1 r2 r3 r4] in Micron
%   - measureOption : a string, compute the distribution using
%                     "max" grain size
%                     "mean" grain size
%
%   Output Arguments
%   - grainDiameter : a (nGrain*1) double matrix,
%                     a max diameter of each grain

%   Revision 1: May 2018 Nattavadee Srisutthiyakorn




%%
if (~exist('measureOption', 'var'))
    measureOption = "max";
end




[~, nRadius]= size(grainRadius);
% Add to get the grain diameter instead of the radius.
if nRadius == 6
    grainDiameter(:,1) = grainRadius(:,1) + grainRadius(:,2);
    grainDiameter(:,2) = grainRadius(:,3) + grainRadius(:,4);
    grainDiameter(:,3) = grainRadius(:,5) + grainRadius(:,6);
elseif nRadius == 4
    grainDiameter(:,1) = grainRadius(:,1) + grainRadius(:,2);
    grainDiameter(:,2) = grainRadius(:,3) + grainRadius(:,4);
end

% Use the find the average grain size or max grain size for sieving
if measureOption == "mean"
    grainDiameter   = mean(grainDiameter,2);
elseif measureOption == "max"
    grainDiameter   = max(grainDiameter,[],2);
end



end
```

```
function [histFB, binCenter, statGSD] ...
```

```
    = computeHistFB( grainProp, binEdge )
%computeHistFB compute frequency-based histogram of the different
properties of GSD
%   Input Arguments
%   - grainProp     : a (nGrain*1) double vector, grain properties
such as
%                     maximum diameter, nContact, and etc.
%   - binEdge       : (optional) a vector, histogram bin edge
%
%   Output Arguments
%   - histFB        : a (nBin*1) vector, probability distribution
function (PDF)
%   - binCenter     : a (nBin*1) vector, bin center in mm as specified
in the code
%   - statGSD       : a struct, containing grain size distribution
%                     statistics
%
%   Notes
%   - Beware that the output from computeGSD is the voxel,
%     please multiply by the resolution in micron for maximum grain
radius and
%     multiply by resolution.^3 for grain volume
%     grainProp       = grainProp.*dx./1000;

%   Revision 1: May 2018 Nattavadee Srisutthiyakorn

%% Program
if (~exist('binEdge', 'var'))
    % Prefixed Histogram bin in mm due to the plot nature in log scale
    % for grain diameter
    binEdge = fliplr([4.0000 3.3600 2.8300 2.3800 2.0000 1.6800 1.4100
1.1900 1.0000 ...
           0.8500 0.7100 0.6000 0.5000 0.4200 0.3500 0.2970 0.2500
0.2100 ...
           0.1770 0.1490 0.1250 0.1050 0.0880 0.0740 0.0620 0.0530
0.0440 ...
           0.0370 0.0310 0.0260 0.0220 0.0190 0.0160 0.0130 0.0110
0.0093 ...
           0.0078 0.0065 0.0055 0.0046 0.0039 0.0033 0.0028 0.0023
0.0019 ...
           0.0016 0.0014 0.0012 0.0010]);
end

% Sort data into histogram bin
nEdge           = length(binEdge);
nBin            = nEdge - 1;
histFB          = zeros(nEdge - 1, 1);
binCenter(1)    = binEdge(1);

for iEdge = 1:nEdge - 1
    [index{iEdge}] = find(and(grainProp <= binEdge(iEdge + 1),...
                            grainProp  >  binEdge(iEdge)));
    binCenter(iEdge)   = (binEdge(iEdge + 1) + binEdge(iEdge))./2;
end

for iBin = 1:nBin
```

```
    histFB(iBin)   = length(index{iBin});
end

% Normalized to get the PDF
nGrainCheck = sum(histFB);
histFB = histFB./nGrainCheck;

% Export the statistics
statGSD.mean = mean(grainProp);
statGSD.std  = std(grainProp);

% Find the cumulative density function
% histCDF = cumsum((histFB))./sum(histFB);
```

```
function [histVB, binCenter, statGSD] ...
    = computeHistVB( grainProp, grainVolume, binEdge )
%computeHistVB computes volume-based histogram of the different
properties of GSD
%   Input Arguments
%   - grainProp     : a (nGrain*1) double vector, grain properties
such as
%                     maximum diameter, nContact, and etc.
%   - grainVolume   : a (nGrain*1) double vector, volume of each grain
or
%                     other weight
%                     *** Beware that the output from computeGSD is
the
%                     voxel, please multiply grainVolume*resolution.^3
%   - binEdge       : (optional) a vector, histogram bin edge
%
%   Output Arguments
%   - histVB        : a (nBin*1) vector, probability distribution
function (PDF)
%   - binCenter     : a (nBin*1) vector, bin center in mm as specified
in the code
%   - statGSD       : a struct, containing grain size distribution
%                     statistics
%
%   Notes
%   - Beware that the output from computeGSD is the voxel,
%     please multiply by the resolution in micron for maximum grain
radius and
%     multiply by resolution.^3 for grain volume
%     grainProp       = grainProp.*dx./1000;

%   Revision 1: May 2018 Nattavadee Srisutthiyakorn

%% Program
if (~exist('binEdge', 'var'))
    % Prefixed Histogram bin in mm due to the plot nature in log scale
    % for grain diameter
    binEdge = fliplr([4.0000 3.3600 2.8300 2.3800 2.0000 1.6800 1.4100
1.1900 1.0000 ...
```

```
            0.8500 0.7100 0.6000 0.5000 0.4200 0.3500 0.2970 0.2500
0.2100 ...
            0.1770 0.1490 0.1250 0.1050 0.0880 0.0740 0.0620 0.0530
0.0440 ...
            0.0370 0.0310 0.0260 0.0220 0.0190 0.0160 0.0130 0.0110
0.0093 ...
            0.0078 0.0065 0.0055 0.0046 0.0039 0.0033 0.0028 0.0023
0.0019 ...
            0.0016 0.0014 0.0012 0.0010]);
end

% Sort data into histogram bin
nEdge           = length(binEdge);
nBin            = nEdge - 1;
histVB          = zeros(nEdge - 1, 1);
binCenter(1)    = binEdge(1);

for iEdge = 1:nEdge - 1
    [index{iEdge}] = find(and(grainProp <= binEdge(iEdge + 1),...
                             grainProp  >  binEdge(iEdge)));
    binCenter(iEdge)   = (binEdge(iEdge + 1) + binEdge(iEdge))./2;
end

% Find the weight/volume/area of each bin in order to plot the
percentage
for iBin = 1:nBin
    histVB(iBin)  = sum(grainVolume(index{iBin}))./sum(grainVolume);
end

% Normalized to get the PDF
nGrainCheck = sum(histVB);
histVB = histVB./nGrainCheck;

% Export the statistics
statGSD.mean = sum(grainProp.*grainVolume)./sum(grainVolume);
statGSD.std  = std(grainProp, grainVolume);

% Find the cumulative density function
% histCDF = cumsum((histVF))./sum(histVF);
```

```
function [histPC, binCenter, statGSD, idxSelectedGrain] ...
    = computeHistPC( grainProp, grainCentroid, gridPC, binEdge )
%computeHistPC compute frequency-based histogram of the different
properties of GSD
%   Input Arguments
%   - grainProp     : a (nGrain*1) double vector, grain properties
such as
%                     maximum diameter, nContact, and etc.
%   - grainCentroid : a (nGrain*2) or (nGrain*3) integer matrix, xy or
xyz
%                     location of the grain in voxel
%   - gridPC        : (optional) a vector, for creating a grid for
point
%                     count
%   - binEdge       : (optional) a vector, histogram bin edge
%
%   Output Arguments
%   - histFB        : a (nBin*1) vector, probability distribution
function (PDF)
%   - binCenter     : a (nBin*1) vector, bin center in mm as specified
in the code
%   - statGSD       : a struct, containing grain size distribution
%                     statistics
%   - idxSelectedGrain : a (nSelectedGrains*1), the index of the grain
that
%                        is closest to the grid.
%
%   Notes
%   - Beware that the output from computeGSD is the voxel,
%     please multiply by the resolution in micron for maximum grain
radius and
%     multiply by resolution.^3 for grain volume
%     grainProp       = grainProp.*dx./1000;

%   Revision 1: May 2018 Nattavadee Srisutthiyakorn

%% Program
if (~exist('binEdge', 'var'))
    % Prefixed Histogram bin in mm due to the plot nature in log scale
    % for grain diameter
    binEdge = fliplr([4.0000 3.3600 2.8300 2.3800 2.0000 1.6800 1.4100
1.1900 1.0000 ...
           0.8500 0.7100 0.6000 0.5000 0.4200 0.3500 0.2970 0.2500
0.2100 ...
           0.1770 0.1490 0.1250 0.1050 0.0880 0.0740 0.0620 0.0530
0.0440 ...
           0.0370 0.0310 0.0260 0.0220 0.0190 0.0160 0.0130 0.0110
0.0093 ...
           0.0078 0.0065 0.0055 0.0046 0.0039 0.0033 0.0028 0.0023
0.0019 ...
           0.0016 0.0014 0.0012 0.0010]);
end
if (~exist('gridPC', 'var'))
    maxPixel     = max(max(grainCentroid));
    maxNumGrain     = length(grainProp);
    nSampling    = round((maxNumGrain./10).^(1/3)); % The total of
nSampling*nSampling samples for 2-D and
                      % and nSampling*nSampling*nSampling samples for
3-D will be taken from point count
    gridPC       = linspace(1, maxPixel, nSampling);
    gridPC       = round(gridPC);
end

% Create the grid
[x, y, z]        = meshgrid(gridPC);
gridCentroid     = [x(:) y(:) z(:)];
idxSelectedGrain = dsearchn(grainCentroid, gridCentroid);
```

```
% Select only the index
grainProp       = grainProp(idxSelectedGrain);

% Sort data into histogram bin
nEdge           = length(binEdge);
nBin            = nEdge - 1;
histPC          = zeros(nEdge - 1, 1);
binCenter(1)    = binEdge(1);

for iEdge = 1:nEdge - 1
    [index{iEdge}] = find(and(grainProp <= binEdge(iEdge + 1),...
                             grainProp  >  binEdge(iEdge)));
    binCenter(iEdge)   = (binEdge(iEdge + 1) + binEdge(iEdge))./2;
end

for iBin = 1:nBin
    histPC(iBin)   = length(index{iBin});
end

% Normalized to get the PDF
nGrainCheck = sum(histPC);
histPC = histPC./nGrainCheck;

% Export the statistics
statGSD.mean = mean(grainProp);
statGSD.std  = std(grainProp);

% Find the cumulative density function
% histCDF = cumsum((histFB))./sum(histFB);
```

Bibliography

Adler, P. M., C. G. Jacquin, and J. A. Quiblier, 1990, Flow in simulated porous media, 691–712.

Ahmadi, M. M., S. Mohammadi, and A. N. Hayati, 2011, Analytical derivation of tortuosity and permeability of monosized spheres: a volume averaging approach., Physical review. E, Statistical, nonlinear, and soft matter physics, **83**, no. 2 Pt 2, 026312.

Andrä, H. et al., 2013a, Digital rock physics benchmarks—Part I: Imaging and segmentation, Computers & Geosciences, **50**, 25–32.

Andrä, H. et al., 2013b, Digital rock physics benchmarks—part II: Computing effective properties, Computers & Geosciences, **50**, 33–43.

Arch, J., and A. Maltman, 1990, Anisotropic permeability and tortuosity in deformed wet sediments, JOURNAL OF GEOPHYSICAL RESEARCH, **95**, no. 10, 9035–9045.

Arns, C. H., M. A. Knackstedt, W. V. Pinczewski, and E. E. J. Garboczi, 2002, Computation of linear elastic properties from microtomographic images: Methodology and agreement between theory and experiment, Geophysics, **67**, no. 5, 1396.

Bear, J., 1988, Dynamics of fluids in porous media.

Berryman, J. G., and S. C. Blair, 1986, Use of digital image analysis to estimate fluid permeability of porous materials: Application of two-point correlation functions, Journal of Applied Physics, **60**, no. 6, 1930–1938.

Carman, P. C., 1937, Fluid flow through granular beds, Chemical Engineering Research and Design, **75**, S32–S48.

Carrier, W. D., 2003, Goodbye, Hazen; Hello, Kozeny-Carman, Journal of Geotechnical and Geoenvironmental Engineering, **129**, no. 11, 1054–1056.

Chen, H., S. Chen, and W. Matthaeus, 1992, Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method, Physical Review A.

Chilingar, G. V, 1979, Principles of Sedimentology, Wiley, 184–185.

Clennell, M. B., 1997, Tortuosity: a guide through the maze, Geological Society, London, Special Publications, **122**, no. 1, 299–344.

Cruz-Orive, L. M., 1983, Distribution-free estimation of sphere size distributions from slabs showing overprojection and truncation, with a review of previous methods, Journal of Microscopy, **131**, no. 3, 265–290.

Dullien, F., 1991, Porous media: fluid transport and pore structure, Academic press.

Dvorkin, J., 2009, Kozeny-Carman Equation Revisited.

Dvorkin, J., 2008, Some exact solutions for viscous fluid flow.

Dvorkin, J., M. Armbruster, C. Baldwin, Q. Fang, N. Derzhi, C. Gomez, B. Nur, A. Nur, and Y. Mu, 2008, The future of rock physics: Computational methods vs. lab testing, First Break, **26**, no. 9, 63–68.

Dvorkin, J., N. Derzhi, E. Diaz, and Q. Fang, 2011, Relevance of computational rock physics, Geophysics, **76**, no. 5, E141–E153.

Dvorkin, J., Q. Fang, and N. Derzhi, 2012, Etudes in computational rock physics: Alterations and benchmarking, GEOPHYSICS, **77**, no. 3, D45–D52.

Exner, H. E., 1972, Analysis of Grain- and Particle-Size Distributions in Metallic Materials, International Materials Reviews, **17**, no. 1, 25–42.

Finney, J. L., 1970, Random Packings and the Structure of Simple Liquids. I. The Geometry of Random Close Packing, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, **319**, no. 1539, 479–493.

Flavio S. Anselmetti, 2 Stefan Luthi, 1998, Quantitative Characterization of Carbonate Pore Systems by Digital Image Analysis.

Fredrich, J. T. J., D. D. R. Noble, R. R. M. O'Connor, W. Lindquist, D. D. R. Noble, and R. R. M. O'Connor, 1999, Development, implementation, and experimental validation of the Lattice Boltzmann method for modeling three-dimensional complex flows.

Garboczi, E., 1998, Finite element and finite difference programs for computing the linear electric and elastic properties of digital images of random materials, Building and Fire Research Laboratory, National Institute of Standards and Technology.

Ghanbarian, B., A. G. Hunt, R. P. Ewing, and M. Sahimi, 2013, Tortuosity in Porous Media: A Critical Review, Soil Science Society of America Journal, **77**, no. 5, 1461.

Gomez, C. T., J. Dvorkin, and T. Vanorio, 2010, Laboratory measurements of porosity, permeability, resistivity, and velocity on Fontainebleau sandstones, GEOPHYSICS,

75, no. 6, E191–E204.

Graton, L. C., and H. J. Fraser, 1935, Systematic Packing of Spheres: With Particular Relation to Porosity and Permeability, 785–909.

Guardiano, F., and R. Srivastava, 1993, Multivariate geostatistics: beyond bivariate moments.

Guéguen, Y., and V. Palciauskas, 1994, Introduction to the Physics of Rocks.

Gunstensen, A. K., D. H. Rothman, S. Zaleski, and G. Zanetti, 1991, Lattice Boltzmann model of immiscible fluids, Physical Review A, **43**, no. 8, 4320–4327.

Jain, A. K., 2010, Data clustering: 50 years beyond K-means, Pattern Recognition Letters, **31**, no. 8, 651–666.

Jin, G., T. W. Patzek, and D. B. Silin, 2003, Physics-based Reconstruction of Sedimentary Rocks, *in* SPE Western Regional/AAPG Pacific Section Joint Meeting, Society of Petroleum Engineers.

Jin, G., T. W. Patzek, and D. B. Silin, 2008, Reconstruction of Sedimentary Rock Based on Mechanical Properties, Lawrence Berkeley National Laboratory.

Jin, G., C. Torres-Verdín, and E. Toumelin, 2009, Comparison of NMR simulations of porous media derived from analytical and voxelized representations, Journal of Magnetic Resonance, **200**, no. 2, 313–320.

Kainourgiakis, M. E., E. S. Kikkinides, A. Galani, G. C. Charalambopoulou, and A. K. Stubos, 2005, Digitally reconstructed porous media: Transport and sorption properties, *in* Upscaling Multiphase Flow in Porous Media: From Pore to Core and Beyond, 43–

62.

Keehm, Y., 2003, Computational rock physics: transport properties in porous media and applications, Stanford University.

Keehm, Y., 2004, Permeability prediction from thin sections: 3D reconstruction and Lattice-Boltzmann flow simulation, Geophysical Research Letters, **31**, no. 4, L04606.

Keehm, Y., and W. J. Bosl, 2003, Comparison Of Different Lattice-Boltzmann Flow Simulation Implementations: Efficiency, Convergence And Stability.

Keehm, Y., T. Mukerji, and A. Nur, 2001, Computational rock physics at the pore scale: Transport properties and diagenesis in realistic pore geometries, The Leading Edge.

Konert, M., and J. Vandenberghe, 1997, Comparison of laser grain size analysis with pipette and sieve analysis: A solution for the underestimation of the clay fraction, 523–535.

Koponen, A., M. Kataja, and J. Timonen, 1997, Permeability and effective porosity of porous media, Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics, **56**, no. 3, 3319–3325.

Ladd, A., 1994, Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation, Journal of Fluid Mechanics.

Legland, D., K. Kiêu, and M. Devaux, 2007, Computation of Minkowski measures on 2D and 3D binary images, Image Analysis & Stereology.

Lehmann, P., M. Berchtold, B. Ahrenholz, J. Tölke, A. Kaestner, M. Krafczyk, H. Flühler, and H. R. Künsch, 2008, Impact of geometrical properties on permeability and fluid phase distribution in porous media, Advances in Water Resources, **31**, no. 9, 1188–1204.

Lock, P. A., X. Jing, R. W. Zimmerman, and E. M. Schlueter, 2002, Predicting the permeability of sandstone from image analysis of pore structure, Journal of Applied Physics, **92**, no. 10, 6311–6319.

Manwart, C., S. Torquato, and R. Hilfer, 2000, Stochastic reconstruction of sandstones, Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics, **62**, no. 1 B, 893–899.

Mariethoz, P. G., and P. J. Caers, 2014, Multiple-point Geostatistics: Stochastic Modeling with Training Images, Wiley.

Mariethoz, G., P. Renard, and J. Straubhaar, 2010, The direct sampling method to perform multiple-point geostatistical simulations, Water Resources Research, **46**, no. 11.

Mason, G., and N. R. Morrow, 1991, Capillary behavior of a perfectly wetting liquid in irregular triangular tubes, 262–274.

Mauret, E., and M. Renaud, 1997, Transport phenomena in multi-particle systems - I. Limits of applicability of capillary model in high voidage beds-application to fixed beds of fibers and fluidized beds of spheres, Chemical Engineering Science, **52**, no. 11, 1807–1817.

Mavko, G., T. Mukerji, and J. Dvorkin, 2009a, The rock physics handbook: Tools for seismic analysis of porous media, Cambridge University Press.

Mavko, G., T. Mukerji, and J. Dvorkin, 2009b, The rock physics handbook, Cambridge

University press.

Mavko, G., and A. Nur, 1997, The effect of a percolation threshold in the Kozeny-Carman relation, GEOPHYSICS, **62**, no. 5, 1480–1482.

Mota, M., J. A. Teixeira, W. R. Bowen, and A. Yelshin, 2001, Binary spherical particle mixed beds: porosity and permeability relationship measurement, 101–106.

Ohser, J., and K. Sandau, 2000, Considerations About the Estimation ofthe Size Distribution in Wicksell's Corpuscle Problem, *in* Statistical Physics and Spatial Statistics, 185–202.

Okabe, H., and M. J. Blunt, 2005, Pore space reconstruction using multiple-point statistics, Journal of Petroleum Science and Engineering, **46**, no. 1–2, 121–137.

Okabe, H., and M. J. Blunt, 2004, Prediction of permeability for porous media reconstructed using multiple-point statistics, Physical Review E - Statistical, Nonlinear, and Soft Matter Physics, **70**, no. 6 2.

Øren, P. E., and S. Bakke, 2002, Process based reconstruction of sandstones and prediction of transport properties, Transport in Porous Media, **46**, no. 2–3, 311–343.

Osserman, R., 1987, A strong form of the isoperimetric inequality in R n, Complex Variables, Theory and Application: An International Journal, **9**, no. 2–3, 241–249.

Ozgumus, T., M. Mobedi, and U. Ozkol, 2014, Determination of Kozeny constant based on porosity and pore to throat size ratio in porous medium with rectangular rods, Engineering Applications of Computational Fluid Mechanics, **8**, no. 2, 308–318.

Patzek, T. W., and J. G. Kristensen, 2001, Shape Factor Correlations of Hydraulic

Conductance in Noncircular Capillaries., Journal of colloid and interface science, **236**, no. 2, 305–317.

Pech, D., 1984, Etude de la perméabilité de lits compressibles constitués de copeaux de bois partiellement destructurés, These de 36me cycle.

Du Plessis, J. P., and J. H. Masliyah, 1991, Flow through isotropic granular porous media, Transport in Porous Media, **6**, no. 3, 207–221.

Richa, 2010, Preservation of transport properties trends: computational rock physics approach, Stanford University.

Sain, R., 2011, Numerical simulation of pore-scale heterogeneity and its effects on elastic, electrical, and transport properties, Stanford University.

Saxena, N., 2014, The Impact of Grain-Scale Elastic and Viscoelastic Changes on Seismic Wave Propagation, Stanford University.

Saxena, N., and G. Mavko, 2016, Estimating elastic moduli of rocks from thin sections: Digital rock study of 3D properties from 2D images, Computers and Geosciences, **88**, 9–21.

Saxena, N., G. Mavko, R. Hofmann, and N. Srisutthiyakorn, 2017, Estimating permeability from thin sections without reconstruction: Digital rock study of 3D properties from 2D images, Computers and Geosciences, **102**, 79–99.

Sezgin, M., and B. Sankur, 2004, Survey over image thresholding techniques and quantitative performance evaluation, 146.

Shepard, J. S., 1993, Using a fractal model to compute the hydraulic conductivity function, Soil Science Society of America Journal, **57**, no. 2, 300–306.

Silin, D. B., G. D. Jin, and T. W. Patzek, 2004, Robust determination of the pore-space morphology in sedimentary rocks, Journal of Petroleum Geology, **56**, no. 5, 69–70.

Silverman, A. B. W., M. C. Jones, J. D. Wilson, D. W. Nychka, S. Journal, R. Statistical, S. Series, and B. B. W. Silvermant, 1990, A Smoothed EM Approach to Indirect Estimation Problems , with Particular , Reference to Stereology and Emission Tomography, Journal of Royal Statistical Society, **52**, no. 2, 271–324.

Sisavath, S., X. D. Jing, and R. W. Zimmerman, 2000, Effect of stress on the hydraulic conductivity of rock pores, Physics and Chemistry of the Earth, Part A: Solid Earth and Geodesy, **25**, no. 2, 163–168.

Srisutthiyakorn, N., and G. M. Mavko, 2017, What is the role of tortuosity in the Kozeny-Carman equation?, Interpretation, **5**, no. 1, SB57-SB67.

Stoyan, D., W. S. Kendall, and J. Mecke, 1995, Stochastic geometry and its applications.

Strebelle, S., 2002, Conditional simulation of complex geological structures using multiple-point statistics, Mathematical Geology, **34**, no. 1, 1–21.

Strebelle, S., K. Payrazyan, and J. Caers, 2003, Modeling of a Deepwater Turbidite Reservoir Conditional to Seismic Data Using Principal Component Analysis and Multiple-Point Geostatistics.

T Bourbié, O Coussy, B. Z., 1987, Acoustics of porous media.

Tchelepi, H., 2015, Energy 221 Multiphase Flow in Porous Media, Stanford University.

Torquato, S., 2002, Statistical Description of Microstructures, Annual review of materials research, **32**, no. 1, 77–111.

Verberg, R., and A. Ladd, 1999, Simulation of low-Reynolds-number flow via a time-independent lattice-Boltzmann method, methods.

Visher, G. S., 1969, Grain Size Distributions and Depositional Processes, Journal of sedimentary petrology, **39**, no. 3, 1074–1106.

Vogel, H. J., U. Weller, and S. Schlüter, 2010, Quantification of soil structure based on Minkowski functions, Computers and Geosciences, **36**, no. 10, 1236–1245.

Wentworth, C. K., 1922, A Scale of Grade and Class Terms for Clastic Sediments, The Journal of Geology, **30**, no. 5, 377–392.

Wicksell, S. D., 1925, Wicksell_The corpscle problem a mathematical study of a biometric problem.pdf, Biometrika, **17**, no. 1/2, 84–99.

Wu, K., M. I. J. Dijke, G. D. Couples, Z. Jiang, J. Ma, K. S. Sorbie, J. Crawford, I. Young, and X. Zhang, 2006, 3D Stochastic Modelling of Heterogeneous Porous Media – Applications to Reservoir Rocks, 443–467.

Yeong, C., and S. Torquato, 1998, Reconstructing random media. II. Three-dimensional media from two-dimensional cuts, 224–233.

Zhang, D., and R. Zhang, 2000, Pore scale study of flow in porous media: Scale dependency, REV, and statistical REV, Geophysical research ….

Zomorodian, A., and G. Carlsson, 2005, Computing persistent homology, Discrete and Computational Geometry, **33**, no. 2, 249–274.